

El manual para el clustering con openMosix

por **Miquel Catalán i Coït** ≡ miKeL a.k.a.mc2
inspirado en el HOWTO de Kris Buytaert





Este manual está dedicado a todos aquellos que han hecho, hacen y harán que haya algo que documentar. A todos los desarrolladores del proyecto openMosix, muchas gracias.

Menciones especiales para:

Moshe Bar principal desarrollador, autor del MFS y DFSA

Louis Zechtzer

Martin Høy

Brian Pontz

Bruce Knox

Matthew Brichacek

Matthias Rechenburg

Maurizio Davini

Michael Farnbach

Mark Veltzer

Muli Ben Yehuda (a.k.a. mulix)

David Santo Orcero

openMosix -because they said it couldn't be done
openMosix, porque dijeron que no podía hacerse

Este COMO ha estado posible gracias a las colaboraciones de:

Ana Pérez Arteaga correcciones ortográficas y léxico-semánticas

C. W. Strommer traducción de las PMF (preguntas más frecuentes)

Jaime Perea capítulo sobre PCMCIA (openMosix para ordenadores portátiles)

Marcelo Stutz capítulo sobre el acceso a openMosixView con `ssh`

y **David Santo Orcero** aportaciones desde su inmenso archivo personal

Todos nosotros nos hemos esforzado y lo haremos para que a usuarios como tú les sea útil esta guía, por ello te animamos a hacernos llegar cualquier ambigüedad, error o consejo para poder mejorarla.

También a tí que has sabido ver el poder de la comunidad libre y vas a convertir tus PCs en un super-computador, gracias.

Índice de figuras

1.	openMosixview: Aplicación principal	29
2.	openMosixview: Propiedades de los nodos	30
3.	openMosixview: Ejecución avanzada	31
4.	openMosixprocs: Administración de procesos	33
5.	openMosixprocs: La ventana de migrado de un proceso(1)	34
6.	openMosixprocs: La ventana de migrado de un proceso(2)	35
7.	openMosixanalyzer. Historial de actividad de procesamiento del cluster . . .	37
8.	openMosixanalyzer. Estadísticas de los nodos	37
9.	openMosixanalyzer. Historial de actividad de memoria de nuestro cluster . .	38
10.	openMosixhistory. Un historial de los procesos ejecutados	39

Índice de cuadros

1.	Cambiando los parámetros en <code>/proc/hpc</code>	19
2.	Binarios en <code>/proc/hpc/admin</code>	19
3.	Escribiendo un 1 en <code>/proc/hpc/decay</code>	20
4.	Información adicional sobre los procesos locales	20
5.	Información de los otros nodos	20
6.	Parámetros de <code>mosctl</code> con más detalle	21
7.	Parámetros adicionales para <code>mosrun</code>	21
8.	Condiciones de inicio avanzadas para procesos	32

openMosix y el mundo de la programación libre avanzan a pasos agigantados. Es un mundo donde el sol nunca se pone y donde nadie entiende de fronteras, razas ni religiones. Lo que cuenta es el código. Y llega en gran cantidad, y de gran calidad.

Moshe Bar

Puedes encontrar este manual y sus futuras revisiones en internet en la dirección <http://w3.akamc2.net>

Índice

1. Introducción	1
1.1. Sobre este documento	2
1.2. Limitación de responsabilidad	2
1.3. Política de distribución	3
1.4. Nuevas versiones de este documento	3
1.5. Mantenimiento	3
2. ¿Qué es realmente openMosix?	4
2.1. Una muy breve introducción al <i>clustering</i>	5
2.1.1. HPC, Fail-over y Load-balancing	5
2.1.2. Mainframes y supercomputadoras vs. clusters	5
2.1.3. Modelos de clusters NUMA, PVM y MPI	6
2.2. Una aproximación histórica	6
2.2.1. Desarrollo histórico	6
2.2.2. openMosix != MOSIX	7
2.3. openMosix en acción: un ejemplo	8
3. Características de openMosix	9
3.1. Pros de openMosix	10
3.2. Contras de openMosix	10
3.3. Subsistemas de openMosix	10
3.3.1. Mosix File System (MFS)	10
3.3.2. Migración de procesos	10
3.3.3. Direct File System Access (DFSA)	10
3.3.4. Memory ushering	11
3.4. El algoritmo de migración	11
3.4.1. El nodo raíz	11

3.4.2.	El mecanismo de migrado	11
3.4.3.	¿ Cuándo podrá migrar un proceso?	12
3.4.4.	La comunicación entre las dos áreas	12
4.	Requerimientos y planteamientos	14
4.1.	Requerimientos hardware	15
4.2.	Lineas básicas en la configuración del hardware	15
4.3.	Requerimientos software	15
4.4.	Planteamientos de tu cluster	15
5.	Instalación de un cluster openMosix	16
5.1.	Obteniendo fuentes y paquetes	17
5.1.1.	Parcheando el kernel de linux	17
5.1.2.	Opciones en el nuevo kernel parcheado	17
5.1.3.	Las herramientas de usuario	17
5.1.4.	Paquetes RPM	17
5.1.5.	Debian	17
5.1.6.	Fuentes	17
5.2.	Compilaciones necesarias	17
5.2.1.	Compilación del nuevo kernel	17
5.2.2.	Configurando las herramientas de usuario	17
5.3.	Instalaciones del programario	17
6.	Administración del cluster	18
6.1.	Administración básica	19
6.2.	Configuración	19
6.3.	Las herramientas de área de usuario	20
6.4.	Detección automática de nodos	22
6.4.1.	Compilaciones	23

6.4.2. Problemas	23
7. Ajustes en el cluster	25
8. openMosixview	26
8.1. Introducción	27
8.2. Instalación	27
8.2.1. Instalación de los paquetes RPM	27
8.2.2. Instalación de las fuentes	28
8.2.3. Script de configuración automático	28
8.2.4. Compilación manual	28
8.3. Utilizando openMosixview	29
8.3.1. Aplicación principal	29
8.3.2. La ventana de configuración	30
8.3.3. Ejecución avanzada	31
8.3.4. La línea de comandos	31
8.3.5. El host-chooser	31
8.3.6. El host-chooser paralelo	32
8.4. openMosixprocs	33
8.4.1. Introducción	33
8.4.2. La ventana de migración	34
8.4.3. Administrando procesos remotamente	35
8.5. openMosixcollector	35
8.6. openMosixanalyzer	36
8.6.1. Una visión general de la carga del sistema	36
8.6.2. Estadísticas sobre los nodos del cluster	38
8.6.3. Monitoraje de la memoria	38
8.7. openMosixhistory	39
8.8. openMosixview + SSH2	40

8.9. FAQ de openMosixview -preguntas más frecuentes	41
9. Casos especiales	44
9.1. Laptops y targetas PCMCIA	45
9.2. Nodos sin disco duro	46
10. Problemas más comunes	51
10.1. No veo todos los nodos	52
10.2. FAQ de openMosix -preguntas más frecuentes	52
10.2.1. General	52
10.2.2. Obteniendo, compilando, instalando y funcionando con openMosix	53
10.2.3. Preguntas del kernel (núcleo)	54
10.2.4. Sistema de ficheros	54
10.2.5. Programando openMosix	55
10.2.6. Recursos	55
10.2.7. Miscelánea	56
11. Precauciones	58
11.1. Procesos bloqueados	59
11.2. Escogiendo tus procesos	59
11.3. Java y openMosix	59
12. Para más información	60
13. APÉNDICE A: Aplicaciones funcionando	62
13.1. Programas que funcionan	63
13.2. Programas que NO funcionan	63
14. APÉNDICE B: Acrónimos	64

1. Introducción

Primero fue MOSIX, ahora es openMosix, un proyecto mucho más interesante no sólo desde un punto de vista técnico sino porque se han mejorado los términos de la licencia que mantenía a Mosix bajo código propietario.

Este COMO (o manual) está dirigido a conocer el proyecto openMosix y no el MOSIX por la simple razón que el primero tiene un sector de usuarios mucho más amplio y con mayores garantías de crecer en los próximos tiempos (Moshe Bar, el principal desarrollador del proyecto openMosix, estima que el 97% de los usuarios de la antigua comunidad Mosix ya han migrado a openMosix). La redacción de los principales capítulos es la traducción del HOWTO escrito por Kris Buytaert para usuarios de Mosix y openMosix. Esta dualidad temática aportaba confusión y obsolescencia así que creo haber mejorado y actualizado bastante capítulos como el de la *Instalación*, donde he profundizado para permitir que cualquier usuario novel de Linux pueda arreglárselas con openMosix. En ésta, mi propuesta en castellano del HOWTO oficial, dejaré de lado Mosix por considerarlo una tecnología obsoleta y aún más cuando ya lleva, a día de hoy, cerca de un año sin ser actualizada. Nada una ya a Mosix y openMosix, e intentar buscarles parecidos resultará, como los grandes avances en el proyecto demuestran, cada vez ms difícil.

Para más información sobre el HOWTO original de Kris Buytaert puedes dirigirte a su sitio web (referenciado más adelante).

1.1. Sobre este documento

Este documento te dará una breve descripción de **openMosix, un paquete software que posibilita que una red de computadoras basadas en GNU/Linux funcionen como un cluster** (además será SSI, Single System Image, como se verá).

A lo largo de este camino que empezaremos juntos se introducirán conceptos como la computación paralela, super-computación, breves tutoriales para programas que tengan utilidades especiales para las posibilidades que openMosix pueda ofrecerte, e incluso un repaso histórico sobre los inicios del clustering como alternativa para la super-computación (ver Apéndice B).

Asimismo este manual amplía su contenido proporcionando documentación para las distintas distribuciones, básicamente considerando las que se basan en Debian (y sus paquetes *.deb*) y las basadas en los paquetes de RedHat (*.rpm*). Obviamente la compilación de los fuentes será una alternativa constante durante todo el proceso.

Kris Buytaert escribió el HOWTO original en febrero de 2002, cuando Scot Stevenson buscaba a alguien para llevar a cabo este trabajo de documentación. En aquella versión original se hacía muchas veces a referencia Mosix cuando debía leerse openMosix, así que me tomaré la libertad de focalizarlo todo hacia el segundo, en pro de una mejor inteligibilidad para el lector y por las causas anteriormente citadas.

1.2. Limitación de responsabilidad

Utilice la información contenida en este documento siendo responsable del riesgo que puedan correr sus equipos. Yo mismo y el equipo de colaboradores repudiamos cualquier responsabilidad sobre las consecuencias que el seguimiento de estos contenidos puedan provocar.

El uso de los ejemplos y conceptos expuestos corren a cargo del lector.

Es recomendable hacer copias de seguridad (*backups*) de su sistema antes de iniciar cualquier instalación, ya que el trabajo desde `root` (administrador de su equipo linux) puede provocar pérdidas y/o modificaciones irreversibles de su información.

Asimismo, para su mayor comodidad en caso de dar un mal paso, es recomendable hacer *backups* regularmente.

1.3. Política de distribución

Copyright © 2002 by Kris Buytaert and Scot W.Stevenson, para el documento original. Para el presente manual no existe licencia alguna, aunque agradecemos de antemano (el equipo de colaboradores y yo mismo) la notificación de cualquier uso que pueda darsele.

1.4. Nuevas versiones de este documento

Las versiones oficiales (tanto las originales en inglés como sus diversas traducciones) de este documento serán hospedadas en *The Linux Documentation Project*¹.

Los borradores del documento original se encontrarán en la web de Kris Buytaert² en el sub-directorio apropiado. Los cambios en este documento normalmente serán anunciados en las listas de distribución de openMosix.

Los posibles cambios en ésta, la versión en castellano, serán igualmente anunciados en la citada lista y podrás obtenerla en mi sitio web³.

1.5. Mantenimiento

Actualmente este manual está mantenido por miKeL a.k.a.mc2 (Miquel Catalán i Coït), por favor manda tus dudas o preguntas a la dirección de correo electrónico que encontrarás en su sitio web.

Envía también tus comentarios, preguntas, errores que puedas encontrar y por supuesto todos los elogios que creas oportunos, al autor.

¹<http://www.tldp.org>

²<http://howto.ipng.be/Mosix-HOWTO/index.html>

³<http://w3.akamc2.net>

Para dudas concretas sobre la tecnología openMosix, por favor dirígete a las listas de correo del proyecto (ver capítulo *Para más información*).

2. ¿Qué es realmente openMosix?

2.1. Una muy breve introducción al *clustering*

La mayor parte del tiempo tu computadora permanece ociosa. Si lanzas un programa de monitorización del sistema como `xload` o `top`, verás probablemente que la lectura de la carga de tu procesador permanece generalmente por debajo del 10 %.

Si tienes al alcance varias computadoras los resultados serán los mismos ya que no podrás interactuar con más de una de ellas al mismo tiempo. Desafortunadamente cuando realmente necesites potencia computacional (como por ejemplo para comprimir un fichero *Ogg Vorbis*, o para una gran compilación) no podrás disponer de la potencia conjunta que te proporcionarían todas ellas como un todo.

La idea que se esconde en el trasfondo del *clustering* es precisamente poder contar con todos los recursos que puedan brindarte el conjunto de computadoras de que puedas disponer para poder aprovechar aquellos que permanecen sin usar, básicamente en otras computadoras.

La unidad básica de un cluster es una computadora simple, también denominada **nodo**. Los clusters pueden crecer en tamaño (o mejor dicho, pueden *escalar*) añadiendo más máquinas.

Un cluster como un todo puede ser más potente que la más veloz de las máquinas con las que cuenta, factor que estará ligado irremediablemente a la velocidad de conexión con la que hemos construido las comunicaciones entre nodos.

Además, el sistema operativo del cluster puede hacer un mejor uso del hardware disponible en respuesta al cambio de condiciones. Esto produce un reto a un cluster heterogéneo (compuesto por máquinas de diferente arquitectura) tal como iremos viendo paso a paso.

2.1.1. HPC, Fail-over y Load-balancing

Básicamente existen tres tipos de clusters: *Fail-over*, *Load-balancing* y *HIGH Performance Computing*.

Los clusters **Fail-over** consisten en dos o más computadoras conectadas en red con una conexión *heartbeat* separada entre ellas. La conexión *heartbeat* entre las computadoras es usualmente utilizada para monitorear cuál de todos los servicios está en uso, así como la sustitución de una máquina por otra cuando uno de sus servicios haya caído.

El concepto en los **Load-balancing** se basa en que cuando haya una petición entrante al servidor web, el cluster verifica cuál de las máquinas disponibles posee mayores recursos libres, para luego asignarle el trabajo pertinente.

Actualmente un cluster *load-balancing* es también *fail-over* con el extra del balanceo de la carga y a menudo con mayor número de nodos.

La última variación en el clustering son los **High Performance Computing**.

Estas máquinas han estado configuradas especialmente para centros de datos que requieren una potencia de computación extrema.

Los clusters **Beowulf** han sido diseñados específicamente para estas tareas de tipo masivo, teniendo en contrapartida otras limitaciones que no lo hacen tan accesible para el usuario como un openMosix.

2.1.2. Mainframes y supercomputadoras vs. clusters

Tradicionalmente los *mainframes* y las supercomputadoras han estado construidas solamente por unos fabricantes muy concretos y para un colectivo elitista que necesitaba gran potencia de cálculo, como pueden ser empresas o universidades.

Pero muchos colectivos no pueden afrontar el costo económico que supone adquirir una máquina de estas características, y aquí es donde toma la máxima importancia la idea de poder disponer de esa potencia de cálculo, pero a un precio muy inferior.

El concepto de cluster nació cuando los pioneros de la supercomputación intentaban difundir diferentes procesos entre varias computadoras, para luego poder recoger los resultados que dichos procesos debían producir. Con un hardware más barato y fácil de conseguir se pudo perfilar que podrían conseguirse resultados muy parecidos a los obtenidos con aquellas máquinas mucho más costosas, como se ha venido probando desde entonces.

2.1.3. Modelos de clusters NUMA, PVM y MPI

Hay diferentes formas de hacer procesamiento paralelo, entre las más conocidas y usadas podemos destacar NUMA, PVM y MPI.

Las máquinas de tipo NUMA (Non-Uniform Memory Access) tienen acceso compartido a la memoria donde pueden ejecutar su código de programa. En el kernel de Linux hay ya implementado NUMA, que hace variar el número de accesos a las diferentes regiones de memoria.

PVM / MPI son herramientas que han estado ampliamente utilizadas y son muy conocidas por la gente que entiende de supercomputación basada en GNU/Linux.

MPI es el estándar abierto de bibliotecas de paso de mensajes. MPICH es una de las implementaciones más usadas de MPI, tras MPICH se puede encontrar LAM, otra implementación basada en MPI también con bibliotecas de código abierto.

PVM (Parallel Virtual Machine) es un primo de MPI que también es ampliamente usado para funcionar en entornos Beowulf.

PVM habita en el espacio de usuario y tiene la ventaja que no hacen falta modificaciones en el kernel de Linux, básicamente cada usuario con derechos suficientes puede ejecutar PVM.

