UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA
F. E. S. ARAGÓN

IMPLEMENTACIÓN DE UN CLUSTER OPENMOSIX PARA CÓMPUTO CIENTÍFICO EN EL INSTITUTO DE INGENIERÍA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N :

JOSÉ CASTILLO CASTILLO

RICARDO BENJAMÍN OLVERA ACOSTA

DIRECTOR DE TESIS:

ING. JUAN JOSÉ CARREÓN GRANADOS

MÉXICO 2006

Agradecimientos

José Castillo Castillo.

Gracias a **Dios**.

A mis padres, Bertha Castillo Millán y Vidal Castillo Serrano, que siempre me han dado su apoyo incondicional y a quienes debo este triunfo profesional, por todo su trabajo y dedicación para darme una formación académica y sobre todo humanista y espiritual. De ellos es este triunfo y para ellos es todo mi agradeciemiento.

Para mis hermanos, Saúl y Susana, para que también continuen superandose. A toda mi familia, muy en especial a mi tia Felipa por todo su apoyo.

A la UNAM, la Máxima Casa de Estudios, y al Instituto de Ingeniería que con el apoyo de una beca me permitieron desarrollar este trabajo de tesis y adquirir mucha experiencia profesional. Al Doctor Gustavo Ayala Milián por el apoyo en espacios y equipo para el desarrollo de la investigación.

A todos mis amigos, amigas y todas aquellas personas que han sido importantes para mi durante todo este tiempo. A todos mis maestros que aportaron a mi formación. Para quienes me enseñaron más que el saber científico, a quienes me enseñaron a ser lo que no se aprende en salón de clase y a compartir el conocimiento con los demás.

A mi amigo y compañero de tesis Benjamín, que me enseño a salir adelante para la culminación del trabajo y a nuestro director de tesis, Ingeniero Juan José Carreón Granados por su confianza y apoyo en nuestra investigación.

A todos . . .

G R A C I A S

Mayo de 2006

Ricardo Benjamín Olvera Acosta.

Este trabajo está dedicado a toda mi familia, por su cariño y porque siempre me han apoyado. Gracias principalmente a mis padres Columba Acosta y Benjamín Olvera: quiero que sepan que para mí son los mejores, gracias por la ayuda que de ustedes y de sus parejas he recibido; éste trabajo es solo una pequeña muestra de lo mucho que les agradezco por la educación que me han brindado, esperando que mis hermanos aprendan algo de él, que les sirva como ejemplo para seguir adelante con sus proyectos de vida. Aprovecho para decirles que cuando de verdad quieran algo, luchen por ello; que cuando las cosas son difíciles se disfrutan mucho más, y que sepan que a nuestros padres no les podemos regalar algo mejor que ver a un hijo feliz por tener lo que se merece gracias a su esfuerzo: Huguín, Nancy, Uriel e Itzel, los quiero mucho.

Gracias a dios que, aunque ya no es el mismo que me enseñaron en el catecismo, me cuida siempre, me escucha las veces que lo necesito y, sobre todo, me da la fuerza para seguir adelante.

Gracias a todos y cada uno de los profesores de la FES Aragón, antes ENEP, por entregar parte de su vida para nuestro desarrollo. Gracias a la UNAM por recibirnos como a sus hijos y vernos crecer como personas y como profesionistas; gracias al Ingeniero Juan José Correón Granados por brindarnos su amistad y confianza para desarrollar este trabajo de tesis y, sobre todo, por ser la gran persona que es.

Y como dicen que detrás de un no tan gran hombre hay una gran mujer, gracias a Sara, ¡mi hermosa!, por estos dos años y todo lo que me has enseñado en ellos: tuve mucha suerte por haberte encontrado. Gracias también a tu familia: a María Luisa Pérez, al Sr. Sergio del Real y a Carlita por su amistad y confianza.

