Nuevo modelado de computación paralela con clusters Linux

miKeL a.k.a.mc^2 ≡ Miquel Catalán i Coït

VI Congreso HISPALinux
MADRID - 24 septiembre 2003

...porque en el camino hacia la sociedad del *software* libre deberíamos estar todos.

Menciones especiales para:

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 (a.k.a. irbis)
Moshe Bar principal desarrollador, autor de MFS y DFSA

HISPALINUX

>> PRÓLOGO

Los sistemas cluster hace años que fueron diseñados, la computación paralela y distribuida no es ninguna novedad en el año 2003. No obstante no había sido hasta ahora que el usuario havia empezado a necesitarlas. La tecnología del silicio está llegando a sus postrimerías y los computadores cuánticos aún están en fase de desarrollo.

Mientras grandes empresas, instituciones y universidades selectas disponen –desde hace años– de grandes computadoras superescalares el usuario había estado relegado –hasta ahora– a máquinas SMP en el mejor de los casos.

Pero todo esto está cambiando, la demanda de rendimiento no puede ser suplida por la arquitectura mono-procesador y menos por los x86 compatibles. La solución que han adoptado los fabricantes ha estado saltar a arquitecturas de 64 bits o aumentar más aún las frecuencias, pero simplemente tratan de desplazar el problema en el tiempo.

En este marco toman mayor importancia los clusters, concebidos para proporcionar cálculo paralelo con componentes habituales en el mercado. Estaciones de trabajo conectadas por red trabajando de forma cooperativa que permiten aumentar notablemente las prestaciones de cualquier sistema informático.

En esta ponencia se sentarán los conocimientos básicos para llegar a construir un cluster openMosix a partir de cero, es decir, desdel hardware.

La documentación aportada dejará claras las grandes capacidades tecnológicas de openMosix, que se podrán utilizar en proyectos de pequeña, mediana o gran dimensión gracias a su escalabilidad y flexibilidad. También se enfatizará en la importancia de la aplicación de las tecnologías de programario libre como mejor solución para poner en manos del usuario las mejores herramientas que posibilitaran un futuro enriquecedor tanto tecnologica como socialmente. Este entorno es el que mejor defiende la propia congruencia de intenciones ya sea en la lógica docente donde priman -o tendrían que hacerlo- el conocimiento y su libre difusión, o dentro de la lógica empresarial -donde priorizamos el beneficio al menor coste posible-.

PALABRAS CLAVE: **cluster, linux, openmosix, supercomputación, ssi, diskless.**

Índice general

Capítulo 1

Preliminares

Now this is not the end. it's not even the beginning of the end.
But it's, perhaps, the end of the beginning.

Winston Churchill

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.

1.1. Computación paralela

Computación paralela puede entenderse en base a lo que técnicamente es, el cómputo en varias unidades de procesamiento, o en base a la finalidad que persigue, el mayor aprovechamiento de un mayor número de recursos para la reducción del tiempo invertido en un proceso informático.

Uno de los anhelos, seguramente el pioritario, en una sociedad tecnológica cada vez más abruamda por la ingente cantidad de operaciones necesarias para el análisis de la realidad que nos envuelve, es sin duda el poder computacional. Esto es a lo que se ha dado referencia en la anterior segunda definición, este es el objetivo. Pero existen muchas técnicas que intentan acelerar algorismos. En la computación paralela se ha optado por añadir la gestión de los recursos en el espacio, a la del tiempo ya muy utilizada en sistemas monoprocesador.

La mayor diferencia que implica esta reflexión es sin duda la primera definición, i.e. disponer de varios procesadores, o recursos para mayor generalización. Esto difiere notablemente de la computación concurrente. Cuando se hace referencia a supercomputación debe ampliarse el concepto de multitarea al hecho de tener varias tareas ejecutándose realmente a la vez.

¿Como se concibe pues un sistema de computación paralela? Primeramente debe disponer de varios procesadores. Las diferencias entre las diferentes arquitecturas con este requisito estriban en su disposición y gestión de la memoria, tipo y topología de la red de comunicación, entre otras.