2.2. Una aproximación histórica

2.2.1. Desarrollo histórico

Algunos rumores hablaban que MOSIX venía de Moshe Unix. Inicialmente Mosix empezó siendo una aplicación para BSD/OS 3.0, como podemos leer en este email:

```
Announcing M06 for BSD/OS 3.0
Oren Laadan (orenl@cs.huji.ac.il)
Tue, 9 Sep 1997 19:50:12 +0300 (IDT)
```

Hi:

We are pleased to announce the availability of M06 Version 3.0 Release 1.04 (beta-4) - compatible with BSD/OS 3.0, patch level K300-001 through M300-029.

M06 is a 6 processor version of the MOSIX multicomputer enhancements of BSD/OS for a PC Cluster. If you have 2 to 6 PC's connected by a LAN, you can experience truly multi-computing environment by using the M06 enhancements.

The M06 Distribution

M06 is available either in "source" or "binary" distribution. It is installed as a patch to BSD/OS, using an interactive installation script.

M06 is available at <http://www.cnds.jhu.edu/mirrors/mosix/>
or at our site: <http://www.cs.huji.ac.il/mosix/>

Main highlights of the current release:

- Memory ushering (depletion prevention) by process migration.
- Improved installation procedure.
- Enhanced migration control.
- Improved administration tools.
- More user utilities.
- More documentation and new man pages.
- Dynamic configurations.

Please send feedback and comments to mosix@cs.huji.ac.il.

La plataforma GNU/Linux para el desarrollo de posteriores versiones fue elegida en la séptima reencarnación, en 1999.

A principios de 1999 Mosix M06 fue lanzado para el kernel de Linux 2.2.1.

Entre finales de 2001 e inicios de 2002 nació openMosix, la versión de código abierto, de forma separada.

2.2.2. openMosix != MOSIX

openMosix en principio tenía que ser una ampliación a lo que años atrás ya se podía encontrar en www.mosix.org, respetando todo el trabajo llevado a cabo por el Prof. Barak y su equipo.

Moshe Bar estuvo ligado al proyecto Mosix, en la Universidad Hebrea de Jerusalem, durante bastantes años. Era el co-administrador del proyecto y el principal administrador de los asuntos comerciales de *Mosix company*.

Tras algunas diferencias de opinión sobre el futuro comercial de Mosix, Moshe Bar empezó un nuevo proyecto de *clustering* alzando la empresa *Qlusters, Inc.* en la que el profesor A. Barak⁴ decidió no participar ya que no quería poner Mosix bajo licencia GPL.

Como había una significativa base de usuarios clientes de la tecnología Mosix (unas 1000 instalaciones a lo ancho del planeta) Moshe Bar decidió continuar el desarrollo de Mosix pero bajo otro nombre, openMosix, totalmente bajo licencia GPL2.

openMosix es un parche (*patch*) para el kernel de linux que proporciona compatibilidad completa con el estandard de Linux para plataformas IA32. Actualmente se está trabajando para portarlo a IA64.

El algoritmo interno de balanceo de carga migra, transparentemente para el usuario, los procesos entre los nodos del cluster. La principal ventaja es una mejor compartición de recursos entre nodos, así como un mejor aprovechamiento de los mismos.

El cluster escoge por sí mismo la utilización óptima de los recursos que son necesarios en cada momento, y de forma automática.

Esta característica de migración transparente hace que el cluster funcione a todos los efectos como un gran sistema SMP (*Symmetric Multi Processing*) con varios procesadores disponibles. Su estabilidad ha sido ampliamente probada aunque todavía se está trabajando en diversas líneas para aumentar su eficiencia.

openMosix está respaldado y siendo desarrollado por personas muy competentes y respetadas en el mundo del *open source*, trabajando juntas en todo el mundo.

El punto fuerte de este proyecto es que intenta crear un estándar en el entorno del clustering para todo tipo de aplicaciones HPC.

openMosix tiene una página web⁵ con un árbol CVS⁶ y un par de listas de correo para los

⁴<http://www.cs.huji.ac.il/~amnon/>

⁵<http://www.openmosix.org>

⁶<http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/openmosix/>

desarrolladores y para los usuarios.

2.3. openMosix en acción: un ejemplo

Los clusters openMosix pueden adoptar varias formas. Para demostrarlo intentad imaginar que compartes el piso de estudiante con un chico adinerado que estudia ciencias de la computación. Imaginad también que tenéis las computadoras conectadas en red para formar un cluster openMosix.

Asume también que te encuentras convirtiendo ficheros de música desde tus CDs de audio a Ogg Vorbis para tu uso privado, cosa que resulta ser legal en tu país.

Tu compañero de habitación se encuentra trabajando en un proyecto de C++ que según dice podrá traer la paz mundial, pero en este justo momento está en el servicio cantando cosas ininteligibles, y evidentemente su computadora está a la espera de ser intervenida de nuevo.

Resulta que cuando inicias un programa de compresión, como puede ser `bladeenc` para convertir un prelude de Bach desde el fichero `.wav` al `.ogg`, las rutinas de openMosix en tu máquina comparan la carga de procesos en tu máquina y en la de tu compañero y deciden que es mejor migrar las tareas de compresión ya que el otro nodo es más potente debido a que el chico disponía de más medios económicos para poderse permitir una computadora más potente, a la vez que en ese momento permanece ociosa ya que no se encuentra frente a ella.

Así pues lo que normalmente en tu pentium233 tardaría varios minutos te das cuenta que ha terminado en pocos segundos.

Lo que ha ocurrido es que gran parte de la tarea ha sido ejecutada en el AMD AthlonXP de tu compañero, de forma transparente a ti.

Minutos después te encuentras escribiendo y tu compañero de habitación vuelve del servicio. Éste reanuda sus pruebas de compilación utilizando `pmake`, una versión del `make` optimizada para arquitecturas paralelas. Te das cuenta que openMosix está migrando hacia tu máquina algunos subprocesos con el fin de equilibrar la carga.

Esta configuración se llama *single-pool*: todas las computadoras están dispuestas como un único cluster. La ventaja o desventaja de esta disposición es que tu computadora es parte del *pool*: tus procesos serán ejecutados, al menos en parte, en otras computadoras, pudiendo atentar contra tu privacidad de datos. Evidentemente las tareas de los demás también podrán ser ejecutadas en la tuya.

3. Características de openMosix

3.1. Pros de openMosix

- No se requieren paquetes extra
- No son necesarias modificaciones en el código

3.2. Contras de openMosix

- Es dependiente del kernel
- No migra todos los procesos siempre, tiene limitaciones de funcionamiento
- Problemas con memoria compartida

Además los procesos con múltiples *threads* no ganan demasiada eficiencia

Tampoco se obtendrá mucha mejora cuando se ejecute un solo proceso, como por ejemplo el navegador.

3.3. Subsistemas de openMosix

Actualmente podemos dividir los parches de openMosix dentro del kernel en cuatro grandes subsistemas, veámoslos.

3.3.1. Mosix File System (MFS)

El primer y mayor subsistema (en cuanto a líneas de código) es MFS que te permite un acceso a sistemas de ficheros (FS) remotos (i.e. de cualquier otro nodo) si está localmente montado.

El sistema de ficheros de tu nodo y de los demás podrán ser montados en el directorio */mfs* y de esta forma se podrá, por ejemplo, acceder al directorio */home* del nodo 3 dentro del directorio */mfs/3/home* desde cualquier nodo del cluster.

3.3.2. Migración de procesos

Con openMosix se puede lanzar un proceso en una computadora y ver si se ejecuta en otra, en el seno del cluster.

Cada proceso tiene su único nodo raíz (UHN, *unique home node*) que se corresponde con el que lo ha generado.

El concepto de migración significa que un proceso se divide en dos partes: la parte del usuario y la del sistema.

La parte, o área, de usuario será movida al nodo remoto mientras el área de sistema espera en el raíz.

openMosix se encargará de establecer la comunicación entre estos 2 procesos.

3.3.3. Direct File System Access (DFSA)

openMosix proporciona MFS con la opción DFSA que permite acceso a todos los sistemas de ficheros, tanto locales como remotos. Para más información diríjase a la sección de *Sistema de ficheros* de las FAQs (preguntas más frecuentes) del presente documento.

3.3.4. Memory ushering

Este subsistema se encarga de migrar las tareas que superan la memoria disponible en el nodo en el que se ejecutan. Las tareas que superan dicho límite se migran forzosamente a un nodo destino de entre los nodos del cluster que tengan suficiente memoria como para ejecutar el proceso sin necesidad de hacer *swap* a disco, ahorrando así la gran pérdida de rendimiento que esto supone. El subsistema de *memory ushering* es un subsistema independiente del subsistema de equilibrado de carga, y por ello se le considera por separado.

3.4. El algoritmo de migración

De entre las propiedades compartidas entre Mosix y openMosix podemos destacar el mecanismo de migración, en el que puede migrar-se cualquiera proceso a cualquier nodo del cluster de forma completamente transparente al proceso migrado. La migración también puede ser automática: el algoritmo que lo implementa tiene una complejidad computacional del orden de $O(n)$, siendo n el número de nodos del cluster.

Para implementarlo openMosix utiliza el modelo *fork-and-forget*⁷, desarrollado en un principio dentro de Mosix para máquinas PDP11/45 empleadas en las fuerzas aéreas norteamericanas. La idea de este modelo es que la distribución de tareas en el cluster la determina openMosix de forma dinámica, conforme se van creando tareas. Cuando un nodo está demasiado cargado, y las tareas que se están ejecutando puedan migrar a cualquier otro nodo del cluster. Así desde que se ejecuta una tarea hasta que ésta muere, podrá migrar de un nodo a otro, sin que el proceso sufra mayores cambios.

Podríamos pensar que el comportamiento de un cluster openMosix es como una máquina NUMA, aunque estos clusters son mucho más baratos.

⁷hace referencia a que el sistema cuando reconoce un subproceso se encarga de ejecutarlo en otro nodo, en paralelo, sin ningun efecto ni notificación al propietario del mismo

3.4.1. El nodo raíz

Cada proceso ejecutado en el cluster tiene un único nodo raíz, como se ha visto. El nodo raíz es el nodo en el cual se lanza originalmente el proceso y donde éste empieza a ejecutarse.

Desde el punto de vista del espacio de procesos de las máquinas del cluster, cada proceso (con su correspondiente PID) parece ejecutarse en su nodo raíz. El nodo de ejecución puede ser el nodo raíz u otro diferente, hecho que da lugar a que el proceso no use un PID del nodo de ejecución, sino que el proceso migrado se ejecutará en éste como una hebra del kernel. La interacción con un proceso, por ejemplo enviarle señales desde cualquier otro proceso migrado, se puede realizar exclusivamente desde el nodo raíz.

El usuario que ejecuta un proceso en el cluster ha accedido al cluster desde el nodo raíz del proceso (puesto que ha logado en él). El propietario del proceso en cuestión tendrá control en todo momento del mismo como si se ejecutara localmente.

Por otra parte la migración y el retorno al nodo raíz de un proceso se puede realizar tanto desde el nodo raíz como desde el nodo dónde se ejecuta el proceso. Esta tarea la puede llevar a término el administrador de cualquiera de los dos sistemas.

3.4.2. El mecanismo de migrado

La migración de procesos en openMosix es completamente transparente. Esto significa que al proceso migrado no se le avisa de que ya no se ejecuta en su nodo de origen. Es más, este proceso migrado seguirá ejecutándose como si siguiera en el nodo origen: si escribiera o leyera al disco, lo haría en el nodo origen, hecho que supone leer o grabar remotamente en este nodo.

3.4.3. ¿ Cuándo podrá migrar un proceso?

Desgraciadamente, no todos los procesos pueden migrar en cualquiera circunstancia. El mecanismo de migración de procesos puede operar sobre cualquier tarea de un nodo sobre el que se cumplen algunas condiciones predeterminadas. Éstas son:

- el proceso no puede ejecutarse en modo de emulación VM86
- el proceso no puede ejecutar instrucciones en ensamblador propias de la máquina donde se lanza y que no tiene la máquina destino (en un cluster heterogéneo)
- el proceso no puede mapear memoria de un dispositivo a la RAM, ni acceder directamente a los registros de un dispositivo
- el proceso no puede usar segmentos de memoria compartida

Cumpliendo todas estas condiciones el proceso puede migrar y ejecutarse migrado. No obstante, como podemos sospechar, openMosix no adivina nada. openMosix no sabe a priori si alguno de los procesos que pueden migrar tendrán algunos de estos problemas.

Por esto en un principio openMosix migra todos los procesos que puedan hacerlo si por el momento cumplen todas las condiciones, y en caso de que algún proceso deje de cumplirlas, lo devuelve de nuevo a su nodo raíz para que se ejecute en él mientras no pueda migrar de nuevo.

Todo esto significa que mientras el proceso esté en modo de emulación VM86, mapee memoria de un dispositivo RAM, acceda a un registro o tenga reservado/bloqueado un puntero a un segmento de memoria compartida, el proceso se ejecutará en el nodo raíz, y cuando acabe la condición que lo bloquea volverá a migrar.

Con el uso de instrucciones asociadas a procesadores no compatibles entre ellos, openMosix tiene un comportamiento diferente: solo permitirá migrar a los procesadores que tengan la misma arquitectura.

3.4.4. La comunicación entre las dos áreas

Un aspecto importante en el que podemos tener interés es en cómo se realiza la comunicación entre el área de usuario y el área de kernel.

En algún momento, el proceso migrado puede necesitar hacer alguna llamada al sistema. Esta llamada se captura y se evalúa

- si puede ser ejecutada al nodo al que la tarea ha migrado, o
- si necesita ser lanzada en el nodo raíz del proceso migrado

Si la llamada puede ser lanzada al nodo dónde la tarea migrada se ejecuta, los accesos al kernel se hacen de forma local, es decir, que se atiende en el nodo dónde la tarea se ejecuta sin ninguna carga adicional a la red.

Por desgracia, las llamadas más comunes son las que se han de ejecutar forzosamente al nodo raíz, puesto que *hablan* con el hardware. Es el caso, por ejemplo, de una lectura o una escritura a disco. En este caso el subsistema de openMosix del nodo dónde se ejecuta la tarea contacta con el subsistema de openMosix del nodo raíz. Para enviarle la petición, así como todos los parámetros y los datos del nodo raíz que necesitará procesar.

El nodo raíz procesará la llamada y enviará de vuelta al nodo dónde se está ejecutando realmente el proceso migrado:

- el valor del éxito/fracaso de la llamada
- aquello que necesite saber para actualizar sus segmentos de datos, de pila y de *heap*
- el estado en el que estaría si se estuviera ejecutando el proceso al nodo raíz

Esta comunicación también puede ser generada por el nodo raíz. Es el caso, por ejemplo, del envío de una señal. El subsistema de openMosix del nodo raíz contacta con el subsistema de openMosix del nodo dónde el proceso migrado se ejecuta, y el avisa que ha ocurrido un evento asíncrono. El subsistema de openMosix del nodo dónde el proceso migrado se ejecuta parará el proceso migrado y el nodo raíz podrá empezar a atender el código del área del kernel que correspondería a la seal asíncrona.

Finalmente, una vez realizada toda el operativa necesaria de la área del kernel, el subsistema de openMosix del nodo raíz del proceso envía al nodo donde está ejecutándose realmente el proceso migrado el aviso detallado de la llamada, y todo aquello que el proceso necesita saber (anteriormente enumerado) cuando recibió la señal, y el proceso migrado finalmente recuperará el control.

Por todo esto el proceso migrado es como sí estuviera al nodo raíz y hubiera recibido la señal de éste. Tenemos un escenario muy simple donde el proceso se suspende esperando un recurso. Recordemos que la suspensión esperando un recurso se produce únicamente en área de kernel. Cuando se pide una página de disco o se espera un paquete de red se resuelto como en el primero caso comentado, es decir, como un llamada al kernel.

Este mecanismo de comunicación entre áreas es el que nos asegura que

- la migración sea completamente transparente tanto para el proceso que migra como para los procesos que cohabiten con el nodo raíz
- que el proceso no necesite ser reescrito para poder migrar, ni sea necesario conocer la topología del cluster para escribir una aplicación paralela

No obstante, en el caso de llamadas al kernel que tengan que ser enviadas forzosamente al nodo raíz, tendremos una sobrecarga adicional a la red debida a la transmisión constante de las llamadas al kernel y la recepción de sus valores de vuelta.

Destacamos especialmente esta sobrecarga en el acceso a sockets y el acceso a disco duro, que son las dos operaciones más importantes que se habrán de ejecutar en el nodo raíz y suponen una sobrecarga al proceso de comunicación entre la área de usuario migrada y la área de kernel del proceso migrado.

4. Requerimientos y planteamientos

4.1. Requerimientos hardware

Para la instalación básica de un cluster necesitaremos al menos dos computadoras conectadas en red. Podremos conectarlas mediante un cable cruzado entre las respectivas tarjetas de red, con un *hub* o con un *switch*.

Evidentemente cuanto más rápida sea la conexión entre máquinas, más eficaz será nuestro sistema global.

Actualmente *Fast Ethernet* es un estándar, permitiendo múltiples puertos en una máquina. *Gigabit Ethernet* es más cara y no es recomendable probar con ella sin antes haberse asegurado un correcto funcionamiento con *Fast Ethernet* y comprobar que realmente se necesita este extra en la velocidad de transferencia de datos entre nodos.

Siempre podremos hacer funcionar varias tarjetas *Fast* en cada nodo para asignarles luego la misma dirección (IP) y de esta forma poder obtener múltiples en la velocidad.