Gracias a todos y cada uno de mis amigos del Instituto de Ingeniería: becarios, trabajadores y secretarias, tanto de Computo como de Sismología y Mecánica Aplicada, y a algunos que en el paso de los años he tenido la oportunidad de conocer; al Dr. Francisco Chávez, al Dr. Jorge Aguirre, al M. en C. Miguel Rodríguez y al M. en I. Javier Lermo: gracias a todos por depositar su confianza en mi, por el buen ejemplo y porque, tal vez sin que lo sepan, me han inspirado para ser cada día mejor: a todos gracias por su amistad.

Gracias a las comunidades en resistencia por la dignidad que han mostrado; gracias a la comunidad del software libre que, aunque está muy lejos los unos de los otros, sus ideologías son muy parecidas: gracias por sus enseñanzas y por darme la esperanza de un mundo mejor. ¡Y que viva la música campesina del mundo!

Y cómo olvidar a mi compañero y amigo José Castillo Castillo por el tiempo dedicado a este trabajo y por todo lo que me ha enseñado en este proyecto de tesis: ojalá sea el primero de muchos triunfos.

Gracias a todos y cada uno de los que lean y han leído este trabajo porque, por ese simple hecho, ya forman parte de él.

Índice

Objetivos

- Proponer una nueva solución a la problemática de ejecución de programas de cómputo científico que requieren de alta capacidad de procesamiento en el Instituto de Ingeniería.

- Evaluar el uso de clusters como alternativa viable a la problemática de ejecución de programas de cómputo científico que requieren de sistemas de alta capacidad para procesamiento de datos dentro del Instituto de Ingeniería.

- Abrir nuevas alternativas a la adquisición de sistemas de cómputo de alto rendimiento como son los clusters, que puedan proporcionar resultados similares a menor costo y escalables, comparables con las supercomputadoras.

- Encaminar a los investigadores del Instituto de Ingeniería al uso de nuevas tecnologías para el procesamiento de grandes volúmenes de datos como también se hace en otros centros de investigación.

Prefacio

En temas relacionados con la ingeniería, hoy en día es común procesar grandes cantidades de información, y es necesario emplear principalmente sistemas de cómputo rápidos y con buenos recursos tales como procesador y memoria. En el mercado existen sistemas que pueden ofrecer estos requerimientos; tal es el caso de las computadoras multiprocesador o las llamadas supercomputadoras, pero este tipo de tecnología es muy costosa, tanto en el precio como en el mantenimiento de los equipos.

Ante esta situación, se buscan soluciones que puedan proporcionar resultados similares, a menor costo y escalables. Una de estas soluciones son los clusters.

¿Por qué darse a la tarea de diseñar y construir clusters, cuando hay supercomputadoras comerciales, perfectamente buenas y disponibles en el mercado?. La respuesta rápida es el dinero.

Cuando se construye un cluster con hardware disponible en los centros de cómputo, los costos son menores comparados con el de una supercomputadora comercial, obteniendo resultados satisfactorios tomando en cuenta el costo/beneficio.

Los primeros sistemas cluster fueron explorados por Don Becker y sus colegas en la NASA. Debido a que las restricciones presupuestarias les imposibilitaban el acceso a una supercomputadora comercial, ellos necesitaban analizar un conjunto complejo y muy grande de datos, que las misiones de la NASA tienden a generar. Encontraron la forma de conseguir el rendimiento de cómputo que necesitaban. Nombraron a su creación “Beowulf“ en honor al héroe mítico inglés Beowulf[1].

Los clusters asombrosamente han llegado a tener un gran alcance. Actualmente existe un

[1] http://www.beowulf.org/overview/faq.html [Última consulta: febrero 2006]

listado actualizado semestralmente por la universidad de Manhiem en Alemania que describe las mejores 500 supercomputadoras en el mundo[2], entre éstas figuran algunos clusters. Hasta 1997, casi todos los sistemas enumerados eran los sistemas comerciales de supercomputadoras de fabricantes bien conocidos tales como Cray, Silicon Graphics e IBM. A partir de 1998, algo extraordinario comenzó a aparecer en la lista, los clusters basados en Linux.