Disponiendo pues de varias unidades de procesadmiento el siguiente paso es pensar en su comunicación. Es obvio que los diferentes procesadores deben saber en cada instante el estado global del cálculo para asignarse su parte, y devolver el resultado una vez haya sido realizado. Y ya tenemos dos de los principales elementos de estas arquitecturas: las unidades de procesamiento y los elementos de comunicación. Todos estos recursos requieren ser gestionados, y esa es la parte encargada al *software*, el tercer elemento imprescindible.

El software diferirá tanto más cuanto más distintas sean las arquitecturas –y sus recursos– que deba gestionar. En supercómputo básicamente se diferencian dos tipos de arquitectturas tipo: las vectoriales y las distribuidas.

Los **sistemas vectoriales** suelen ser más –mucho más– caros por el hecho de estar construidos con una tecnología mucho más vanguardista. Las prestaciones de estos sistemas pueden llegar a ser incomparables con las modestas computadoras domésticas del mercado de gran consumo aún contando con *solamente* cuatro procesadores, por ejemplo. En estos sistemas se apuesta po un nivel tecnológico superior, con todo lo que esto conlleva.

Los **sistemas distribuidos** o **clusters**, como su propio nombre indica, implementan una política de reparto y decentralización de recursos. Entre las ventajas con las que cuentan básicamente sobresalen una mayor escalabilidad y un menor coste de construcción. Ambos factores han sido decisivos para atraer a un público más extenso, con menor poder adquisitivo aunque no por esto menos exigente.

Para terminar de entender las diferencias del trasfondo de estas arquitecturas suele darse a un ejemplo fácil de comprender. Puede pensarse una arquitectura vectorial como un amigo muy listo. Si tuviésemos un gran problema seguramente pensaríamos que nosotros tardaríamos demasiado tiempo en realizarlo, así que apostaríamos por ponerlo en manos de un genio como nuestro amigo que sin duda lo realizaría con una parte del tiempo que invertiríamos nosotros. Las arquitecturas distribuidas se mueven en un contexto más social, es decir, buscaríamos varios de nuestros amigos de clase para encargarles deberes a cada uno de ellos. Juntaríamos los resultados aportados por cada uno de nosotros y finalmente podríamos sacar el resultado total.

Las posibilidades de encontrar algunos amigos en clase dispuestos a ayudarnos son mayores que la de encontrar un amigo muy listo, en caso de haberlo. Otro aliciente para tomar este camino es la envergadura de los problemas con los que solemos tratar. Un sistemas de diez ecuaciones puede ser una árdua tarea que nuestro

amigo listo podría realizar con más o menos esfuerzo, pero intentar encontrar un amigo capaz de enfrentarse a la resolución de una matriz cuadrada de factor 300 resultaría harto complicado.

La tecnología ha demostrado dar poco márgen a las iniciativas que dan una sola solución pero muy eficiente, que suele dejar de serlo en poco tiempo al aparecer una mejor. Esta carencia tecnológica puede cubrirse dando muchas soluciones, y uniéndolas.

Toda esta política ha estado muy estudiada por desarrolladores del mundo entero y ahora mismo estamos en un marco donde cada una de las dos intenta sobreponerse a la otra, siempre dando cifras de rendimiento. Entre las gentes que trabajan en estos proyectos contamos con grandes multinacionales del hardware y del software, como son IBM o Sun. No obstante los regulares en el uso del programario libre también podemos disponer de esta tecnología en nuestras arquitecturas –en referencia al x86– gracias a proyectos como openMosix.

El proyecto openMosix tiene su antepasado en MOSIX, un sistema de clustering reencarnado 9 veces, y cada una de ellas para correr en un sistema operativo distinto. Diferencias entre los mismos desarrolladores generaron el *fork* que hoy se conoce como openMosix. La elección de Linux como sistema operativo para su sistema de clustering se ha basado en las siguientes evidencias:

- el so de Redmond está más extendido en los x86, pero nunca ha sido una opción en temas de supercomputación,
- los sistemas UNIX propietarios –sean true64, Solaris, HPUX, ...– requieren hardware propietario,
- el desarrollo de Linux es rápido, abierto y sobretoddo muy evolutivo,
- Linux se preocupa para no dejar obsoleto el hardware que ya no está a la última.