4.2. Líneas básicas en la configuración del hardware

Para poder configurar un *gran* cluster (refiriéndose al número de nodos) tendremos que pensar en ciertos aspectos, como por ejemplo dónde situar las máquinas, ya que tenerlas en medio de una oficina puede resultar incómodo en muchos aspectos. La mejor opción sería “raquearlas”.

El acondicionamiento de la sala donde deba situarse el cluster también es importante para evitar sobrecalentamientos y demás incomodidades a la hora de trabajar con él.

Si el número de computadoras que van a disponerse para el cluster no justifica estas inversiones, asegúrate de poder tener siempre un fácil acceso a los nodos, siempre podemos encontrarnos con el problema de tener que cambiar un ventilador o un disco averiado.

4.3. Requerimientos software

El sistema planteado requiere un sistema básico Linux instalado. Podemos elegir cualquier distribución del mercado que encontremos atractiva, este aspecto no es importante.

Lo que sí es importante es usar al menos la versión 2.4 del kernel y que tus tarjetas de red estén bien configuradas.

4.4. Planteamientos de tu cluster

Para configurar tu cluster openMosix en un *pool* de nodos, o conjunto de estaciones de trabajo, tendremos diferentes opciones, cada una con sus ventajas e inconvenientes.

En una **single-pool** todos los servidores y estaciones de trabajo son utilizadas como un

cluster único: cada máquina forma parte del cluster y puede migrar procesos hacia cada uno de los otros nodos existentes.

Esta configuración hace que tu propia máquina forme parte del *pool*.

En un entorno llamado **server-pool** los servidores son parte del cluster mientras que las estaciones de trabajo no lo son. Si quisiéramos ejecutar aplicaciones en el cluster necesitaríamos entrar en él de forma específica. De este modo las estaciones de trabajo permanecerán libres de procesos remotos que les pudieran llegar.

Existe una tercera alternativa llamada **adaptive-pool**, donde los servidores son compartidos mientras que las estaciones de trabajo podrán entrar y salir del cluster. Podemos imaginar que las estaciones deban ser usadas durante un cierto intervalo de tiempo diario, y que fuera de este horario puedan ser aprovechadas para las tareas del cluster.

5. Instalación de un cluster openMosix

5.1. Obteniendo fuentes y paquetes

5.1.1. Parcheando el kernel de linux

5.1.2. Opciones en el nuevo kernel parcheado

5.1.3. Las herramientas de usuario

5.1.4. Paquetes RPM

5.1.5. Debian

5.1.6. Fuentes

5.2. Compilaciones necesarias

5.2.1. Compilación del nuevo kernel

5.2.2. Configurando las herramientas de usuario

5.3. Instalaciones del programario

6. Administración del cluster

El mantenimiento de un sistema resulta incluso más delicado y costoso (en tiempo) que su correcta instalación. En este capítulo se tomará contacto con todas las herramientas con las que cuenta openMosix para poder gestionar tu sistema.

La recomendación que lanzamos desde este manual es que pruebes con ellas para conocer exactamente la respuesta de tu cluster, ya que más tarde puede facilitarte la detección de errores o configuraciones poco adecuadas.

6.1. Administración básica

openMosix proporciona como principal ventaja la migración de procesos hacia aplicaciones HPC. El administrador puede configurar el cluster utilizando las herramientas de área de usuario de openMosix (openMosix-user-space-tools⁸) o editando la interficie que encontraremos en */proc/hpc* y que será descrita con más detalle seguidamente.

6.2. Configuración

Los valores en los ficheros del directorio */proc/hpc/admin* presentan la configuración actual del cluster. El administrador del mismo puede configurar estos valores para cambiar la configuración en tiempo de ejecución, tal como se muestra en las tablas.

⁸<http://www.orcero.org/openmosix>

echo 1 > /proc/hpc/admin/block	bloquea la llegada de procesos remotos
echo 1 > /proc/hpc/admin/bring	lleva todos los procesos a su nodo raíz

Cuadro 1: Cambiando los parámetros en /proc/hpc

config	el fichero de configuración principal (escrito por la utilidad <i>setpe</i>)
block	permite/prohíbe la llegada de procesos remotos
bring	lleva todos los procesos a su nodo raíz
dfslinks	lista los actuales enlaces DFSA
expel	envía los procesos huésped a su nodo raíz
gateways	numero máximo de gateways
lstay	los procesos locales se suspenderán
mospe	contiene el ID de nuestro nodo openMosix
nomfs	activa/desactiva MFS
overheads	para ajustes
quiet	detiene la obtención de información sobre la carga del sistema
decay-interval	intervalo para recoger información sobre la carga
slow-decay	por defecto 975
fast-decay	por defecto 926
speed	velocidad relativa a un PIII/1GHz
stay	activa/desactiva el proceso de migrado automático

Cuadro 2: Binarios en /proc/hpc/admin

Existen utilidades adicionales a la interfície de */proc* y a las líneas de comandos. Por ejemplo existen unos parches para *ps* y para *top* (llamados *mps* y *mtop*) los cuales muestran adicionalmente el ID de nuestro nodo openMosix en una columna. Esta posibilidad es interesante para encontrar dónde ha migrado un cierto proceso.

Para clusters pequeños pueden ser muy útiles las utilidades de openMosixView, una GUI para las tareas de administración más comunes y que más adelante se detalla en un capítulo.

6.4. Detección automática de nodos

El demonio de auto-detección de nodos, *omdiscd*, proporciona un camino automático para la configuración de nuestro cluster openMosix. Con él podremos eliminar la necesidad de configuraciones manuales como son la edición del fichero */etc/mosix.map*.

omdiscd genera un envío de paquetes *multicast* (a todas las direcciones, en nuestro caso, nodos) para notificar a los otros nodos que hemos añadido uno nuevo. Esto significa que al añadir un nodo sólo tendremos que iniciar *omdiscd* en él.

Debemos ocuparnos de algunos requisitos previos como pueden ser una buena configuración de la red de interconexión de los nodos, principalmente para el correcto enrutamiento de paquetes. Sin una ruta por defecto deberemos especificar a *omdiscd* la interfície con la opción *-i*. De otra forma acabará con un error parecido al siguiente:

clear	resetea las estadísticas
cpujob	informa a openMosix que el proceso está ligado al procesador
iojob	informa a openMosix que el proceso está ligado a la E/S
slow	informa a openMosix que actualice las estadísticas más lentamente
fast	informa a openMosix que actualice las estadísticas más rápidamente

Cuadro 3: Escribiendo un 1 en /proc/hpc/decay

/proc/[PID]/cantmove	razón por la cual el proceso ha sido migrado
/proc/[PID]/goto	a qué nodo el proceso podrá migrar
/proc/[PID]/lock	si el proceso se ve bloqueado en su nodo raíz
/proc/[PID]/nmigs	el numero de veces que el proceso ha migrado
/proc/[PID]/where	donde el proceso se encuentra siendo computado actualmente
/proc/[PID]/migrate	same as goto remote processes
/proc/hpc/remote/from	el nodo raíz del proceso
/proc/hpc/remote/identity	información adicional del proceso
/proc/hpc/remote/statm	estadística de memoria del proceso
/proc/hpc/remote/stats	estadística del procesador del proceso

Cuadro 4: Información adicional sobre los procesos locales

/proc/hpc/nodes/[openMosix_ID]/CPUs	el número de CPUs que posee el nodo
/proc/hpc/nodes/[openMosix_ID]/load	la carga de openMosix en este nodo
/proc/hpc/nodes/[openMosix_ID]/mem	memoria disponible para openMosix
/proc/hpc/nodes/[openMosix_ID]/rmem	memoria disponible para Linux
/proc/hpc/nodes/[openMosix_ID]/speed	velocidad del nodo relativa a un PIII/1GHz
/proc/hpc/nodes/[openMosix_ID]/status	estado del nodo
/proc/hpc/nodes/[openMosix_ID]/tmem	memoria disponible
/proc/hpc/nodes/[openMosix_ID]/util	utilización del nodo

Cuadro 5: Información de los otros nodos

6.3. Las herramientas de área de usuario

Estas herramientas permitirán un fácil manejo del cluster openMosix. Seguidamente se enumeran con todos sus parámetros.

`migrate [PID] [openMosix_ID]` envía una petición de migrado del proceso identificado con el ID, al nodo que indiquemos..

`mon` es un monitor de los *daemons* basado en el terminal y da información relevante sobre el estado actual que puede ser visualizada en diagramas de barras.

`mosctl` es la principal utilidad para la configuración de openMosix. Su sintaxis es:

```
mosctl [stay|nostay]
       [block|nblock]
       [quiet|noquiet]
       [nomfs|mfs]
       [expel|bring]
       [gettune|getyard|getdecay]
```

```

mosct whois [openMosix_ID|IP-address|hostname]
mosct [getload|getspeed|status|isup|getmem|getfree|getutil] [openMosix_ID]
mosctl setyard [Processor-Type|openMosix_ID||this]
mosctlsetspeed interger-value
mosctlsetdecay interval [slow fast]

```

```

Aug 31 20:41:49 localhost omdiscd[1290]: Unable to determine address of
default interface. This may happen because there is no default route
configured. Without a default route, an interface must be: Network is
unreachable

```

```

Aug 31 20:41:49 localhost omdiscd[1290]: Unable to initialize network.
Exiting.

```

Un ejemplo de buena configuración podría ser la siguiente:

```

[root@localhost log]# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
10.0.0.0         0.0.0.0         255.0.0.0      U        0      0      0 eth0
127.0.0.0       0.0.0.0         255.0.0.0      U        0      0      0 lo
0.0.0.0         10.0.0.99      0.0.0.0        UG       0      0      0 eth0

```

La importancia de poder automatizar el reconocimiento de nuevos nodos conectados al sistema ha facilitado que se llegue a la simplicidad con la que contamos actualmente para iniciar dicha detección, con el comando `omdiscd`.

Ahora echando un vistazo a los *logfiles* tendríamos que ver algo parecido a

```

Sep  2 10:00:49 oscar0 kernel: openMosix configuration changed: This is openMosix
#2780 of 6 configured)
Sep  2 10:00:49 oscar0 kernel: openMosix #2780 is at IP address 192.168.10.220
Sep  2 10:00:49 oscar0 kernel: openMosix #2638 is at IP address 192.168.10.78
Sep  2 10:00:49 oscar0 kernel: openMosix #2646 is at IP address 192.168.10.86
Sep  2 10:00:49 oscar0 kernel: openMosix #2627 is at IP address 192.168.10.67
Sep  2 10:00:49 oscar0 kernel: openMosix #2634 is at IP address 192.168.10.74

```

Tendremos el cluster listo para ser utilizado.

`omdiscd` tiene otras opciones entre las que cuentan poder ejecutarse como un demonio (por defecto) o en *background* (segundo plano) donde la salida será la pantalla (la salida estándar), con el comando `omdiscd -n`. La interfície, como ya se ha indicado, debe ser especificada con la opción `-i`.

Ahora vamos a ver brevemente la herramienta `showmap`. Esta utilidad nos mostrará el nuevo mapa.

stay	desactiva la migración automática
nostay	migración automática (defecto)
lstay	local processes should stay
nolstay	los procesos locales podran migrar
block	bloquea la llegada de otros procesos
noblock	permite la llegada de procesos
quiet	desactiva la posibilidad de dar información sobre la carga del nodo
noquiet	activa la posibilidad de dar información sobre la carga del nodo
nomfs	desactiva MFS
mfs	activa MFS
expel	envía fuera del nodo los procesos que han llegado previamente
bring	traerá todos los procesos migrados hacia su nodo raíz
gettune	muestra el parámetro de overhead
getyard	muestra la utilización actual de Yardstick
getdecay	muestra el estado del parámetro decay
whois	nos muestra el openMosix-ID, la dirección IP y los nombres de host del cluster
getload	muestra la carga (openMosix-)
getspeed	muestra la velocidad (openMosix-)
status	muestra el estado y la configuración actual
isup	nos informa de si un nodo está funcionando o no (ping openMosix)
getmem	muestra la memoria lógica libre
getfree	muestra la memoria física libre
getutil	muestra la utilización del nodo
setyard	establece un nuevo valor para Yardstick
setspeed	establece un nuevo valor para la velocidad (openMosix-)
setdecay	establece un nuevo valor para el intervalo del decay

Cuadro 6: Parámetros de `mosctl` con más detalle

Con `mosrun` ejecutaremos un comando especialmente configurado en un nodo establecido.

Su sintaxis: `mosrun [-h|openMosix_ID| list_of_openMosix_IDs] command [arguments]`

El comando `mosrun` puede ser ejecutado con diversas opciones. Para evitar complicaciones innecesarias viene con ciertas pre-configuraciones para ejecutar las tareas con configuraciones especiales de openMosix.

nomig	runs a command which process(es) won't migrate
runhome	ejecuta un comando bloqueado en el nodo raíz
runon	ejecutará un comando el cuál será directamente migrado y bloqueado a cierto nodo
cpujob	informa a openMosix que el proceso está ligado a la CPU
iojob	informa a openMosix que el proceso está ligado a la E/S
nodecay	ejecuta un comando e informa al cluster de no refrescar las estadísticas de carga
slowdecay	ejecuta un comando con intervalo de decay grande para acumular en las estadísticas
fastdecay	ejecuta un comando con intervalo de decay pequeño para acumular en las estadísticas

Cuadro 7: Parámetros adicionales para `mosrun`

`setpe` es una utilidad de configuración manual del nodo *sintaxis*:

```
setpe -w -f [hpc_map]
setpe -r [-f [hpc_map]]
setpe -off
```

`-w` lee la configuración de openMosix desde un fichero (normalmente `/etc/hpc.map`).

`-r` escribe la configuración actual de openMosix en un fichero (normalmente `/etc/hpc.map`).

`-off` desactiva la configuración actual del cluster.

`tune` es una utilidad de calibración y optimización de openMosix (para más información recurra a las páginas *man* de `tune`).

```
[root@oscar0 root]# showmap
My Node-Id: 0x0adc
```

Base Node-Id	Address	Count
0x0adc	192.168.10.220	1
0x0a4e	192.168.10.78	1
0x0a56	192.168.10.86	1
0x0a43	192.168.10.67	1
0x0a4a	192.168.10.74	1

Existen otras muchas utilidades que pueden ser útiles para la detección automática de nodos, como un mecanismo de routing para clusters con más de una red de conexión. Toda esta información es actualizada constantemente y podremos encontrarla en los ficheros README y DESIGN de las herramientas de usuario.

6.4.1. Compilaciones

Si queremos obtener este módulo a partir de las fuentes tendremos que hacer una pequeña modificación en el fichero `openmosix.c`. Una de las líneas, concretamente

```
#define ALPHA
```

tendremos que comentarla ya que nosotros estamos trabajando en plataforma x86.

Si quisiéramos tener un historial más detallado podemos editar `main.c` para escribir

```
log_set_debug(DEBUG_TRACE_ALL);
```

(en la línea 84 aproximadamente)

ahora podremos ejecutar

```
make clean
```

```
make
```

6.4.2. Problemas

Algunas veces la auto-detección no funcionará tal como podríamos esperar, per ejemplo cuando un nodo debería ser detectado y no ve el tráfico multicast que se lleva a cabo en la red.

Esto ocurre con algunas targetas PCMCIA. Una solución posible sería poner la interfície en *modo promíscuo*, tal como se detalla seguidamente:

```
Aug 31 20:45:58 localhost kernel: openMosix configuration changed:
This is openMosix #98 (of 1 configured)
Aug 31 20:45:58 localhost kernel: openMosix #98 is at IP address 10.0.0.98
Aug 31 20:45:58 localhost omdiscd[1627]: Notified kernel to activate
openMosix Aug 31 20:45:58 localhost kernel:
Received an unauthorized information request from 10.0.0.99
```

Algo que podríamos probar es forzar manualmente nuestro NIC a modo promíscuo y/o multicast, así:

```
ifconfig ethx promisc
```

```
o
```

```
ifconfig ethx multicast
```

Podremos ejecutar igualmente

```
tcpdump -i eth0 ether multicast
```

Si se nos muestra **simulated** es que seguramente hemos olvidado poner el comentario a la línea `#define ALPHA`, sería:

```
Aug 31 22:14:43 inspon omdiscd[1422]: Simulated notification to activate openMosix
[root@inspon root]# showmap
My Node-Id: 0x0063

Base Node-Id Address          Count
-----
0x0063      10.0.0.99          1
[root@inspon root]# /etc/init.d/openmosix status
OpenMosix is currently disabled
[root@inspon root]#
```

7. Ajustes en el cluster

8. openMosixview

8.1. Introducción

openMosixview es la siguiente versión (y totalmente reescrita) de MosixView.

Es una interfície gráfica (GUI) libre para la administración y mantenimiento de un cluster openMosix que podemos bajarnos de la web del proyecto⁹.

La *suite* openMosixview contiene 6 aplicaciones altamente eficaces e útiles tanto para la administración como para el monitoraje de nuestro cluster.

- **openMosixview** la principal aplicación de monitoraje y administración
- **openMosixprocs** una aplicación para la administración de procesos
- **openMosixcollector** acumula la información del cluster proporcionada por los *daemons*
- **openMosixanalyzer** para el análisis de la información colectada por openMosixcollector
- **openMosixhistory** un historial de procesos del cluster
- **3dmosmon** un visor para monitoraje de datos en 3D

Todos los componentes son accesibles desde la ventana de la aplicación principal. Los comandos openMosix más comunes podrán ser ejecutados con unos pocos clicks de ratón.