El supercómputo también ha venido a desempeñar un papel importante en diferentes áreas como la ingeniería, química, física, biología, cine, campo militar, entre otras y la tecnología de clusters ha llegado a ser cada vez más importante. Es por ello que dentro de las áreas de investigación como la ingeniería no se pueden obviar estas tecnologías que traen grandes beneficios para quienes saben aprovecharlas correctamente.

[2]Top 500, http://www.top500.org [Última consulta: febrero 2006]

Introducción

El desarrollo de este trabajo de tesis consiste en la implementación de un cluster en el Instituto de Ingeniería de la UNAM.

Se trabajó en dos coordinaciones: Mecánica Aplicada e Ingeniería Sismológica. En ambos centros se trabaja con grandes cantidades de información que requieren del tratamiento matemático mediante métodos numéricos que a su vez tienen que llevarse a cabo con auxilio de sistemas de cómputo. Conforme se avanza en las investigaciones de estos centros, es necesario de mejores y más potentes recursos de cómputo.

En la coordinación de Mecánica Aplicada, existe un grupo especial de mecánica numérica que dirige el Dr. Gustavo Ayala Milián. Este grupo trabaja en proyectos donde se hace empleo de métodos numéricos avanzados aplicados a la Ingeniería Civil que requieren de sistemas de cómputo de alto desempeño. El grupo ha trabajado con sistemas de cómputo tradicionales, que en la mayoría de los casos les ha sido suficiente para resolver los problemas computacionales con que se han enfrentado, y es por ello que con este trabajo se pretende iniciar al grupo en el uso de las tecnologías de clusters.

La coordinación de Ingeniería Sismológica que dirige el Dr. Jorge Aguirre González, opera y mantiene las redes acelerográficas de campo libre y en estructuras del Instituto, y se encarga de analizar los datos colectados de las estaciones sismográficas, lo cual requiere de sistemas de cómputo de alto desempeño para el análisis de estos datos de una forma rápida y eficiente, por lo que un cluster es de gran utilidad para mantener operando ininterrumpidamente la que hoy en día representa la red de monitoreo de temblores fuertes más importante del país.

Organización

La tesis esta compuesta de siete capítulos y a continuación se describe brevemente cada uno de los capítulos contenidos en este trabajo:

Capítulo 1. Antecedentes.

En este capítulo se hace una breve descripción del Instituto de Ingeniería y su misión, pues es donde se realizó la implementación del cluster. También se muestran los antecedentes del proyecto OpenMosix y los cambios que ha tenido en sus diferentes etapas.

Capítulo 2. Conceptos básicos de cómputo científico y clusters.

En este capítulo se describen conceptos fundamentales detrás del cómputo con clusters y se da una respuesta a la pregunta: ¿Por qué emplear Clusters?. También se cubren algunos conceptos básicos implicados en cómputo paralelo, optimización de programas, tecnologías de red elementales y de OpenMosix como un cluster de alto desempeño y balanceo de carga, mostrando sus características y capacidades. Esto permite trabajar con un vocabulario común en el resto del trabajo de tesis.

Capítulo 3. Análisis para el diseño del cluster OpenMosix.

En este capítulo se hace un estudio de los recursos en cuanto a requerimientos ideales básicos para la construcción de un cluster, como el que se plantea en este proyecto: características de los nodos, de la red de computadoras, requerimientos de hardware, requerimientos de software, características de la distribución Linux, de los compiladores y de las herramientas de administración.

Capítulo 4. Diseño del cluster OpenMosix.

En el capítulo tres se hace la descripción ideal para el cluster, ahora se plantea el diseño y organización del cluster de acuerdo al equipo de cómputo disponible para la construcción del mismo.

Capítulo 5. Implementación del cluster OpenMosix.

Se describe la instalación de los nodos y su configuración, la instalación de compiladores y la seguridad lógica del cluster.

Capitulo 6. Administración del cluster.