1.2. Clasificación de los clusters

Un cluster puede pertenecer a la vez a tres subconjuntos:

1. Alta disponibilidad (*high availability, HA*). Sus recursos están dispuestos de manera que sean redundantes, es decir, haya la misma arquitectura duplicada para que cuando el sistema que sirve el servicio falle, su clon pueda tomar el control y servir él.
2. Alto rendimiento (*high performance, HP*). Su finalidad es disponer del máximo número de *flops*, unidad de medida para las operaciones de pun to flotante realizadas por segundo de tiempo.
3. Balanceo de carga (*load balancing*). Existe una jerarquía de maestros y esclavos. Los maestros reciben la petición del servicio y mandan la tarea a los esclavos para que la ejecuten. Éstos devuelven el resultado y el servidor responde.

A veces será difícil clasificar un sistema dentro de un solo grupo, aunque prácticamente es imposible que uno pueda considerarse en los tres a la vez. Un cluster openMosix se encuentra en el segundo grupo.

Capítulo 2

SSI y openMosix

It is true greatness to have in one
the frailty of a man and the security of a god.

Seneca

2.1. SSI. Un nuevo paradigma

SSI es el acrónimo de *Single System Image*. Como puede deducirse lo que pretende conseguir es una imagen única del sistema cluster, mostrándolo al usuario como un todo, siendo transparente todo el montaje de nodos que haya dispuestos para su funcionamiento.

Los pilares en que SSI basa su política son básicamente tres:

1. configuración sin esquema *master-slave*. Esto permite una total decentralización de las políticas de asignación de recursos, de migración de procesos, *etc*.
2. cada nodo será capaz de auto-gobernarse. Es una implicación del enunciado anterior, que deshace cualquier jerarquía entre nodos y permite a cada uno de ellos tomar decisiones sobre los procesos que acepta, los que permite migrar, la política de disponibilidad sobre sus recursos, *etc*.
3. el sistema se percibe como una imagen única de la totalidad de sus recursos. Esta es la carcaterística que los identifica como SSI.

Las ventajas que esto aporta son varias y a la vez interesantes:

1. facilidad de adminstración,
2. escalabilidad –respecto al número de nodos–.
3. ocultación de la complejidad de la topología de red y de arquitectura a los usuarios,
4. las aplicaciones no necesitan ser reprogramadas ni recompiladas,
5. facilidad para implementar un sistema de balanceo automático de carga para cada nodo,
6. las aplicaciones pueden ser instaladas en un nodo, y funcionar en cualquier otro.

Toda esta política conlleva ciertas dificultades de implementación, dificultades que por otra parte han estado salvadas por los desarrolladores con una eficacia que todavía se está perfeccionando.

1. una algorísmica de balanceo de carga adaptativa a las cambiantes condiciones del sistema,
2. migración de procesos entre nodos,
3. sistema de ficheros para el cluster.

Todo ello, enfatizando una vez más en el concepto de SSI, se pretende que sea transparente tanto al

- usuario, que no se percatará de la topología del cluster,
- nodo, que no detectará intrusos en su lista de procesos en ejecución ni cuando procesos foráneos lleguen a él,
- propio proceso, que nunca se percatará de que se le está trasladando a otro nodo.

Han habido y seguramente surgirán nuevos proyectos de *software* –libre o no , sirvan el SYSPlex de IBM, el openSSI, CompaqSSI o MOSIX– intentando conseguir todas las caracterizaciones de un sistema SSI, no obstante ha sido concretamente uno el que ha conseguido reunir a un mayor número de desarrolladores, a la vez que usuarios dispuestos a jugar con la instalación, i.e. el proyecto openMosix.

2.2. La mejor alternativa SSI: openMosix

El fruto más jugoso del proyecto openMosix es un parche para el kernel de Linux –siempre habrá uno para cada nueva versión del mismo–. Aplicando este *patch* a Linux obtenemos una serie de nuevas posibilidades, transformanddo literalmente el kernel en un núcleo totalmente preparado para la computación paralela.