Podremos encontrar también *sliders* para la asignación de prioridad para cada nodo, con el fin de simplificar el balanceo de carga (manual o automático). openMosixview ha sido adaptado al openMosix-auto-discovery y puede obtener toda la información desde /proc-interface.

8.2. Instalación

Requerimientos:

- tener instaladas las librerías QT \geq 2.3.0
- derechos de *root*!
- *rlogin* y *rsh* (o *ssh*) en todos los nodos del cluster y sin contraseñas
- las herramientas de usuario de openMosix (*mosctl*, *migrate*, *runon*, *iojob*, *cpujob*...)

Los paquetes RPM tienen como directorio de instalación la ruta */usr/local/openMosixview*.

⁹<http://www.openMosixview.com>

8.2.1. Instalación de los paquetes RPM

Tendremos que bajarnos la última versión de los paquetes RPM de openMosixview. Luego ejecutaremos el comando (suponiendo la versión 1.2)

```
rpm -i openMosixview-1.2.rpm
```

Esto nos instalará los ficheros binarios en el directorio */usr/bin*. Para desinstalarlo ejecutaremos

```
rpm -e openMosixview
```

8.2.2. Instalación de las fuentes

Bajaremos la última versión de openMosixview y descomprimiremos y desempaquetaremos el paquete (suponiendo la versión 1.2):

```
gunzip openMosixview-1.2.tar.gz
```

```
tar -xvf openMosixview-1.2.tar
```

8.2.3. Script de configuración automático

Sólo será necesario entrar al directorio openMosixview y ejecutar:

```
./setup [directorio_de_instalación_qt_2.3.x]
```

8.2.4. Compilación manual

Será necesario situar la variable QTDIR hacia el directorio de la distribución QT, por ejemplo:

```
export QTDIR=/usr/lib/qt-2.3.0 (para bash)
```

o

```
setenv QTDIR /usr/lib/qt-2.3.0 (para csh)
```

Tras lo anterior tendríamos que ejecutar con éxito la configuración

```
./configure
```

```
make
```

Luego tendremos que hacer lo mismo en los subdirectorios de openMosixcollector, openMosixanalyzer, openMosixhistory and openMosixviewprocs.

Copiaremos todos los binarios a /usr/bin

```
cp openMosixview/openMosixview /usr/bin
cp openMosixviewproc/openMosixviewprocs/mosixviewprocs /usr/bin
cp openMosixcollector/openMosixcollector/openMosixcollector /usr/bin
cp openMosixanalyzer/openMosixanalyzer/openMosixanalyzer /usr/bin
cp openMosixhistory/openMosixhistory/openMosixhistory /usr/bin
```

Y el script de iniciación de openMosixcollector en tu directorio de iniciación, por ejemplo:

```
cp openMosixcollector/openMosixcollector.init /etc/init.d/openMosixcollector
o
cp openMosixcollector/openMosixcollector.init /etc/rc.d/init.d/openMosixcollector
```

Ahora tendremos que copiar los binarios de openMosixprocs de cada nodo del cluster al directorio

/usr/bin/openMosixprocs

```
rcp openMosixprocs/openMosixprocs tu_nodo:/usr/bin/openMosixprocs
```

Y ahora ya podremos ejecutar openMosixview con el comando openMosixview .

8.3. Utilizando openMosixview

8.3.1. Aplicación principal

La Figura 1 nos muestra la ventana de la aplicación.

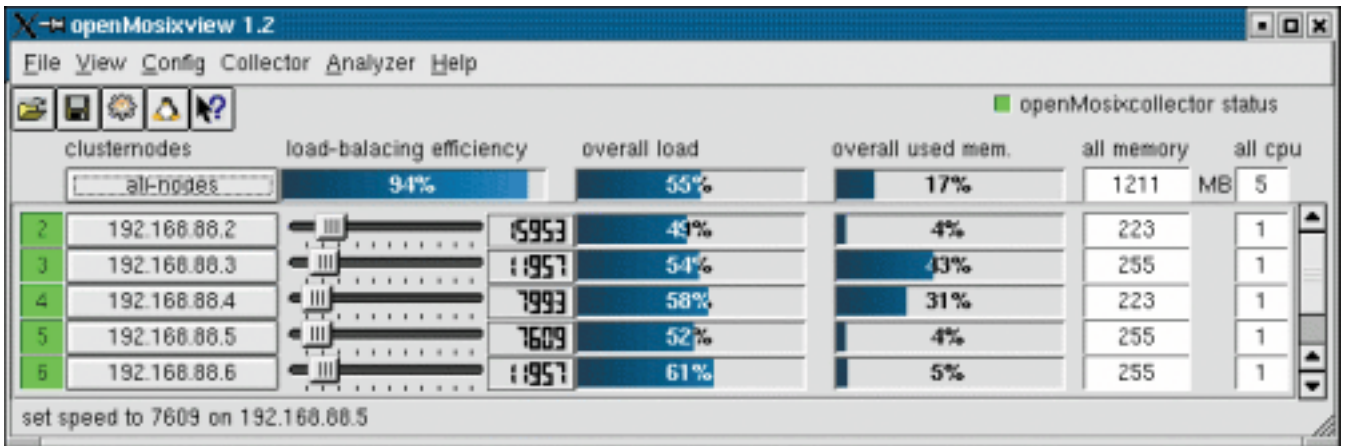


Figura 1: openMosixview: Aplicación principal

openMosixview nos muestra, para cada nodo del cluster (cada fila): una luz, un botón, un slider, un número lcd, dos barras de progreso y un par de etiquetas.

Las luces de la izquierda nos muestran el ID del nodo y su estado. En color rojo significa que el nodo no se encuentra operativo, y verde lo contrario.

Si hacemos clic en el botón que muestra la dirección IP de un nodo habremos invocado al diálogo de configuración que nos mostrará los botones para ejecutar los comandos de `mosctl` más comunes (explicados en próximos subcapítulos).

Con los *sliders* de velocidad podemos establecer la velocidad que considerará el cluster para cada nodo. Este parámetro se muestra en el *display* lcd.

Podemos intervenir también en el balanceo de carga de cada nodo cambiando sus valores. Los procesos en un cluster openMosix migran más fácilmente hacia un nodo cuya velocidad sea más elevada. Recordemos que este concepto de velocidad no tiene que ser el que realmente posea la computadora, es simplemente el parámetro que queremos que openMosix considere para cada máquina.

Las barras de progreso, que conforman el par de columnas en la mitad de la ventana, dan una idea general de la carga de cada nodo del cluster. La primera se refiere a la carga del procesador y muestra un porcentaje que será una aproximación del valor escrito por openMosix en el fichero `/proc/hpc/nodes/x/load`. La segunda barra nos muestra la memoria utilizada en cada nodo. El box de la izquierda nos muestra el total de memoria disponible.

Finalmente el box de más a la derecha nos muestra el número de procesadores de cada nodo.

En la esquina superior-izquierda podemos ver el box *load-balancing efficiency*, éste es un

indicador de la eficiencia del algoritmo de balanceo de carga. Un valor próximo al 100% indica que la carga computacional ha podido dividirse equitativamente entre los nodos.

Podemos utilizar los menús de `collector-` y `analyzer-` para administrar `openMosixcollector` y `openMosixanalyzer`, respectivamente. Estas dos partes de las aplicaciones `openMosixview` son muy adecuadas para construir un historial del trabajo hecho y la manera en como se ha hecho en el cluster.

8.3.2. La ventana de configuración

Esta ventana emergente de la Figura 2 aparecerá si clicamos en el botón de la ip de cualquier nodo.

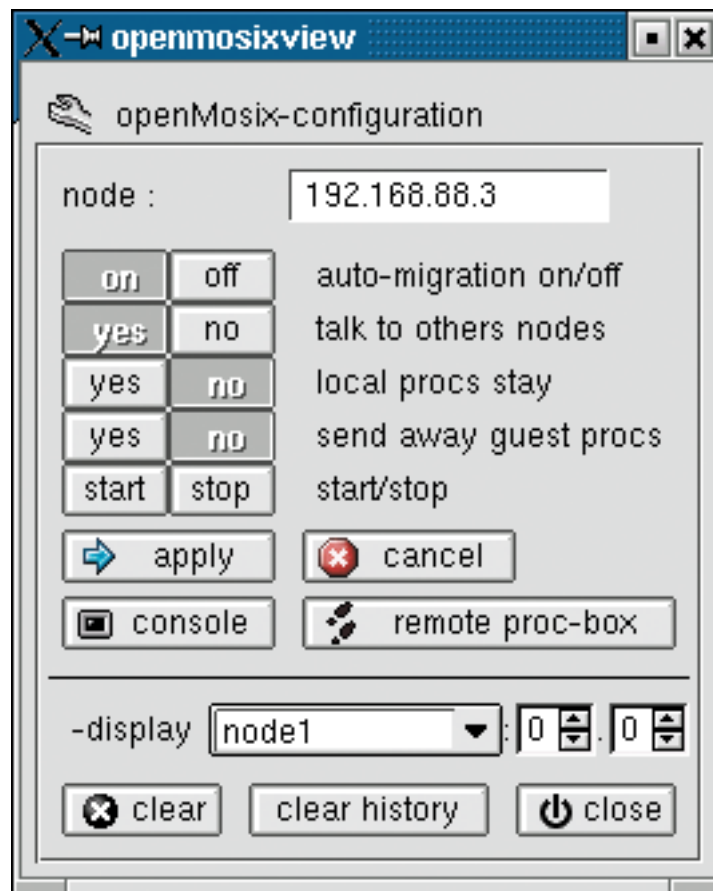


Figura 2: openMosixview: Propiedades de los nodos

La configuración de `openMosix` de cada nodo puede ser cambiada fácilmente. Todos los comandos podrán ser ejecutados con `rsh` o `ssh` en los nodos remotos (y también en el nodo local, evidentemente) como `root` sin necesidad de contraseñas.

Si tenemos instalado `openMosixprocs` en los nodos remotos del cluster podremos clicar en el botón `remote proc-box` para invocar `openMosixprocs` desde el remoto. Se nos mostrará en pantalla los parámetros `[xhost+hostname]` y serán configurados para apuntar a nuestro nodo local. Además el cliente es ejecutado en el remoto con `rsh` o `ssh` (recordemos que tendríamos que tener copiados los binarios de `openmosixprocs` en el directorio `/usr/bin` de cada nodo).

openMosixprocs nos permitirá una administración de nuestros programas.

Si hemos entrado a nuestro cluster desde una estación de trabajo remota podremos introducir nuestro nombre de nodo local en la edit-box, debajo de la *remote proc-box*. Luego openMosixprocs se nos mostrará en nuestra estación de trabajo y no en el nodo del cluster donde hemos entrado (podemos configurar [xhost+clusternode] en nuestra estación de trabajo). Podemos encontrar un historial en el combo-box así como el nombre de nodo que hayamos escrito.

8.3.3. Ejecución avanzada

Si queremos iniciar trabajos en nuestro cluster el diálogo de *advanced execution* mostrado en la Figura 3 podrá ayudarnos para convertirlo a modo gráfico.

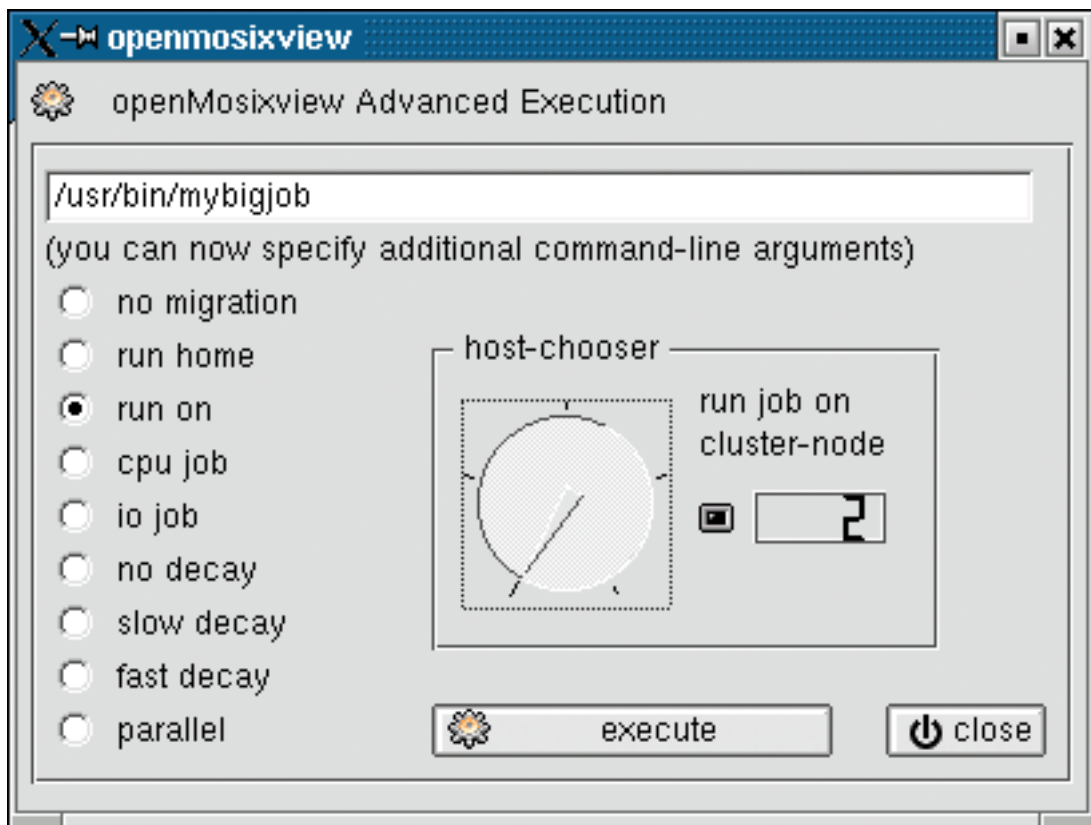


Figura 3: openMosixview: Ejecución avanzada

Elegiremos el programa a iniciar con el botón *run-prog* (en File-Open-Icon) y especificaremos cuál y donde se encuentra el trabajo que queremos iniciar en este diálogo de ejecución. Hay diferentes opciones que se describen en la próxima tabla.

8.3.4. La línea de comandos

Podremos especificar los argumentos a los que antes podíamos acceder gráficamente a través de comandos en el *linedit-widget* en la parte superior de la ventana, tal como se

no migration	iniciar un proceso local que no migrará
run home	iniciar un proceso local
run on	iniciar un trabajo en el nodo que elijamos con "nodo-chooser"
cpu job	iniciar un proceso cpu-intensivo en el nodo (nodo-chooser)
io job	iniciar un proceso IO-intensivo en el nodo (nodo-chooser)
no decay	iniciar un proceso sin decay (nodo-chooser)
slow decay	iniciar un proceso con decay lento (nodo-chooser)
fast decay	iniciar un proceso con decay rápido (nodo-chooser)
parallel	iniciar un proceso paralelo en todos o algunos nodos (special nodo-chooser)

Cuadro 8: Condiciones de inicio avanzadas para procesos

muestra en la Figura 3.

8.3.5. El host-chooser

Para todas las tareas que ejecutemos de manera remota sólo tenemos que elegir un nodo que la hospede con el dial-widget. El ID de openMosix del nodo se nos muestra también en forma de lcd. Luego pulsaremos *execute* para iniciar el trabajo.

8.3.6. El host-chooser paralelo

Podemos configurar el primer y el último nodo con 2 spinboxes. Luego el comando será ejecutado en todos los nodos, desde el primero hasta el último. Podremos también invertir esta opción.

8.4. openMosixprocs

8.4.1. Introducción

La ventana de procesos mostrada en la Figura 4 es muy útil para la administración de los procesos de nuestro cluster.