Se analizan y describen algunas de las herramientas para la administración del cluster y para el monitoreo de procesos distribuidos en los nodos de cluster.

Capitulo 7. Desempeño y pruebas de rendimiento.

Se muestran resultados de pruebas generales de rendimiento del cluster con ejecución de programas de propósito general, y en particular con programas de investigadores del Instituto de Ingeniería de las coordinaciones de Ingeniería Sismológica y Mecánica Aplicada.

Capítulo 1

Antecedentes

1.1. Instituto de Ingeniería

El Instituto de Ingeniería de la UNAM

El Instituto de Ingeniería es parte del Subsistema de Investigación Científica de la Universidad Nacional Autónoma de México y orgánicamente se encuentra dentro de la Coordinación de la Investigación Científica.

En la Universidad Nacional Autónoma de México, el Instituto de Ingeniería es el centro de investigación en diversas áreas de la ingeniería más productivo del país. Es una comunidad de aproximadamente 900 personas, entre ellos investigadores, estudiantes de ingeniería que realizan trabajos de tesis de licenciatura, maestría y doctorado, técnicos académicos, personal secretarial y de servicios.

Desde su fundación en 1956[3], la política del Instituto ha sido realizar investigación orientada a problemas generales de la ingeniería, así como colaborar con entidades públicas y privadas para mejorar la práctica de la ingeniería en el ámbito nacional, al aplicar los resultados de las investigaciones a problemas específicos.

En los programas actuales de trabajo se enfatiza el interés en las necesidades de la ingeniería nacional.

Las actividades que se llevan a cabo en el Instituto son: investigación técnica y aplicada,

[3]http://www.iingen.unam.mx [Última consulta: febrero 2006]

apoyo al desarrollo tecnológico y análisis de los requerimientos sociales a cuya solución puede aportar la ingeniería. Asimismo, se proporcionan servicios de ingeniería a los diversos sectores de la sociedad con el propósito de contribuir al avance de los objetivos propios de la Universidad.

Las principales funciones del Instituto son:

- Realizar investigación para mejorar los conocimientos, métodos y criterios en ingeniería, tanto fundamental como aplicada.
- La formación de investigadores en ingeniería.
- Apoyar el desarrollo tecnológico y análisis de los requerimientos sociales a cuya solución puede aportar la ingeniería.
- Proporcionar servicios de ingeniería a los diversos sectores de la sociedad con el propósito de contribuir al avance de los objetivos propios de la Universidad.
- Apoyar la formación de profesores y las tareas docentes de la Facultad de Ingeniería.
- Estudiar problemas de interés nacional.
- Difundir los resultados de sus investigaciones.
- Llevar a cabo las actividades necesarias para realizar las funciones precedentes.

En el desempeño de estas funciones, el Instituto colabora con otras instituciones afines, técnicas, culturales y científicas del país y en el extranjero.

Áreas de investigación

A continuación se mencionan las áreas de investigación que comprende el Instituto de ingeniería, como lo muestra el organigrama de la figura 1.1.

- Automatización
- Bioprocesos Ambientales
- Estructuras y Materiales
- Geotecnia
- Hidráulica