Los ajustes que permite este parche se verán en la documentación que le acompañ, ahora es más importante detallar sus características, añadidas a las que ya de por sí proporciona cualqueir sistema SSI. Estas son:

1. El parche openMosix aumenta en un 3 % el tamaño del núcleo de Linux. Estas cerca de 55.000 líneas de código C y ensamblador lo hacen medianamente intrusivo aunque, como se verá, nunca ha supuesto un paso atrás a la estabilidad del sistema en el que sea arrancado.

2. El nuevo kernel tendrá nativamente la capacidad de migrar procesos entre máquinas conectadas en red. La política de migración no es trivial y no se describirá, por ahora basta decir que no permite a cualquier poceso ser migrado, es decir, ejecutarlo en otra computadora donde su ejecución sea más ágil. Hay un cierto número de limitaciones –concretamente y a día de hoy, cuatro– que imposibilitan a cualquier proceso que cumpla cualquiera de ellas ejecutarse migrado.

3. Linux se convierte en un sistema de ejecución paralela tanto para aplicaciones cuyo código haya sido paralelizado con anterioridad o no, es decir, será una plataforma de supercómputo tanto para aplicaciones paralelizadas como secuenciales.

4. Actualmente openMosix solo funciona para los núcleos x86, sean SMP o no. El porte hacia arquitecturas Alpha y Sun se está estudiando, aunque existen todavía vías de desarrollo consideradas más prioritarias.

5. La programación óptima para un sistema openMosix es simplemente el uso de *fork*. Por esta razón han bautizado la programación como *fork-and-forget*, lo que da una perfecta idea de la extrema facilidad y rapidez de adaptación para aprovechar al máximo las prestaciones que openMosix puede ofrecer.

2.3. Ventajas, desventajas y limitaciones

No haría falta decir, aunque así se escribe para conocimiento del lector novel en esta literatura, que una ventaja puede no serlo para todo el mundo. Lo mismo para los contras. Las limitaciones suelen ser puntos más objetivos.

Aquí se intenta apuntar, a modo de resumen de todas las características ya vistas, lo que openMosix supone e implica. Para poder decidir en base a él deberían conocerse de antemano las propias posibilidades de linux. Así se sigue:

2.3.1. Ventajas

- no se necesitan librerías extra
- no se necesita reprogramar las aplicaciones
- sistema de ficheros oMFS
- el demonio de autodescubrimiento de nodos `omdiscd`

2.3.2. Desventajas

- código para la migración de procesos que usan memoria compartida todavía en beta
- núcleo-dependiente
- *pthreads* no ganan rendimiento, pues no migran los procesos

2.3.3. Limitaciones

- *pthreads*
- aplicaciones que se sirven de memoria compartida
- sockets migrables aún no implementados

2.4. Add-ons a openMosix

El proyecto openMosix es un proyecto abierto, como su propio nombre indica, y abierto se refiere también a agradecer cualquier contribución que cualquiera quiera hacer, a diferencia del proyecto MOSIX. Los desarrolladores del parche para el núcleo basan la práctica totalidad de su esfuerzo en el estudio de las complejas políticas que van dotando cada vez de mayores y mejores prestaciones a los subsistemas de openMosix, ya sea en la migración, el balanceo de carga, la memoria compartida, *etc.*–.

Y openMosix no es solo el parche para Linux, también ofrece al usuario toda una *suite* de aplicaciones para el manejo y administración de las políticas que ejecuta. Son las **userland tools**. Concretamente han sido desarrolladas por David Santo Orcero[1], uno de los desarrolladores más dotados e implicados internacionalmente con el *free software* de nuestro país.

Matthias Rechenburg se ha encargado de generar una GUI –con las librerías QT– para dichas herramientas, son las **openMosixview tools**.