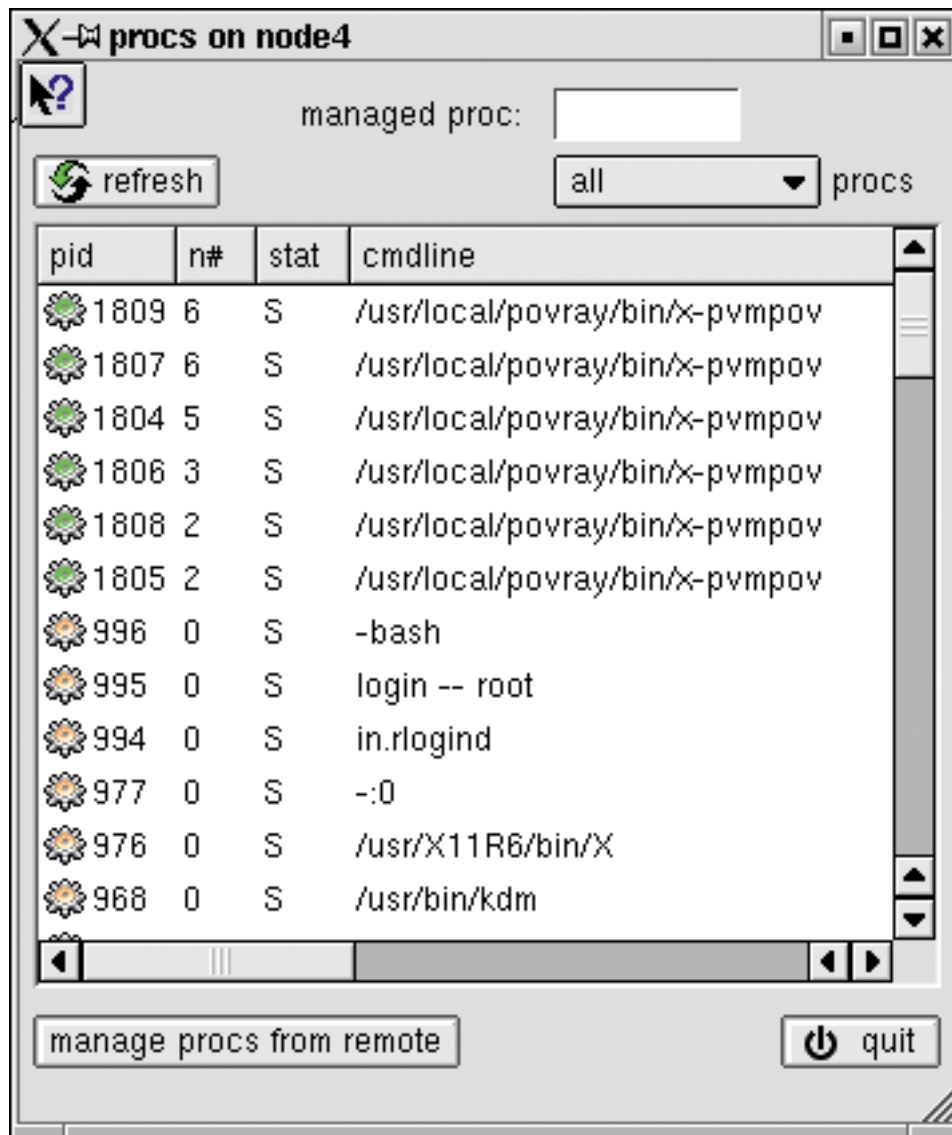


Figura 4: openMosixprocs: Administración de procesos

Deberemos instalar esta aplicación en cada nodo. La lista de procesos da una idea de lo que se está ejecutando y donde. La segunda columna informa del ID del nodo donde se está ejecutando la tarea. Con un doble clic sobre cualquier proceso invocaremos la ventana para administrar su migración (siguiente figura).

Un '0' significa 'local', los demás valores significan 'remoto'. Los procesos migrados están marcados con un icono verde y a los procesos que no pueden migrar se les añade un cerradero.

Hay también varias opciones para migrar el proceso remoto: enviarle las señales SIGSTOP

y SIGCONT o simplemente cambiarle la prioridad, con el comando `nice` por ejemplo.

Si hacemos clic en el botón *manage procs from remote* invocaremos a una ventana emergente (llamada *remote-procs windows*) que nos mostrará el proceso actualmente migrado hacia ese nodo.

8.4.2. La ventana de migración

El diálogo de la Figura 5 emergerá si clicamos sobre cualquier proceso en la ventana anterior.

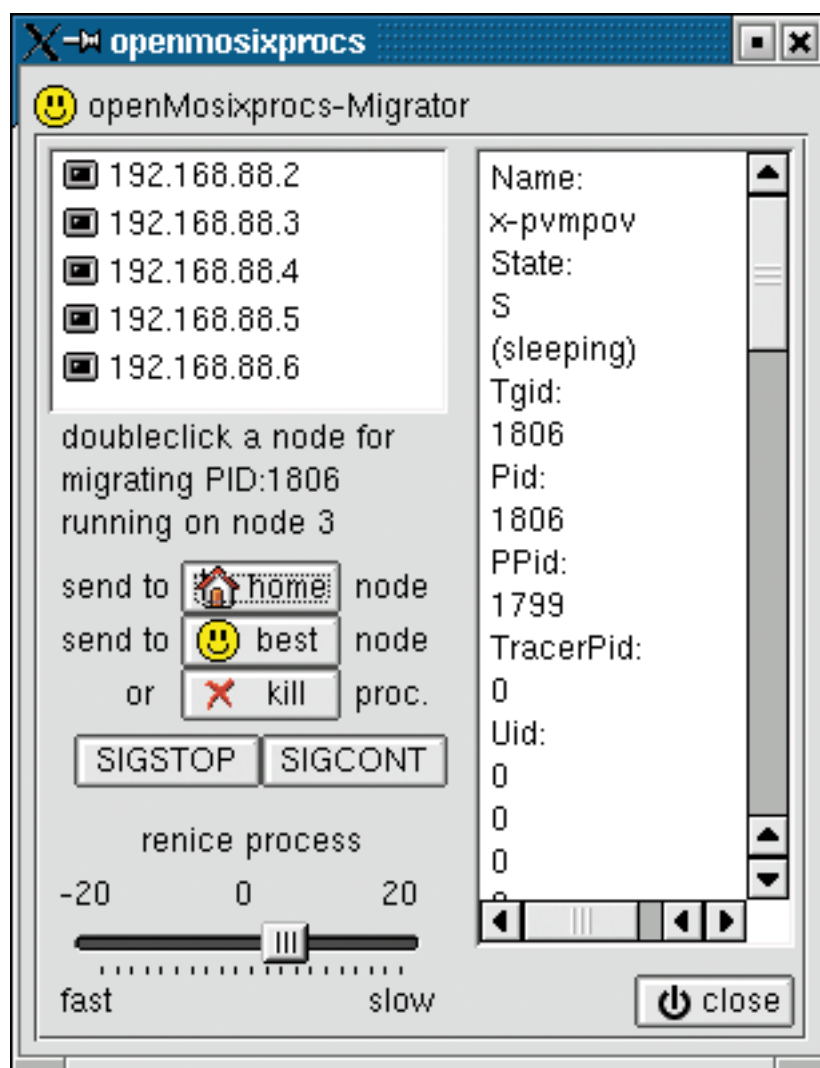


Figura 5: openMosixprocs: La ventana de migrado de un proceso(1)

La ventana openMosixview-migrator muestra todos los nodos de nuestro cluster, arriba a la izquierda. Esta ventana sirve para administrar un proceso (con información adicional).

Si hacemos doble clic en un nodo migrará a este nodo.

Tras breves instantes el icono del proceso cambiará a verde, lo que significa que está siendo

ejecutado remotamente.

El botón *home* envía un proceso a su nodo raíz. Con el botón *best* el proceso será enviado hacia el mejor nodo disponible en el cluster.

Esta migración se ve influenciada por la carga, velocidad, número de procesadores y su mayor o menor velocidad. Con el botón *kill* mataremos al proceso.

Para pausar un programa sólo tendremos que pulsar en el botón etiquetado como SIGSTOP y para reanudarlo, en SIGCONT.

Con el slider de *renice* podremos cambiar la prioridad de los procesos para una administración más completa, así el valor -20 se traduce en una mayor prioridad para terminar el trabajo, 0 le da prioridad normal y 20 le sitúa en una prioridad muy baja para tratarlo.

8.4.3. Administrando procesos remotamente

El diálogo mostrado en la Figura 6 aparecerá si clicamos en el botón *manage procs from remote*.

Las pestañas muestran los procesos que han migrado al nodo local. De forma parecida a los 2 botones en la ventana de migrado, el proceso será enviado a su nodo raíz si pulsamos *goto home node* y enviado al mejor nodo si pulsamos *goto best node*.

8.5. openMosixcollector

openMosixcollector es un *daemon* (demonio) que debe/puede ser invocado en cualquier miembro del cluster.

Genera un historial de la carga de cada nodo del cluster al directorio */tmp/openMosixcollector/**. Este historial puede servir de entrada para openMosixanalyser (explicado posteriormente) para ofrecer una vista general de la carga, memoria y procesos en nuestro nodo durante un intervalo de tiempo.

Existe otro historial principal llamado */tmp/openMosixcollector/cluster*

Y adicionalmente a éstos existen ficheros en los directorios donde se escriben los datos.

Al iniciar, openMosixcollector escribe su PID (ID del proceso) en */var/run/openMosixcollector.pid*

El demonio de openMosixcollector reinicia cada 12 horas y guarda el actual historial a */tmp/openMosixcollector[date]/**. Estos *backups* se hacen automáticamente pero siempre podremos lanzarlos manualmente.

Hay una opción que permite escribir un *checkpoint* al historial. Estos *checkpoints* podemos verlos gráficamente como una fina línea azul vertical si analizamos el historial con openMosixanalyser. Por ejemplo podemos poner un *checkpoint* cuando iniciamos cierto trabajo en el cluster y otro cuando éste finaliza.

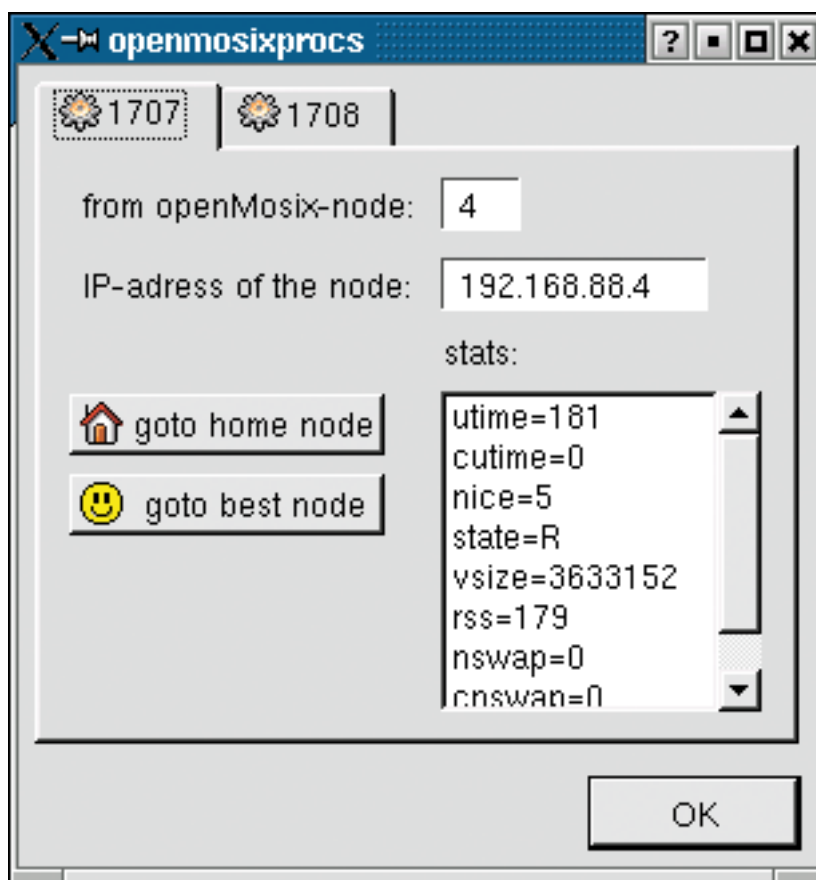


Figura 6: openMosixprocs: La ventana de migrado de un proceso(2)

Aquí tenemos una referencia de los posibles argumentos para la consola:

```
openMosixcollector -d //inicia el collector como un daemon
openMosixcollector -k //detiene el collector
openMosixcollector -n //escribe un checkpoint en el historial
openMosixcollector -r //guarda el actual historial y empieza uno de nuevo
openMosixcollector //saca por la salida estándar una ayuda rápida
```

Podemos iniciar este demonio con su script de iniciación, que se encuentra en */etc/init.d* o en */etc/rc.d/init.d*. Hemos de tener un enlace simbólico a uno de los *runlevels* para un inicio automático.

La forma para analizar los historiales creados la describimos en la siguiente sección, el openMosixanalyzer.

8.6. openMosixanalyzer

8.6.1. Una visión general de la carga del sistema

La siguiente figura nos muestra de forma gráfica la carga en el openMosixanalyzer.

Con el openMosixanalyzer tendremos un historial continuo de nuestro cluster. Los historiales generados por openMosixcollector se mostrarán ahora de forma gráfica, además de continua, lo que nos permitirá ver la evolución del rendimiento y demás parámetros de nuestro cluster a través del tiempo. openMosixanalyzer puede analizar los historiales a tiempo real (datos generados a tiempo real) y evidentemente también puede abrir antiguos *backups* con el menu File.

Los historiales serán guardados en */tmp/openMosixcollector/** (y los backups los tendremos en */tmp/openMosixcollector[date]/**) y sólo tendremos que abrir el historial principal del cluster para visualizar antiguos historiales de informaciones de carga. (el campo [date] en los ficheros de backup se refiere a la fecha en que han sido guardados)

La hora de inicio del backup podemos verla en la parte superior.

Si utilizamos openMosixanalyzer para tratar historiales a tiempo real tendremos que activar el check *refresh* para que se actualice automáticamente.

Las líneas que indican la carga son normalmente de color negro. Si la carga se incrementa a >75 las líneas se volverán rojas.

Estos valores se obtienen desde los ficheros */proc/hpc/nodes/[openMosix ID]/**

El botón *Find-out* de cada nodo calcula diversos valores para las estadísticas. Si lo clicamos abriremos una nueva ventana en la cual veremos las medias de la carga y memoria y algunas informaciones adicionales (estáticas y dinámicas) sobre el nodo en concreto.

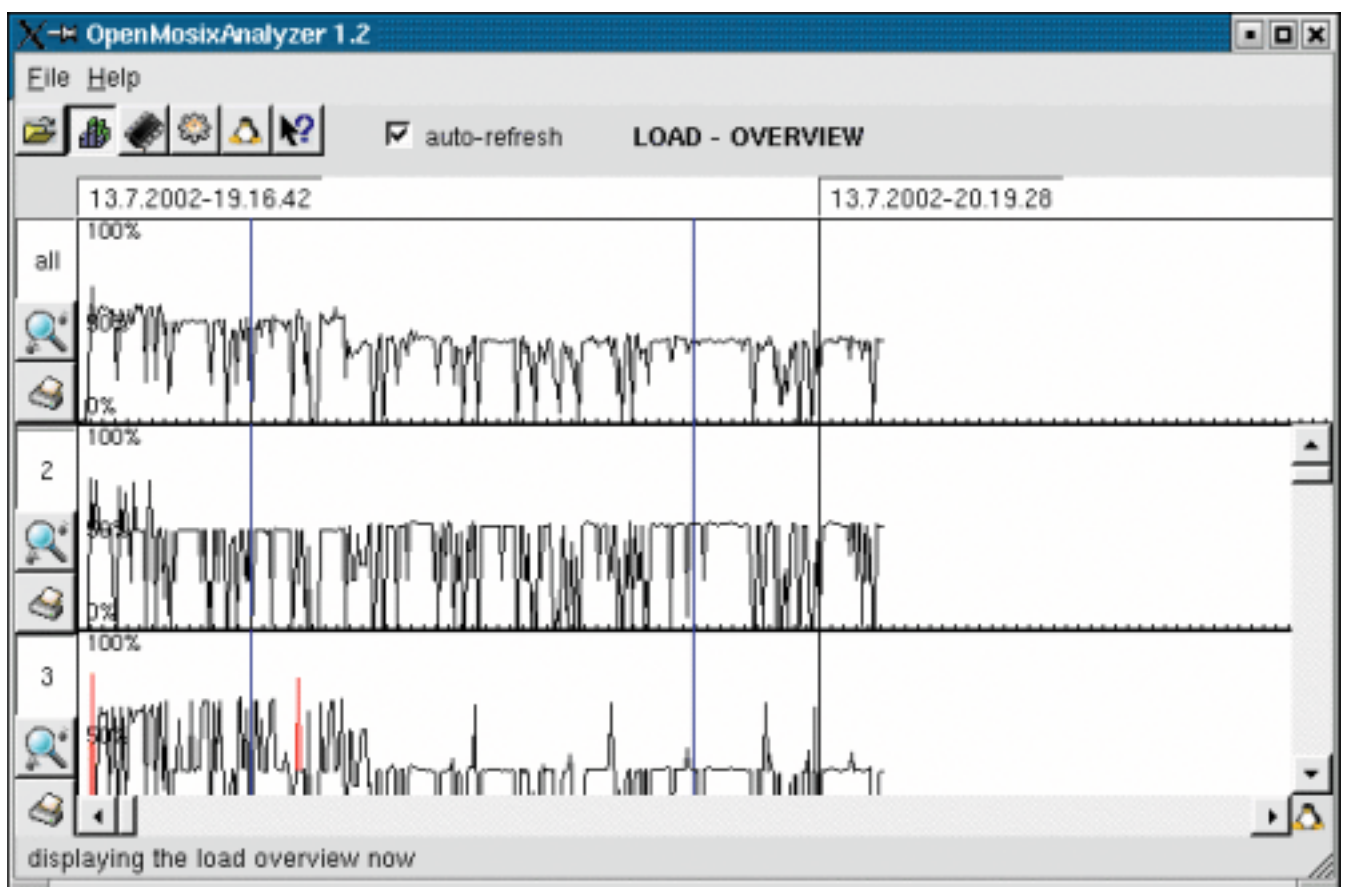


Figura 7: openMosixanalyzer. Historial de actividad de procesamiento del cluster

8.6.2. Estadísticas sobre los nodos del cluster

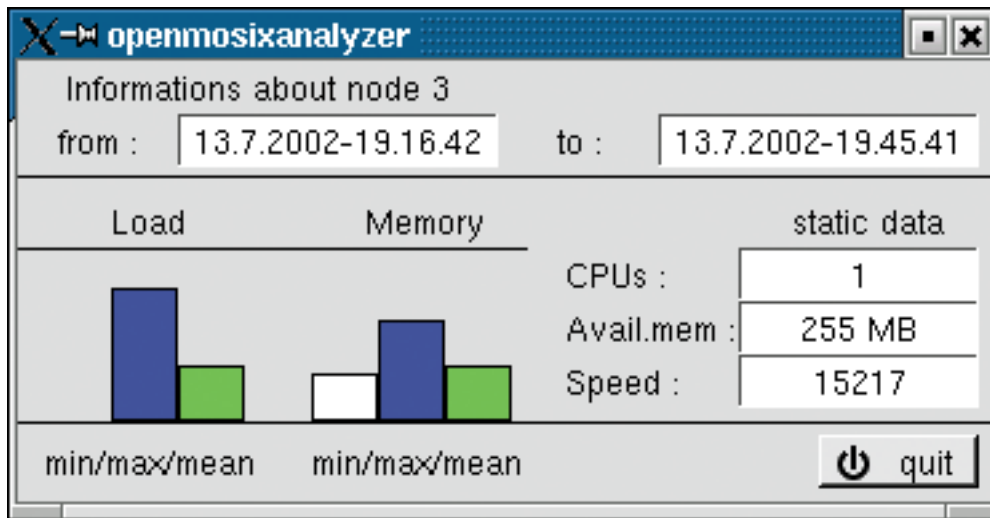


Figura 8: openMosixanalyzer. Estadísticas de los nodos

Si hay *checkpoints* escritos en el historial (generado por openMosixcollector) podremos verlos traducidos como líneas azules verticales. Esto permite comparar valores de carga a posteriori de forma fácil. Véase la Figura 8.