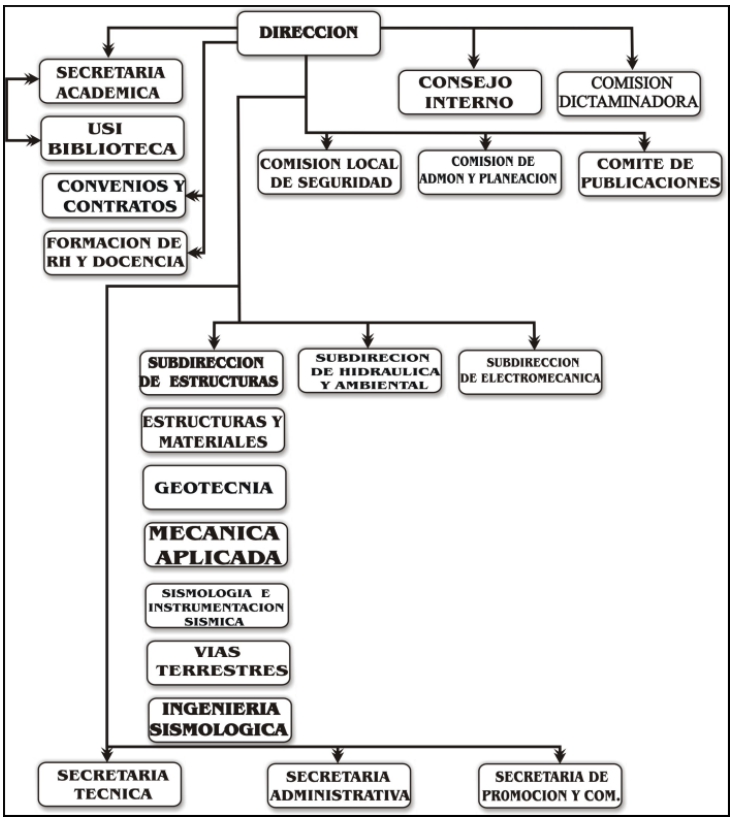

Figura 1.1. Organigrama IINGEN

- Ingeniería Ambiental
- Ingeniería de Procesos Industriales y Ambientales
- Ingeniería de Sistemas
- Ingeniería Mecánica Térmica y de Fluidos
- **Ingeniería Sismológica**
- Instrumentación
- **Mecánica Aplicada**
- Sismología e Instrumentación Sísmica
- Sistemas de Cómputo
- Vías Terrestres

1.2. Antecedentes del proyecto OpenMosix

OpenMosix es un proyecto de software libre para la implementación de un cluster de alto desempeño y balanceo de carga. Primero fue el proyecto Mosix, el cual en un inicio se liberó mediante una licencia de software libre, que permitía la libre distribución y modificación del código fuente de este proyecto; posteriormente los desarrolladores de Mosix deciden cambiar su licencia por una que impedía la modificación y adaptación del código fuente a las necesidades de quien usara este software. Debido a tal situación, varias personas que trabajaban en el proyecto decidieron separarse de él y continuar con el desarrollo bajo un esquema de software libre, el cual hace participar a una gran comunidad involucrada en el tema de clusters con Linux. El resultado es el proyecto OpenMosix.[4]

[4]OpenMosix, http://openmosix.sourceforge.net [Última consulta: febrero 2006]

Capítulo 2

Conceptos básicos de cómputo científico y clusters

En este capítulo se tratan algunos de los conceptos fundamentales detrás del cómputo con clusters, conceptos básicos implicados en cómputo científico, optimización de programas y algunos de tecnologías de red básicos. Esto permite al lector trabajar con un vocabulario común cuando se comience con el tema del diseño, instalación y configuración del cluster.

2.1. Cómputo científico

Cómputo científico: Se entiende por cómputo científico el desarrollo y traducción en términos de tecnología de cómputo para procesar modelos computacionales, cuya finalidad es resolver problemas complejos de ciencia e ingeniería [5].

El cómputo científico usa como parte principal la modelación matemática y/o computacional. Cuando se resuelve un problema complejo, mediante el uso de la computadora, también se le denomina experimentación numérica, la cual se considera otra rama para aprender y obtener información nueva, que se suma a las otras metodologías tradicionales: teoría y experimentación.

[5] http://www.enterate.unam.mx/Articulos/dos/noviembre/supercom.htm [Última consulta: febrero 2006]

2.2. Cluster

Un cluster es un arreglo de computadoras conectadas por una red para trabajar en cierto gran problema que pueda ser resuelto en pequeños pedazos [HDavid00].

Las características principales de un cluster [MCatalan04] son:

- *i*) Un cluster consta de 2 o más nodos.
- *ii*) Los nodos del cluster están conectados entre sí por al menos un canal de comunicación.
- *iii*) Los clusters necesitan software de control especializado.