El lector que quiera consultar el archivo de las *mail-list* del proyecto openMosix se dará cuenta que de entre las filas de usuarios que utilzan este tipo de clusters van saliendo regularmente propuestas de contribuciones, que son estudiadas y muy a menudo llevadas a cabo sin mayores problemas ni retrasos que los que el propio autor disponga. De entre este tipo de aportaciones cabría destacar el manual. El *howto* escrito por Kris Buytaert era insuficiente en muchos aspectos –noviembre de 2002– y el mismo autor del presente artículo propuso encargarse de la coordinación de la documentación, además sería en castellano.

Tras una vuelta por la sección *add-ons* del web del proyecto estaremos al día de las contribuciones existentes hasta el momento.

2.5. openMosix a fondo

Seguidamente se darán unas pinceladas sobre el funcionamiento interno de openMosix, que se divide en cuatro subsistemas:

1. oMFS
2. PPM
3. DFSA
4. Memory ushering.

openMosix File System (oMFS)

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

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

[1] Ponente plenario del jueves 25 en esta edición del VI Congreso Hispalinux.

Migración preventiva de procesos (PPM)

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 estas 2 partes.

Direct File System Access (DFSA)

openMosix proporciona el oMFS con la opción DFSA, que permite acceso a todos los sistemas de ficheros, tanto locales como remotos. Esto evita básicamente que las operaciones de e/s se realicen por la parte de sistema, es decir, que se tengan que ejecutar en el UHN y por tanto evite migrar a los procesos que requieran acceso a disco duro.

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.

Capítulo 3

Puesta a punto

It is a capital mistake to theorize
before one has data.

Sir Arthur Conan Doyle

La ponencia contará finalmente con la exposición de un cluster openMosix simple, de dos nodos. Se hablará del maquinario utilizado y de la instalación y ejecución del software openMosix.

3.1. Disposición del hardware

El sistema que se expondrá estará constituido por dos nodos: `mCii` y `metrakilate`.

`mCii` será una computadora completa en el sentido de disponer de todo lo necesario para funcionar de forma autónoma. `metrakilate` será un nodo *diskless* –DC– capaz de realizar su arranque por red, a partir de la información contenida en `mCii`.

3.2. Arrancando el sistema

Se verá paso a paso el arranque de un sistema Linux con el parche openMosix, para cada nodo. Se verán los servicios utilizados en el servidor `mCii` para esta disposición, comentando todos los ficheros de configración.

3.3. Utilizando el sistema

Se dará un repaso rápido por las herramientas de usuario que permiten administrar y corregir la propia algorísmica de openMosix, para adaptarla a cada necesidad. Asimismo se verá openMosixview, para realizar las mismas tareas en modo gráfico.

Se comentará también el estado actual del proyecto openMosix, y se ubicará dentro del mundo de desarrollo del programario bajo licencia GNU.

Para mayor información puede consultarse la página del proyecto de docuementación en castellano en `http://como.akamc2.net`.

Bibliografía

[SCP] Kai Hwang y Zhiwei Xu, *Scalable Parallel computing,* Ed. McGraw-Hill.

[SO] William Stallings, *Sistemas Operativos.*

[SOCD] Milan Milenković *Sistemas Operativos. Conceptos y Diseño*.

[OSC] Silberschatz Galvin, *Operating System Concepts.*

[PSIC] Miltos D. Grammatikakis, D. Frank Hsu & Miro Kraetzl, *Parallel System Interconnections and Communications.*

[DS] George Coulouris, Jean Dollimore, Tim Kindberg, *Distributed Systems, Concepts and Design.*

[SO] Peterson Silberschatz, *Sistemas operativos.*

[PE] M. A. Rodríguez i Roselló, *Programación Ensamblador para 8088-8086/8087*.

[CN] larry L. Peterson & Bruce S. Davie, .*Computer networks. A Systems Aproach.*

[CU] Jean-Marie Rifflet, *Comunicaciones en UNIX*.

[K24] Fernando Sánchez, Rocío Arango, *El Kernel 2.4 de Linux.*

[LI] John Lombardo, *Linux incrustado.*

[ARPCL] Jason Fink & Matther Sherer, *Ajustes dek rebdunuebti y Planificación de ka capacidad con Linux.*

[TL] Todo Linux Magazine, *vol. 21-28*

[LM] The Linux Magazine, *vol.1-2.*