8.6.3. Monitoraje de la memoria

La Figura 9 muestra las gráficas de memoria proporcionadas por openMosixanalyzer

La filosofía de monitoraje de la memoria es idéntica a la explicada anteriormente con la carga y evidentemente sirven también para poder ver como se ha comportado openMosix para administrar nuestros recursos de memoria, en este caso.

Igualmente podemos tener un monitoraje a tiempo real o abrir antiguos historiales.

Para mostrar sus gráficas, openMosixanalyzer obtiene información de estos ficheros

```
/proc/hpc/nodes/[openMosix-ID]/mem.  
/proc/hpc/nodes/[openMosix-ID]/rmem.  
/proc/hpc/nodes/[openMosix-ID]/tmem.
```

Los *checkpoints* se muestran de igual forma que en la carga, con fina líneas verticales de color azul.

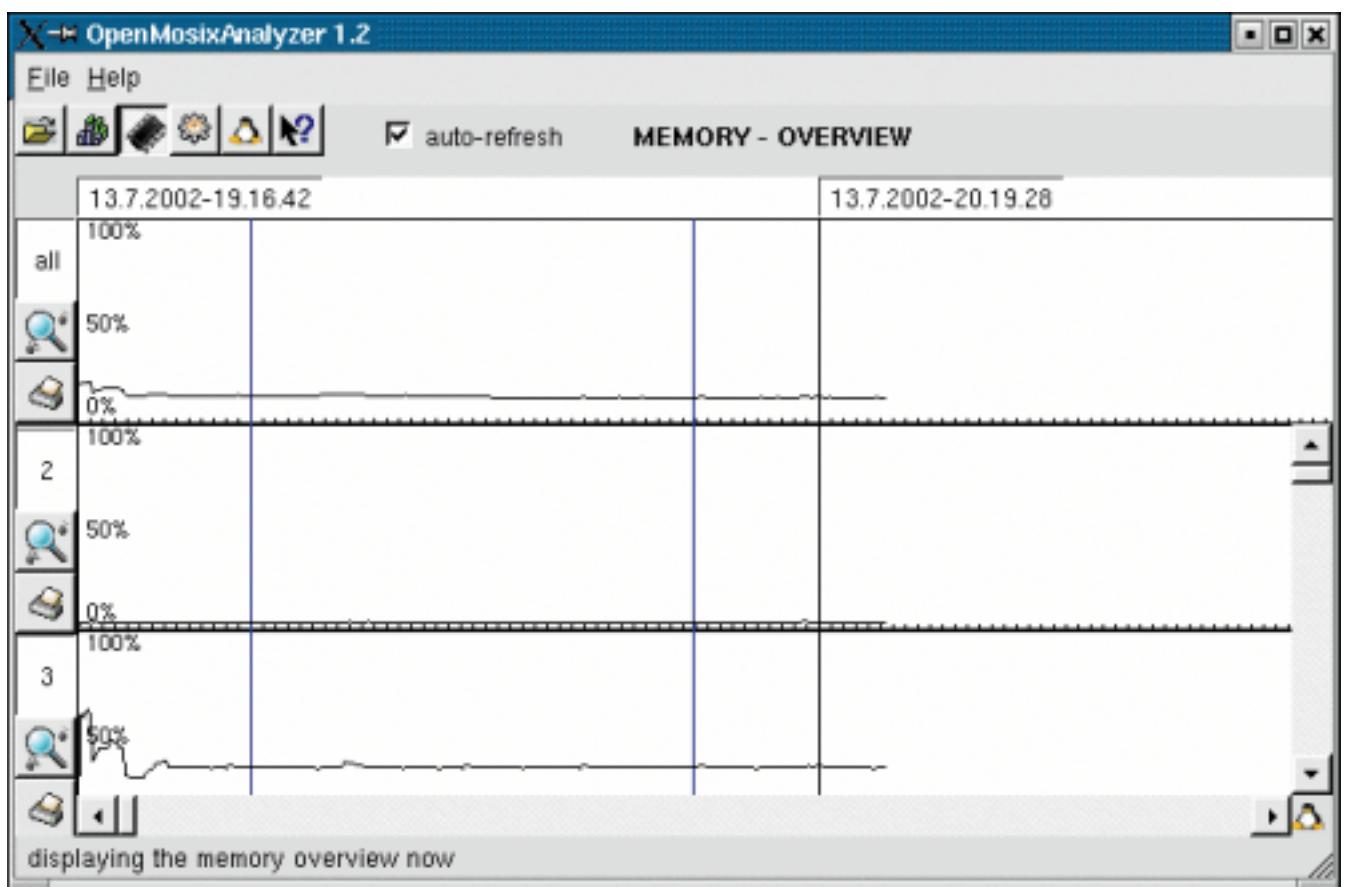


Figura 9: openMosixanalyzer. Historial de actividad de memoria de nuestro cluster

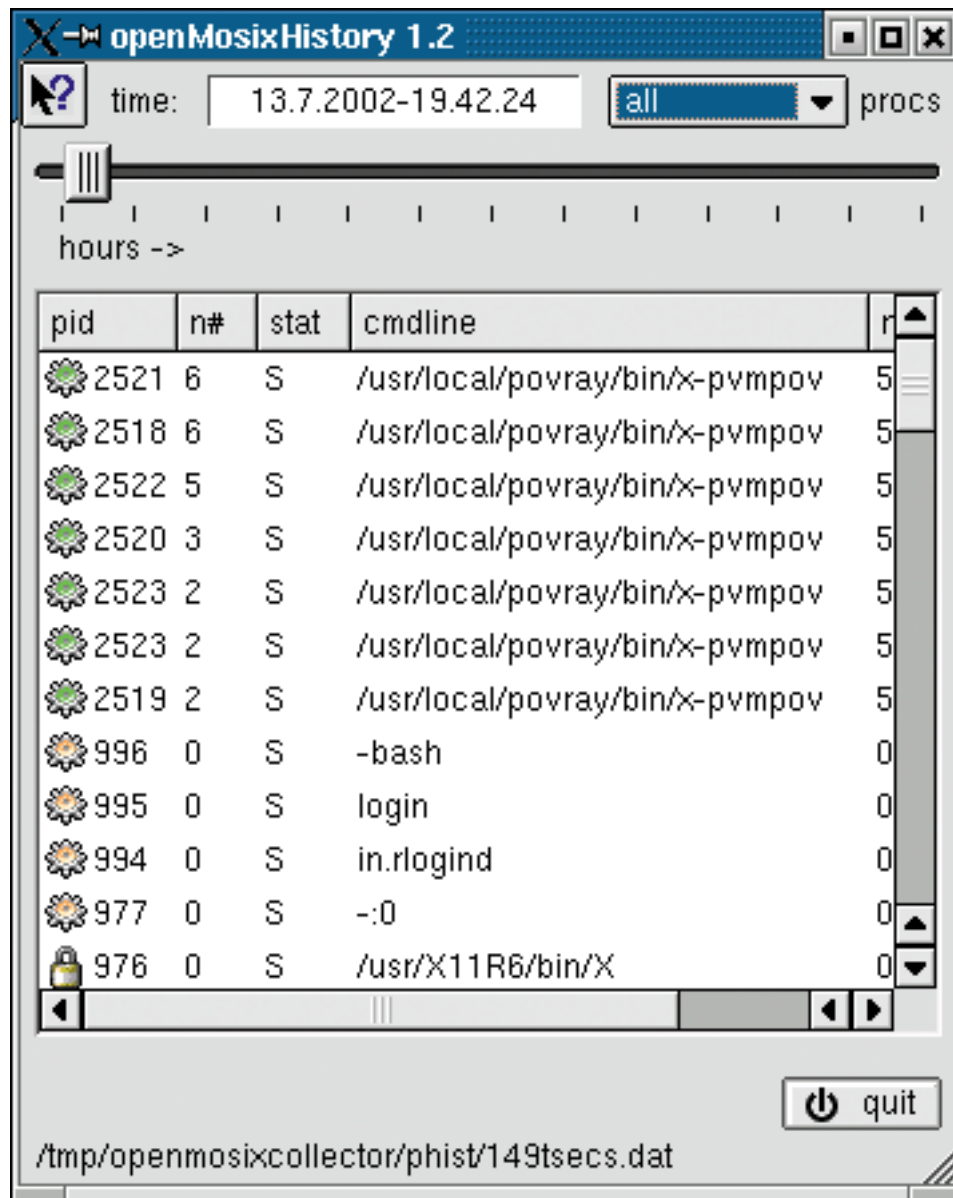


Figura 10: openMosixhistory. Un historial de los procesos ejecutados

8.7. openMosixhistory

Con openMosixhistory podremos acceder a la lista de procesos ejecutados en el pasado, véase Figura 10. Conseguiremos una lista de los procesos que se ejecutaron en cada nodo. openMosixcollector guarda la lista de procesos de cada nodo cuando lo iniciamos y con el openMosixhistory podremos navegar en dicha información para ver el comportamiento que desarrolló nuestro cluster.

Podremos cambiar fácilmente el tiempo de navegación con un *slider* situado en la parte superior, para así ajustar la ventana del pasado.

openMosixhistory puede analizar en tiempo real los historiales, y también abrir antiguos de forma idéntica a como lo hemos podido hacer en los otros analizadores de datos explicados anteriormente.

Los historiales se obtienen nuevamente de `/tmp/openMosixcollector/*` y sólo tendremos que abrir el historial principal para poder navegar en los ficheros de información sobre la carga dada tiempo atrás.

El tiempo de inicio del monitoraje podemos verlo en la parte superior y tenemos 12 horas de vista.

8.8. openMosixview + SSH2

Texto original Matthias Rechenburg. Todo error tipográfico, gramatical, semántico u ortográfico envíenlo al traductor: Marcelo kuntx@kx99.hn.org.

Usted puede leer la razón por la que debe usar SSH en vez de RSH cada día en el diario, cuando otro script-kiddy haya hackeado un sistema/red inseguro. Entonces se dará cuenta que SSH es una buena decisión después de todo.

Libertad x seguridad = constante (sacado de un grupo de noticias de seguridad.)

Es por eso que es un poco difícil de instalar SSH. SSH es seguro incluso si usted lo usa para logarse sin usar contraseña. Seguidamente daremos una forma de como configurarlo.

En principio es requerido que este corriendo el demonio del secure-shell. Si no está instalado instálelo.

```
rpm -i [sshd_rpm_package_from_your_linux_distribution_cd]
```

Si no está ya corriendo inícielo con:

```
/etc/init.d/ssh start
```

Ahora usted debe generar en su computadora un par de llaves para el ssh hagalo con:

```
ssh-keygen
```

Se le pedirá una frase de contraseña para ese par de llaves creado. La frase de contraseña es normalmente más larga que la contraseña incluso llegando quizás a ser una oración completa. El par de llaves es encriptado con esa frase de contraseña y guardado en:

```
t/.ssh/identity//su llave privada
/root/.ssh/identity.pub//su llave pública
```

¡¡No le entregue su llave privada a nadie!!

Ahora copie el contenido completo de `/root/.ssh/identity.pub` (su llave pública que debería tener una longitud de una línea) en `/root/.ssh/authorized_keys` al host remoto (también copie el contenido de `/root/.ssh/identity.pub` a su (local) `/root/.ssh/authorized_keys` como usted hizo con el nodo remoto, debido a que openMosixview necesita logar al nodo local sin que se le pida una contraseña.

Si ahora usted se trata de conectar mediante ssh al host remoto se le preguntará por la frase de contraseña de su llave pública respondiendo correctamente debería poder loguearse. ¿Cuál es la ventaja entonces?

La frase de contraseña es normalmente más larga que la contraseña.

La ventaja usted la puede obtener usando el ssh-agent. Éste maneja la frase de contraseña durante el ssh login: `ssh-agent`

El ssh-agent es iniciado y entrega 2 variables de entorno Usted debe configurarlas (si no lo estan ya), escriba:

```
echo $SSH_AUTH_SOCK
```

y

```
echo $SSH_AGENT_PID
```

para ver si han sido exportadas a su shell. Si esto no es así corte y péguelas desde su terminal de la siguiente forma:

- Para bash-shell:

```
SSH\_AUTH\_SOCK=/tmp/ssh-XXYqbMRe/agent.1065
export SSH\_AUTH\_SOCK
SSH\_AGENT\_PID=1066
export SSH\_AGENT\_PID
```

- Para csh-shell:

```
\texttt{setenv SSH\_AUTH\_SOCK /tmp/ssh-XXYqbMRe/agent.1065
setenv SSH\_AGENT\_PID 1066}
```

Con estas variables el demonio sshd remoto puede conectarse a su ssh-agent local usando el archivo socket en */tmp* (en este ejemplo */tmp/ssh-XXYqbMRe/agent.1065*). El ssh-agent ahora puede darle la frase de contraseña al host remoto usando este socket (obviamente es una transferencia encriptada).

Usted apenas tiene que agregar su llave pública al ssh-agent con el comando ssh-add:

```
ssh-add
```

Ahora usted debe poder logarse usando ssh al host remoto sin que se le pida contraseña alguna. Usted puede (debe) agregar los comandos ssh-agent y ssh-add a su login profile:

```
eval 'ssh-agent'
```

```
ssh-add
```

Con ello será iniciado cada vez que se loguee en su estación de trabajo local. Felicidades, le deseo que tenga logins seguros de ahora en más con openMosixview.

Finalmente y ya para terminar con este COMO, hay una entrada en el menú de openMosixView que permite seleccionar con que trabajar rsh/ssh, actívela y usted prodrá usar openMosixview incluso en redes inseguras. También debe grabar esta configuración, (la posibilidad de grabar la configuración actual en openMosixview fue agregada a partir de la versión 0.7), porque toma datos de inicio del esclavo usando rsh ó ssh (como usted lo haya configurado). Si usted elige un servicio que no está instalado correctamente openMosixview no funcionará.

Si usted no se puede conectar mediante rsh a un esclavo sin que se le pida una contraseña usted no podrá usar openMosixview con RSH.

Si usted no se puede conectar mediante ssh a un esclavo sin que se le pida una contraseña usted no podrá usar openMosixview con SSH.

Como creado por M. Rechenburg. Me olvidé de algo? Seguro. Envíeme un mail seguramente será agregado.

8.9. FAQ de openMosixview *-preguntas más frecuentes*

¡No puedo compilar openMosixview en mi equipo!

Antes que nada, como ya se ha comentado, es fundamental disponer de las librerías QT>=2.3.x.

La variable de entorno *QTDIR* tiene que estar configurada en *QT-installation* como se describe en el fichero de instalación *INSTALL*.

En versiones anteriores a la 0.6 podemos ejecutar `make clean` y borrar los dos ficheros:

```
/openmosixview/Makefile
/openmosixview/config.cache
```

y probar ahora de compilar otra vez ya que el autor, Matt Rechenburg, ha dejado los binarios y los objetos en estas versiones más antiguas.

En caso de tener otro tipo de problemas podemos postear en la lista de correo o directamente informar al autor.

¿Puedo utilizar openMosixview con SSH?

Sí desde la versión 0.7 se ha incorporado soporte para SSH. Tendr'íamos que ser capaces de acceder con `ssh` a cada nodo del cluster sin contraseña (lo mismo que es requerido para un acceso con `rsh`).

He intentado iniciar openMosixview pero se me bloquea antes de iniciarse. ¿Qué he hecho mal?