Otro concepto importante es **clustering**, que se refiere a la técnica que permite combinar múltiples sistemas para trabajar en conjunto, y que se comporten como un recurso informático unificado. Implica proveer niveles de disponibilidad y escalabilidad de un sistema al menor costo.

2.2.1. Conceptos básicos

Alto rendimiento: Gran demanda de procesamiento de datos en procesadores, memoria y otros recursos de hardware, donde la comunicación entre ellos es rápida.

Balanceo de carga: Lo ideal en el procesamiento paralelo es que cada procesador realice la misma cantidad de trabajo, donde además se espera que los procesadores trabajen al mismo tiempo. La meta del balanceo de carga es minimizar el tiempo de espera de los procesadores en los puntos de sincronización.

Compilador: Un compilador es un programa que traduce otro programa escrito en un lenguaje de programación llamado código fuente, en un programa equivalente al lenguaje de computadora llamado ejecutable ó binario.

Computadora vectorial: Posee un conjunto de unidades funcionales utilizados para procesar vectores eficientemente. Contiene registros vectoriales para operar sobre ellos en un solo ciclo de reloj.

Computadora paralela: Máquina con dos o más procesadores que pueden trabajar simultánea y/o coordinadamente.

Estas son de dos tipos: las MIMD donde cada procesador puede ejecutar diferentes instrucciones sobre diferentes datos, y las SIMD donde los procesadores ejecutan las mismas instrucciones pero con diferentes datos, como se explicara en la siguiente sección.

Eficiencia: Es la relación entre el costo computacional y el funcionamiento del cluster; y lo que indica es qué tan eficiente se está utilizando el hardware y se expresa de la siguiente forma:

$e(n) = T(1)/T(n)n$; donde e es la eficiencia,n es el numero de procesadores, $T(1)$ es el tiempo en que tarda en procesar un programa en particular en un procesador, $T(n)$ es el tiempo en que tarda en procesar un programa en particular en n procesadores.

Escalabilidad: Generalmente se mide la eficiencia de un problema, utilizando un tamaño y un número de procesadores fijo, pero esto es insuficiente, pues los resultados serán diferentes cuando se aumente o disminuya el tamaño del problema y el número de procesadores. Esto es, existe un problema de escalabilidad.

Cuando se aumenta el número de procesadores para el mismo tamaño del problema, la sobrecarga debido al paralelismo (comunicaciones, desbalanceo de carga), aumenta y similarmente podemos tener casos en donde el tamaño del problema es muy pequeño para tener una evaluación real del problema sobre cierta máquina.

Flops: Un flop es utilizado para medir operaciones de punto flotante por segundo. Es una medida de la velocidad del procesamiento numérico del procesador. Se utilizan en unidades de millones de flops (MegaFlops), Miles de Millones de flops (GigaFlops), etc.

Kernel: El kernel, también conocido como núcleo; es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora.

Memoria compartida: En una máquina paralela existe una sola memoria que puede ser accedida por todos los procesadores.

Memoria distribuida: Cada uno de los procesadores de un multiprocesador tiene asociado a él una unidad de memoria.

Nodo: Se refiere a una computadora sola que contiene recursos específicos, tales como memoria, interfaces de red, uno o más CPU, etc.

Paralelismo: Consiste en el procesamiento de una serie de instrucciones de programa que son ejecutables por múltiples procesadores que trabajan de manera independiente [MCatalan04]
.

Existen dos formas conocidas de hacer paralelismo: una es en hardware y otra en software. Por hardware depende de la tecnología de cómputo y la de software se refiere a la habilidad del usuario para encontrar áreas bien definidas del problema que se desea resolver, de tal forma que éste pueda ser dividido en partes que serán distribuidas entre los nodos del cluster.

Proceso: Un proceso es básicamente un programa en ejecución. Cada proceso tiene asociado un espacio de direcciones, es decir una lista de posiciones de memoria desde algún mínimo hasta algún máximo que el proceso puede leer y escribir [ATanenbaum03].