No deberemos ejecutar `openMosixview` en *background*, esto es llamarlo desde la consola con el comando `openMosixview&`.

También puede darse el caso que no podamos acceder a él a través de `rsh/ssh` (dependiendo de lo que usemos) como `root` sin contraseñas. Entonces deberíamos probar el comando `rsh hostname` como `root`. No se nos debería pedir ninguna contraseña pero se nos mostrará el shell para entrar nuestro login (nombre de usuario). Si estamos utilizando SSH el comando sería `ssh hostname`.

Tendremos que poder acceder como `root` en el sistema porque es el único modo de poder ejecutar los comandos administrativos que `openMosixview` requiere.

Si sólo tuviéramos instalado SSH en nuestro cluster, crearemos el fichero `/root/.openMosixview` y escribiremos `1111` en él.

Éste es el fichero de configuración principal y el último `1` se refiere a utilizar `ssh` en lugar de `rsh`.

El cliente `openMosixviewprocs/mosixview_client` no funciona.

El cliente `openMosixview_client` se ejecuta por `rsh` (o `ssh`, lo que tengamos configurado) en el host remoto. Debe estar instalado en `/usr/bin/` en cada nodo.

Si usamos RSH podemos probar

```
xhost +hostname
```

```
rsh hostname /usr/bin/openMosixview_client -display nombre_de_host_local:0.0
```

y si usamos SSH

```
xhost +hostname
```

```
ssh hostname /usr/bin/openMosixview_client -display nombre_de_host_local:0.0
```

Si esto funciona funcionará también en openMosixview.

openMosixview genera un *segmentation fault*

Puede que estemos utilizando versiones antiguas de openMosixview.

¿Por qué los botones del diálogo openMosixview-configuration no están preseleccionados? (automigración on/off, bloqueo on/off...)

Sería preferible que estuviesen preseleccionados, el problema se encuentra al obtener información del nodo.

Tenemos que logar en cada nodo ya que esta información no se encuentra dentro del cluster en sí .

El estado de cada nodo se guarda en el directorio */proc/hpc/admin/* de cada nodo.

9. Casos especiales

9.1. Laptops y targetas PCMCIA

No hay ninguna razón para que openMosix no funcione en laptops (ordenadores portátiles). El portátil puede pasar a ser miembro de cualquier cluster con el que estemos trabajando en un determinado momento.

La única condición es tener kernels compatibles y los mismos UID, identificadores de usuario.

Así pues será necesario que la dirección IP del laptop esté en los `/etc/mosix.map`. De hecho este tipo de entornos presentan varias ventajas para los portátiles. Para arrancar el laptop en openMosix es imprescindible una ip fija.

Podemos componer un par de scripts que arrancan con una dirección u otra dependiendo de la línea de comandos que pondremos en diferentes entradas para lilo. El truco para el arranque dependiendo del entorno es utilizar la directiva `append` del `/etc/lilo.conf`.

Podemos ver esto en un interesante ejemplo de la configuración del equipo de Jaime Perea, desde el Dep. Astrofísica Extragaláctica del Instituto de Astrofísica de Andalucía (CSIC) en Málaga, España. Según escribe:

En mi portátil tengo dos entradas en el lilo.conf

```
image=/boot/vmlinuz-2.4.19-openmosix
    label=iaa
    root=/dev/hda7
    initrd=/boot/initrd-2.4.19.img
    append=" devfs=mount quiet iaa"
    read-only

image=/boot/vmlinuz-2.4.19-openmosix
    label=casa
    root=/dev/hda7
    initrd=/boot/initrd-2.4.19.img
    append=" devfs=mount quiet casa"
    read-only
```

esta línea pasa a /proc/cmdline si he arrancado usando la entrada iaa (la del trabajo) este fichero contiene

```
BOOT_IMAGE=iaa ro root=307 devfs=mount quiet iaa
```

Así dependiendo de ese último string puedo hacer un arranque u otro.

Yo lo hago con un script python bastante idiota que se ejecuta desde rc.local

```
#!/usr/bin/env python
#
# dependiendo de que opción le hemos pasado al kernel
```

```
# ejecuta un tipo de arrancada u otro
#
from string import *
import os
f = open('/proc/cmdline','r')
lin = split(f.read())
f.close()
op = lin[-1]
command = "/usr/local/snet/"+op
os.execl(command, '/')
```

si 'op' en este string es casa se ejecuta /usr/local/snet/casa que contiene lo siguiente

```
#!/bin/sh
# Configuracion de red de casa
#
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
#
# Direccion ip
ifconfig eth0 192.168.0.2 up
# nombre
hostname "cristina"
# ruta
route add -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.0.1 eth0
# DNS de wanadoo
echo "nameserver 62.37.220.75" > /etc/resolv.conf
echo "nameserver 62.37.228.20" >> /etc/resolv.conf
# mosix.map
echo "1 192.168.0.1 2" > /etc/mosix.map
cp /etc/auto.master.noiaa /etc/auto.master
/etc/rc.d/init.d/autofs restart
/etc/rc.d/init.d/openmosix start
/etc/rc.d/init.d/webmin restart
hostname "cristina"
```

El ordenador 192.168.0.1 es el hijo de mi casa y hace masquerading al portátil. Hay un script similar para las otras opciones de arrancada, por ejemplo si es iaa (mi trabajo) pues arranca cupsy hace otro ifconfig.

De hecho se puede hacer an mejor usando las herramientas de autodescubrimiento de open-mosix, así el portátil lo podras arrancar usando dhcp, pero ese negocio yo no lo conozco, ya que las direcciones que debo utilizar son fijas.

En otro orden de cosas. Tengo un cluster de 16 biprocesadores que van sobre openmosix pero con Myrinet... la verdad es que vuelan, pero es una solucin sub-óptima ya que el tcpipde las myrinet va sobre una emulación ethernet, así no consigo ese Ghz que debera tener de

comunicaciones y sólo me quedo en la mitad, pero bueno no está mal, al fin y al cabo nada me impide usar otros tipos de comunicaciones.

9.2. Nodos sin disco duro

Previo a intentar aventurarnos con este propósito (i.e. arrancar nodos *disk-less*) deberemos configurar un servidor DHCP para que las peticiones de dirección ip puedan ser atendidas, cuando un nodo intente arrancar (botar). A este servidor DHCP lo referenciamos a partir de ahora como *maestro* y hará las funciones de servidor de NFS, el cual exportará la totalidad del sistema de ficheros para los nodos *disk-less*, a partir de ahora referenciados como *esclavos*, cuando dispongan de ip asociada.

Para todo ello deberemos estar seguros de haber incluido soporte para servidor NFS en la configuración del kernel. Hay varios tipos de NFS aunque para el propósito que nos ocupa quizás el más recomendable sea *daemon-nfs*.

Para generar la estructura de sistema de ficheros para que los nodos esclavos puedan arrancar seguiremos estos pasos:

- calcularemos de 300 a 500 MB en el nodo maestro para cada esclavo que queramos montar
- crearemos un directorio para la totalidad del sistema de ficheros del cluster y haremos un enlace simbólico hacia */tftpboot*¹⁰
- crearemos un directorio al que llamaremos igual que la dirección ip del primer esclavo que queramos configurar, así por ejemplo introduciremos `mkdir /tftpboot/192.168.45.45`

Dependiendo del espacio de que dispongamos en el maestro podremos copiar todo el sistema de ficheros de éste en el directorio que sirve de raíz para el esclavo. En el caso que nuestro nodo maestro esté demasiado cargado para realizar la anterior operación, deberemos preocuparnos para que al menos quepan los directorios:

```
/bin
/usr/bin
/usr/sbin
/etc
/var
```

Hay que asegurarse de crear directorios vacíos para los puntos de montaje. Así por ejemplo la estructura de sistema de ficheros en */tftpboot/192.168.45.45* en el esclavo tendría que ser similar al directorio raíz (/) del maestro, de esta forma se mantendrían correspondencias como:

¹⁰este directorio, o cualquier enlace hacia él, es indispensable ya que los nodos esclavos buscarán en el directorio */tftpboot/direccion_ip_del_esclavo* del nodo maestro para arrancar.

El sistema de ficheros del primer nodo esclavo estaría listo a partir de ahora. Si tuviéramos más nodos con el mismo hardware podríamos aprovechar estas configuraciones y reutilizarlas. Se puede cambiar la configuración para el esclavo como se indica:

```
CONFIG_IP_PNP_DHCP=y
and
CONFIG_ROOT_NFS=y
```

Se recomienda utilizar el menor número de módulos posible (o ninguno) ya que la configuración es un poco delicada. Ahora deberemos crear un dispositivo `nfsroot`¹¹. Esto solamente se usa para parchear el kernel del esclavo que queremos botar por NFS.

```
mknod /dev/nfsroot b 0 255
rdev bzImage /dev/nfsroot
```

Aquí `bzImage` tiene que ser la imagen del kernel del nodo *disk-less* en cuestión. Podremos encontrarla en `/usr/src/linux-version/arch/i386/boot` tras su compilación.

Ahora tendremos que cambiar el dispositivo `root` por este kernel

```
rdev -o 498 -R bzImage 0
```

y copiar el kernel en un disquete

```
dd if=bzImage of=/dev/fd0
```

Resta configurar el servidor DHCP en el nodo maestro. Necesitaremos la dirección MAC (dirección hardware) de la tarjeta de red del primer esclavo.

La forma más fácil de obtener esta dirección es botar el cliente (esclavo) con el disquete de arranque que hemos creado, que fallará en su propósito de arrancar el nodo, pero nos dirá su dirección MAC. Si el kernel ha estado configurado correctamente para el esclavo, el sistema arrancará desde el disquete botando el kernel *disk-less*, detectando su tarjeta de red y enviando sendas peticiones DHCP y ARP. Ahora sabremos si hemos introducido la dirección correcta, será algo parecido a `68:00:10:37:09:83`. En tal caso pasaremos a editar el fichero `etc/dhcp.conf` como el siguiente ejemplo:

```
option subnet-mask 255.255.255.0;
default-lease-time 6000;
max-lease-time 72000;
subnet 192.168.45.0 netmask 255.255.255.0 {
```

¹¹para información más detallada podemos consultar la documentación del proyecto Beowulf.

```

    range 192.168.45.253 192.168.45.254;
    option broadcast-address 192.168.45.255;
    option routers 192.168.45.1;
}
host firstslave
{
    hardware ethernet 68:00:10:37:09:83;
    fixed-address firstslave;
    server-name "master";
}

```

Ahora podremos iniciar DHCP y NFS con sus scripts de inicialización

```

/etc/init.d/nfsserver start
/etc/init.d/dhcp start

```

Podremos botar nuestro primer esclavo con el disquete, otra vez. Ahora tendría que funcionar: tras reconcer su targeta de red el esclavo obtendrá su dirección ip desde el servidor DHCP y su sistema de ficheros por NFS.

Tendremos que añadir el fichero `.rhost` en el directorio `/root` (para usuario root) en cada miembro del cluster, sería algo como esto:

```

node1  root
node2  root
node3  root
....

```

También tendremos que activar el login remoto por rsh en el `/etc/inetd.conf`, en el que tendríamos que tener las siguientes líneas:

- si nuestra distribución linux utiliza `inetd`

```

shell  stream  tcp      nowait  root    /bin/mosrun mosrun -l -z /usr/sbin/tcpd :
login  stream  tcp      nowait  root    /bin/mosrun mosrun -l -z /usr/sbin/tcpd :

```

- si utiliza `xinetd`

```

service shell
{
    socket_type      = stream
    protocol        = tcp
    wait            = no
    user            = root
}

```

```

server          = /usr/sbin/in.rshd
server_args    = -L
}
service login
{
socket_type    = stream
protocol       = tcp
wait           = no
user           = root
server         = /usr/sbin/in.rlogind
server_args    = -n
}

```

Seguidamente tendríamos que reinicializar `inetd`

```
/etc/init.d/inetd restart
```

tras lo cual se leerá la nueva configuración. Puede que la aplicación de configuración de nuestra distribución linux difiera de lo explicado hasta ahora. En cualquier caso sería recomendable utilizar `ssh` para el acceso remoto en vez de `rsh` si éste nos da problemas de configuración o simplemente no podemos garantizar la seguridad del sistema. `openMosixview` será accesible con los dos métodos.

Configurar `ssh` para un login remoto sin contraseñas es un poco delicado, eche un vistazo al capítulo que trata sobre `openMosixview` de este mismo manual.

Si quisiéramos copiar archivos a un nodo en nuestro cluster *disk-less* tendríamos dos posibilidades:

- utilizar `rcp` o `scp`
- copiar directamente, en el nodo maestro, los archivos en el sistema de ficheros del nodo esclavo

Así los siguientes comandos darían el mismo resultado

```

rcp /etc/hosts 192.168.45.45./etc
cp /etc/hosts /tftpboot/192.168.45.45/etc/

```

10. Problemas más comunes

10.1. No veo todos los nodos

Antes de nada deberá asegurarse de estar utilizando la misma versión del kernel en todas las máquinas y que todos ellos hayan estado debidamente parcheados.

Ejecute `mosmon` y pulse `t` para ver el total de nodos que están funcionando.

Si este proceso le advierte que no tiene openMosix funcionando en alguna máquina, entonces asegúrese de haber incluido su dirección ip en el `/etc/mosix.map` (no use 127.0.0.1 ya que probablemente tendremos problemas con el servidor de nombres/dhcp. Si openMosix puede ver su máquina entonces probablemente está protegido por un firewall que no deja actuar debidamente a openMosix.

En caso contrario el problema se encuentra la mayor parte de las veces en los nodos que no se muestran. En el caso de tener varias targetas nic en cada nodo tendremos que editar el fichero `/etc/hosts` para tener una línea con

```
non-cluster_ip cluster-hostname.cluster-domain cluster-hostname
```

Tendremos que configurar debidamente la tabla de enrutamiento, tarea que está fuera del alcance de este COMO.

Otros posibles problemas pueden venir por los diferentes parámetros de configuración del kernel, especialmente si usamos clusters con la opción de topología de red compleja. Con esa opción hay que vigilar de no utilizar el mismo valor que aparece en la opción de *Maximum network-topology complexity support* en cada nodo.

10.2. FAQ de openMosix *-preguntas más frecuentes*

10.2.1. General

¿Qué es openMosix?

El sistema openMosix es una extensión del kernel del sistema operativo GNU/Linux para un cluster de una sola imagen (o SSI). Está basado en el proyecto MOSIX del profesor Barak, pero licenciado bajo licencia GNU (General Public License, GPL).

¿Qué significa *cluster de una sola imagen*?

Hay muchas variedades de clusters, y un cluster de una sola imagen tiene copias de un único kernel de sistema operativo.

¿Existe una página web para openMosix?

Sí es <http://openmosix.sourceforge.net>. También está la página web del proyecto en SourceForge <http://sourceforge.net/projects/openmosix>

¿Hay una lista de correo para openMosix?

Sí claro, temenos dos:

- Para discusiones generales está `openmosix-general@lists.sourceforge.net` , cuya página de información general es `http://lists.sourceforge.net/lists/listinfo/openmosix-general`
- Para programadores úsese `openmosix-devel@lists.sourceforge.net`, cuya página de información general es `http://lists.sourceforge.net/lists/listinfo/openmosix-devel`

¿Puedo contribuir al proyecto openMosix?

Por supuesto! El proyecto ya tiene más de 10 contribuidores. A diferencia del sistema de manutención del kernel de linux, Moshe Bar (coordinador principal) nombra desarrolladores oficiales y les otorga a éstos la clave de commit (*commit bit*) del directorio CVS para el código fuente de openMosix, de manera similar al sistema usado para FreeBSD.

De momento se está buscando programadores familiarizados con el kernel de linux para trabajar en nuevas características como checkpoint/restart.

Escriba a `moshe@moelabs.com` si está interesado contribuir en al proyecto openMosix.

¿Quién posee la licencia de openMosix?

Todo el código fuente de MOSIX está licenciado por el profesor Amnon Barak de la Hebrew University of Jerusalem. Todo el código fuente de openMosix está licenciado por Moshe Bar de Tel Aviv. El sistema openMosix no contiene ningún código fuente que no esté licenciado bajo la GPL (por ejemplo código fuente de MOSIX).

¿Es openMosix una versión de MOSIX?

Originalmente openMosix era una versión de MOSIX, pero se ha desarrollado en una plataforma avanzada de clustering. El sistema openMosix no contiene ningún código fuente que no esté licenciado bajo la GPL. Comparado con MOSIX, hay cierta cantidad de características que ya han sido agregadas:

- una versión para la arquitectura UML (linux de modo de usuario)
- nuevo y más ordenado código de migración
- un mejor balanceador de trabajo
- reducción de latencias del kernel
- soporte para Dolphin e IA64 (arquitectura 64-bit)
- un sistema de instalación simplificado usando paquetes RPM
- gran cantidad de documentación

¿Por qué se separó openMosix de MOSIX?

La cuestión principal era que MOSIX **no** está licenciado bajo una licencia abierta.

10.2.2. Obteniendo, compilando, instalando y funcionando con openMosix

¿Dónde puedo conseguir openMosix?

Los fuentes oficiales para openMosix pueden encontrarse aquí ¹². Asegúrese de leer la documentación antes de bajar los archivos.

¿Puedo mezclar nodos MOSIX y openMosix en el mismo cluster?