Rendimiento: Es la efectividad del desempeño de una computadora sobre una aplicación o prueba de rendimiento (benchmark) en particular. En las mediciones de rendimiento están involucrados velocidad, costo y eficiencia.

Speedup(velocidad): Se define como el tiempo que tarda en ejecutarse el mismo programa en un solo procesador, dividido entre el tiempo que toma ejecutarse el mismo programa en N procesadores.

$s = T(1)/T(n)$.

Donde s es el speedup, $T(1)$ es el tiempo de ejecución en un procesador y $T(n)$ el tiempo de ejecución en n procesadores.

En un problema que es completamente paralelo, el valor del speedup debe ir incrementando linealmente con el valor de n, sin embargo en muchos problemas donde el balanceo de carga no es perfecto y la comunicación entre procesos sobrepasa el tiempo de cómputo, el valor del speedup es menor que el valor de $T(n)$. La mejor solución es la que se acerque más al valor de $T(n)$

2.2.2. Lenguajes de programación paralela

Existen varios lenguajes de programación paralela, sobresaliendo de estos MPI(Message Passing Interface) y PVM(Parallel Virtual Machine), por ser uno de los estándares más aceptados.

MPI

MPI consiste de una biblioteca estándar para programación paralela en el modelo de intercambio de mensajes. En este estándar se han incluido los aspectos más relevantes de otras bibliotecas de programación paralela.

Entre las ventajas de MPI se encuentra la disponibilidad de varios modos de comunicación, los cuales permiten al programador el uso de buffers para el envío rápido de mensajes cortos, la sincronización de procesos o el traslape de procesos de cómputo con procesos de comunicación. Esto último reduce el tiempo de ejecución de un programa paralelo, pero tiene la desventaja de que el programador debe ser más cuidadoso para evitar la corrupción de mensajes. Dos de las principales distribuciones libres de MPI son: **LAM/MPI y MPICH**.

PVM

PVM se comenzó a desarrollar en el verano de 1989 por el Oak Ridge National Laboratory, y posteriormente junto con la Universidad de Tennessee en los EUA. Es una biblioteca de envío de mensajes, totalmente libre, capaz de trabajar en redes homogéneas y heterogéneas, y que hace uso de los recursos existentes en algún centro de trabajo para poder construir una máquina paralela de bajo costo, obteniendo su mejor rendimiento en "horas muertas".

Maneja transparentemente el ruteo de todos los mensajes, conversión de datos y calendarización de tareas a través de una red de arquitecturas incompatibles. Está diseñado para conjuntar recursos de cómputo y proveer a los usuarios de una plataforma paralela para correr sus aplicaciones, independientemente del número de computadoras distintas que utilicen y donde éstas se encuentren localizadas. El modelo computacional de PVM es simple y además muy general. El usuario escribe su aplicación como una colección de tareas cooperativas. Las tareas acceden los recursos de PVM a través de una biblioteca de rutinas. Estas rutinas permiten la inicialización y terminación de tareas a través de la red, así como la comunicación y sincronización entre tareas.

Los constructores de comunicación incluyen aquellos para envío y recepción de estructuras

de datos así como primitivas de alto nivel, tales como emisión, barreras de sincronización y sumas globales.

2.3. Tipos y modelos de clusters

En esta sección se analizan los tipos y modelos de cluster basados en la taxonomía de Flynn.

2.3.1. Arquitecturas de computadoras

En 1966 Michael Flynn propuso un mecanismo de clasificación de las computadoras. La taxonomía de Flynn es la manera clásica de organizar las computadoras, y aunque no cubre todas las posibles arquitecturas, proporciona una importante visión para varias de éstas.