NO! Como en MOSIX, no es recomendado mezclar nodos porque los protocolos están ligados a cambiar sin anuncios entre versiones. Además, cada nueva versión provee correcciones de errores anteriores que vale la pena eliminar.

¿Cómo compilo openMosix?

Primero desempaquete tanto el código fuente del kernel de linux como la distribución correspondiente de openMosix en un directorio, por ejemplo

```
/usr/src
$ cd /usr/src
$ tar xzf linux-2.X.Y.tar.gz
$ gunzip openMosix2.X.Y.gz
```

Aplique los parches de openMosix al código fuente original del kernel de linux con el comando

```
$ patch -p1 openMosix2.X.Y-Z
```

El directorio `/usr/src/linux-2.x.xx` ahora contiene el código fuente del kernel 2.x.xx de linux con los parches de openMosix aplicados. Haga un `compile` e instale el kernel obtenido como cualquier otro kernel.

¿Qué son las userland tools (herramientas de usuario)?

Las herramientas de usuario son una colección de herramientas administrativas para examinar y controlar un nodo openMosix.

10.2.3. Preguntas del kernel (núcleo)

¿Qué versiones del kernel están soportadas por openMosix?

El último kernel de linux soportado es 2.4.20. Versiones futuras de la serie 2.4 y la serie 2.5 también van a ser soportadas por openMosix.

Estoy tratando de compilar el kernel con el parche de openMosix. ¿Qué versión de compilador debo usar?

¹²http://sourceforge.net/project/showfiles.php?group_id=46729

La versión recomendada para los kernels es gcc-2.95.3. Este es no sólo un requerimiento de openMosix, sino también del kernel de linux. Sin embargo, esto no lo previene a ud. de tener gcc-2.95.3 para los kernels y gcc-3.x para compilaciones que no son para el kernel.

Notas adicionales: actualmente ya se ha conseguido un código fuente para compilaciones con versiones 3 del gcc.

He compilado el kernel de las fuentes. ¿Cómo hago para agregarlo al cargador de inicio (*bootloader*) (LILO, GRUB, otro)?

El kernel de openMosix es como cualquier otro kernel porque openMosix es tan solo una extensión del kernel, y va a ser tratado como un kernel estándar. Use los métodos normales del cargador de inicio para agregar un kernel.

He instalado una distribución de linux y dice que el kernel es x.x.x-x. El README de openMosix dice que no debemos mezclar kernels. ¿Significa esto que el RPM para el kernel openmosix-x.x.x-y no va a funcionar en mi máquina?

No. Cuando decimos que no se deben mezclar versiones de kernels no referimos a los parches de openMosix. Mientras todas tus máquinas usen las mismas versiones del kernel de openMosix.

Nótese que con versión nos referimos al número de versión de openMosix; ud. puede mezclar kernels de openMosix que son para diferentes arquitecturas de hardware (por ejemplo, si ud. va a instalar el kernel openmosix-2.4.18-4, puede usar el RPM openmosix-2.4.18-4-i686 en una máquina con una Pentium II o superior, y el kernel openmosix-2.4.18-4-athlon en una máquina con un procesador Athlon).

10.2.4. Sistema de ficheros

¿Puede alguien explicarme la diferencia entre MFS y DFSA, y por qué me conviene tener DFSA?

DFSA significa Direct File System Access (acceso directo al sistema de archivos) y es una optimización. Permite que procesos remotos ejecuten llamadas de sistema localmente en vez de mandarlas al nodo local. MFS significa Mosix File System (sistema de archivos Mosix) y permite que todos los nodos en un cluster tengan acceso al sistema de archivos de todos los otros nodos. DFSA se ejecuta encima de un sistema de archivos de cluster, en este caso MFS.

¿Qué es oMFS, cómo lo uso, y dónde lo consigo?

El sistema de archivos openMosix (oMFS) es la implementación de MFS en openMosix (véase la pregunta anterior para detalles acerca de MFS). oMFS se instala cuando ud. configura un kernel de openMosix con la opción oMFS (viene estándar con los archivos RPM). Si ud. instala su propio kernel, le sugerimos que también use la opción DFSA (vea la pregunta anterior para detalles acerca de DFSA). El uso y la administración de oMFS es muy similar a NFS, pero oMFS provee ciertas ventajas sobre NFS como

- consistencia de cache
- timestamp
- link consistency

La instalación de DFSA encima de oMFS asegura que el proceso se mueva con la información y vice versa, cuando hga sentido. Asegúrese de leer más acerca de oMFS y de como usarlo en el presente COMO de openMosix.

10.2.5. Programando openMosix

En general, ¿cómo escribo un programa que utilice openMosix?

Escriba sus programas de manera normal. Cualquier proceso que sea creado es un candidato a ser migrado a otro nodo.

Un consejo para poder aprovechar al máximo la tecnología que openMosix pone a su alcance sería servirse de la sentencia *fork()* de UNIX. Como se ha comentado en este COMO, openMosix hereda de MOSIX el *fork-and-forget* que realiza la migración automática de cualquier subproceso que genere cualquiera de los procesos que tenemos corriendo en nuestro cluster.

¿Es posible escribir programas para openMosix con Perl?

Evidentemente sí. Ud. deberá usar `Parallel::ForkManager`, disponible en CPAN o directamente de <http://www.cpan.org/authors/id/D/DL/DLUX/Parallel-ForkManager-x.x.x.tar.gz>.

10.2.6. Recursos

¿Dónde puedo encontrar detalles técnicos acerca de openMosix?

Aquí hay algunos links (en inglés):

- El navegador de código fuente de Brian Pontz¹³
- Detalles de openMosix - cómo funciona openMosix¹⁴
- LTSP y openMosix - un HOWTO¹⁵
- Sistemas operativos distribuidos - una descripción general de openMosix¹⁶

¿Qué otras fuentes hay?

La meta del esqueleto de ejecución en paralelo es de crear un dispositivo único que permita crear un cluster de PCs estándar de manera fácil. Véase <http://parallel.sourceforge.net>

¹³<http://www.openmosix.org/lxr/source>

¹⁴http://sourceforge.net/docman/display_doc.php?docid=10390&group_id=46729

¹⁵http://sourceforge.net/docman/display_doc.php?docid=10431&group_id=46729

¹⁶http://sourceforge.net/docman/display_doc.php?docid=9562&group_id=46729

10.2.7. Miscelánea

Mi máquina no puede ver otras máquinas. ¿Qué está pasando?

Cuando ud. ejecuta `mosmon`, presione la tecla 't' para ver el total de máquinas corriendo. Si le dice a ud. que `openMosix` no está corriendo, asegúrese que la IP de su máquina está incluida en `/etc/mosix.map`, o de que ud. esté corriendo la herramienta `omdiscd`. Nunca agregue la IP 127.0.0.1 a su `mosix.map`.

Si `mosmon` no le indica a ud. que `openMosix` no está corriendo, observe qué otras máquinas aparecen en la utilidad de monitoreo. Si ud. ve sólo a la máquina en la que está corriendo `mosmon`, es probable que su máquina esté detrás de una firewall que está bloqueando `openMosix`. Asegúrese también de que TODAS sus máquinas estén listadas en su `/etc/mosix.map`, o que todas sus máquinas estén corriendo `omdiscd` (véase el capítulo pertinente para más información acerca de `omdiscd`).

Si su máquina tiene dos (o más) tarjetas de red, ud. deberá editar el archivo `/etc/hosts` y agregar una línea con el siguiente formato

```
[IP externa] [nombre en cluster].[dominio en cluster] [nombre en cluster]
```

(sin []). Es probable que ud. también deba configurar una tabla de routing.

Si todo lo anterior no funciona, es posible que ud. esté utilizando diferentes parámetros de kernel en las máquinas. Esto es especialmente importante cuando ud. está usando la opción 'Support clusters with a complex network topology'. Asegúrese de usar los mismos valores en cada máquina.

¿Cuál es la diferencia entre `/etc/misix.map`, `/etc/hpc.map` y `/etc/openmosix.map`?

Estos archivos representan diferentes estados de evolución de `openMosix`. El archivo `/etc/mosix.map` es el mapa original de MOSIX, mientras que `/etc/hps.map` es un archivo de mapa utilizado por versiones anteriores de `openMosix`. El archivo actual es `/etc/openmosix.map`. (Nótese que `openMosix` puede utilizar cualquiera de estos archivos - si ud. está instalando `openMosix` en un cluster con MOSIX instalado, no hace falta renombrar `/etc/mosix.map` a `/etc/openmosix.map`).

Estoy recibiendo un mensaje que dice

```
setpe
the supplied table is well-formatted, but my IP address (127.0.0.1) is not there!
```

Ud. deberá modificar el archivo `/etc/hosts`. En máquinas con Red Hat instalado ud. encontrará una línea similar a ésta

```
127.0.0.1 hostname.domain.com localhost
```

Si `hostname.domain.com` tiene una dirección de IP de 192.168.10.250, cualquier utilización de `hostname.domain.com` va a usar 127.0.0.1 en vez de la IP que debiera tener. Ud. va a tener que modificar su archivo `/etc/hosts` para que contenga la siguiente información

```
192.168.10.250 hostname.domain.com
127.0.0.1      localhost
```

Quiero instalar openMosix en mi máquina, pero temo que es demasiado lenta.

Cualquier máquina puede beneficiarse con el uso de openMosix. Mientras su máquina sea del x86 (Intel Pentium o compatible) or mejor, ud. podrá correr openMosix. Ni siquiera hace falta tener máquinas del mismo tipo, y ud. podrá agregar y remover máquinas cuando se le de la gana! Sin embargo, Charles Nadeau recomienda el uso de una red 100-base-T o mejor.

¿Bajo qué condiciones se puede usar VMware con openMosix?

Si ud. desea correr VMware en una máquina con openMosix instalado para aprovechar la capacidad de computación de las máquinas remotas, no hay problema. Por otro lado, ud. no podrá correr openMosix dentro de sesiones de VMware y dejar que las sesiones balanceen su carga porque VMware tiene un defecto en su emulación de Pentium que causa que VMware (no openMosix) falle cuando un proceso migre.

¿Qué arquitecturas aparte de x86 (por ejemplo SPARC, AXP, PPC) son soportadas por openMosix?

openMosix solo corre en x86.

¿Existe una utilidad paralela de 'make' para openMosix similar a 'MPmake'?

Ud. puede usar el 'make' de gcc con la opción '-j'. Por ejemplo,

```
make -j 10
```

ejecuta make con 10 procesos en paralelo.

11. Precauciones

11.1. Procesos bloqueados

Si por alguna razón los procesos están bloqueados en el nodo raíz, podemos añadir las siguientes líneas dentro de `~/.profile` como medida para desbloquear los procesos al cabo de un intervalo de tiempo como medida de migración automática:

```
if [ -x /proc/$$/lock ]; then
    echo 0 > /proc/$$/lock
fi
```

11.2. Escogiendo tus procesos

Probablemente querrás probar tu configuración antes de decidir qué programas quieres activar para la migración. Por ejemplo, si estás ejecutando KDE2 en una máquina poco potente y tienes otra máquina relativamente más potente en el cluster, puedes configurar que procesos como KMail (que requieren muchos recursos) migren.

11.3. Java y openMosix

Green Threads JVM, permite la migración ya que cada hilo (*thread*) de Java es un proceso separado.

Los demás *threads* de Java que no sean *Green Threads* no podrán migrar en linux ya que openMosix será incapaz de migrar aplicaciones que los utilicen.

Si dispusiéramos del código fuente de nuestra aplicación Java podríamos compilar la aplicación nativa. En este caso podría migrar a otros nodos.

12. Para más información

Para información actualizada inscríbete a las listas de correo¹⁷ del proyecto openMosix y de openMosixview¹⁸.

Puede ser realmente útil ver la página *wiki* que tiene montada Kris Buytaert¹⁹, podrás encontrar los aspectos pendientes de documentación y los comentarios sobre testeos de diferentes instalaciones de clusters que algunos usuarios han querido compartir con la comunidad. También tiene una lista de enlaces.

Puedes leer el log de la conferencia que dio David Santo Orcero²⁰, desarrollador de las herramientas de usuario, para UMEET2002. Dani y yo mismo nos encargamos de la traducción en castellano en el canal #redes del mismo servidor IRC²¹.

Otra charla que dio David, algo anticuada a día de hoy pero que puede ayudarte a ver cómo ha evolucionado el clustering, puedes leerla (en inglés) en <http://umeet.uninet.edu/conferencias/23-1>

¹⁷<http://openmosix.sourceforge.net/mailling.html>

¹⁸<http://lists.sourceforge.net/lists/listinfo/mosixview-user>

¹⁹<http://howto.ipng.be/openMosixWiki/>

²⁰<http://umeet.uninet.edu/umeet2002/talk/2002-12-17-linux1.txt.html>

²¹<http://umeet.uninet.edu/umeet2002/talk/2002-12-17-redes1.txt.html>

13. APÉNDICE A: Aplicaciones funcionando

Este apéndice contendrá una lista de aplicaciones que han sido probadas y otra que mostrará las NO pueden funcionar, en openMosix.

13.1. Programas que funcionan

- **Utilidades MJPEG.** Utilizan *pipes* de entrada y salida (i/o) intensivamente, hecho que hace que funcionen excepcionalmente bien en clusters de pequeño y mediano tamaño. La codificación y decodificación se realizan en el nodo raíz pero los filtros migrados en los nodos incrementan el rendimiento en la compresión de ficheros de alta calidad (que requieren un mayor procesamiento).
- **bladeenc.** Esta utilidad sirve para ripear rápidamente nuestros ficheros de audio mp3.
- **Povray.** Podemos dividir nuestros frames del trabajo de renderización en múltiples procesos desde un shell script o utilizar la versión paralela (PVM) para que se haga automáticamente.
- **MPI.** Entre MPI y openMosix parece existir una totalidad compatibilidad.
- **FLAC.** Se trata de un codificador de audio *lossless* (sin pérdidas). <http://flac.sourceforge.net>

Podremos encontrar información más actualizada en las referencias Wiki del HOWTO de Kris Buytaert²².

13.2. Programas que NO funcionan

Las aplicaciones que utilizan memoria compartida funcionan en un linux estándar, pero no migran. Tampoco podrán migrar programas que hagan uso de recursos que no puedan migrar, como por ejemplo pasa con todas las aplicaciones que puedan apoyarse en Java green thread JVM's.

- **Programas Java que utilizan *threads* nativos.** Estos programas no migrarán porque utilizan memoria compartida. Los *Green Threads JVM's* permiten migración pero cada thread en un proceso separado.
- **Aplicaciones que utilizan threads.** El hecho de que no migren los threads no es una limitación de openMosix sino de Linux. Contrariamente a las plataformas como Solaris donde los threads son procesos *ligeros* con su propio espacio de memoria, en Linux no poseen dicho espacio. Si hacemos un `ps` en Linux podremos ver cada thread ya que cada uno será una tarea en el scheduler. No obstante cada una de estas tareas no puede funcionar por ella misma ya que necesita el espacio de direcciones donde fue lanzada. De esta manera queda pues como hecho imposible poder migrar *pthread* a otro nodo sin poderlo conectar con su espacio de direcciones.

²²<http://howto.ipng.be/openMosixWiki/index.php/work%20smoothly>

- **Phyton** con el threading activado.
- **MySQL** utiliza memoria compartida.
- **Apache** utiliza memoria compartida.
- **Mathematica** utiliza memoria compartida.
- **SAP** utiliza memoria compartida.
- **Oracle** utiliza memoria compartida.
- **Baan** utiliza memoria compartida.
- **Postgres** utiliza memoria compartida.

Podremos encontrar información más actualizada en la página Wiki²³.

²³<http://howto.ipng.be/openMosixWiki/index.php/don't>

14. APÉNDICE B: Acrónimos

COTS - Common Off The Shelf, *componentes comunes en el mercado*

SCI - Scalable Coherent Interface

ATM - Asynchronous Transmission Mode

PVM - Parallel Virtual Machine

MPI - Message Passing interface

SMP - Symmetric Multi Process, máquinas con varios procesadores en la misma placa madre

SSI - Single System Image, permet veure un cluster com un únic equip

LVS - Linux Virtual Machine

PVFS - Parallel Virtual File System

ESS - Earth & Space Sciences

API - Application Programming Interface

DFSA - Direct File System Access

MFS - Mosix File System

VFS - Virtual File System

DIPC - Distributed Inter Process Communication

NUMA - Non Uniform Memory Access

FS - File System, *sistema de ficheros*

GUI - Graphical User Interface

UHN - Unique Home Node

DHCP - Dynamic Host Configuration Protocol

CPU - Central Process Unit, *unidad central de procesamiento*

DMA - Direct Memory Access, *acceso directo a memoria*

IP - Internet Protocol

IPC - Inter Process Communication

MHz - Megahercios

NFS - Network File System, *sistema de ficheros en red*

NIC - Network Internet Controller

NOW - Network of Workstations, *red de estaciones de trabajo*

PC - personal Computer

POSIX - Portable Operating System based on UNIX

RAM - Random Access Memory, *memoria de acceso aleatorio*

SO - Sistema Operativo

TCP - Transfer Control Protocol, *protocolo fiable orientado a bytes para intercambio de información entre ordenadores que establecen una conexión*

UID - User Identification

GID - Group Identification

ERRANDO DISCITUR