El método de Flynn se basa en el número de instrucciones y de la secuencia de datos que la computadora utiliza para procesar información. Puede haber secuencias de instrucciones sencillas o múltiples y secuencias de datos sencillas o múltiples. Dicha clasificación da lugar a cuatro tipos de computadoras (figura 2.1):

- SISD : Single Instruction Single Data
- SIMD : Single Instruction Multiple Data
- MIMD : Multiple Instruction Multiple Data
- MISD : Multiple Instruction Single Data

Los modelos **SIMD** y **MIMD** son los únicos modelos aplicables a las computadoras paralelas. También hay que mencionar que el modelo MISD es teórico.

SISD

SISD es el modelo tradicional de computadora secuencial donde una unidad de procesamiento recibe una sola secuencia de instrucciones que opera en una secuencia de datos, ejemplo: para procesar la suma de N números $a1, a2, ...an$, el procesador necesita acceder a memoria N veces consecutivas (para recibir un número). También son ejecutadas en secuencia N-1 sumas, es decir los algoritmos para las computadoras SISD no contienen ningún paralelismo, éstas están construidas de un solo procesador (figura 2.2).

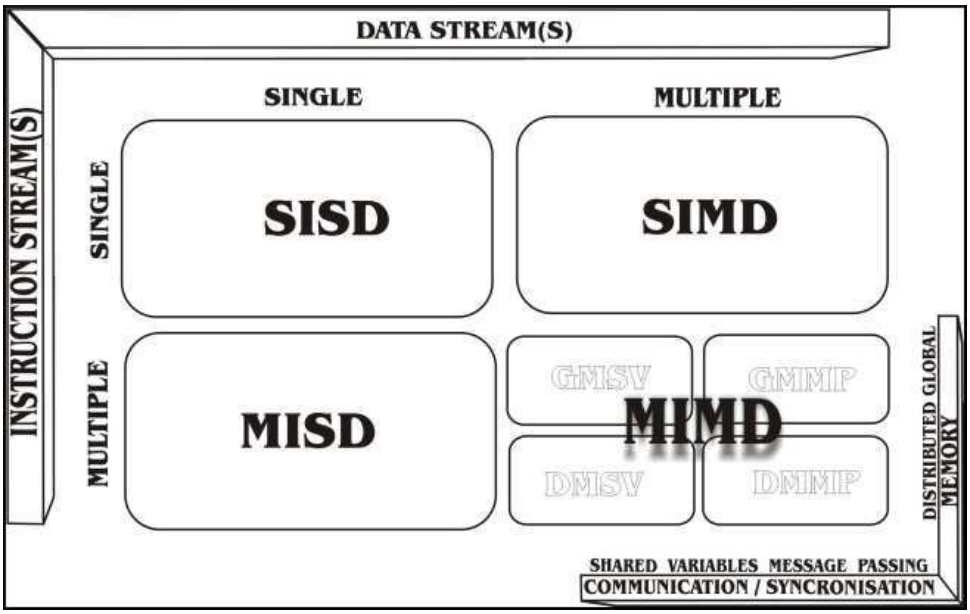

Figura 2.1. Taxonomía de Flynn.

Figura 2.2. Modelo SISD (Single Instruction Single Data).

SIMD

A diferencia de SISD, en el modelo **SIMD** se tienen múltiples procesadores que sincronizadamente ejecutan la misma secuencia de instrucciones, pero en diferentes datos (figura 2.3). El tipo de memoria que estos sistemas utilizan es distribuida, ejemplo: aquí hay N secuencias de datos, una por procesador, por lo que diferentes datos pueden ser utilizados en cada procesador. Los procesadores operan sincronizadamente y un reloj global se utiliza para asegurar esta operación, es decir, en cada paso todos los procesadores ejecutan la misma instrucción, cada uno con diferente dato, ejemplo: sumando dos matrices $A + B = C$, siendo A y B de orden 2 y teniendo 4 procesadores.

```
A11 + B11 = C11   ,   A12 + B12 = C12
A21 + B21 = C21   ,   A22 + B22 = C22
```

La misma instrucción se ejecuta en los 4 procesadores (sumando dos números) y los 4 ejecutan las instrucciones simultáneamente. Esto toma un paso en comparación con cuatro pasos en máquina secuencial.

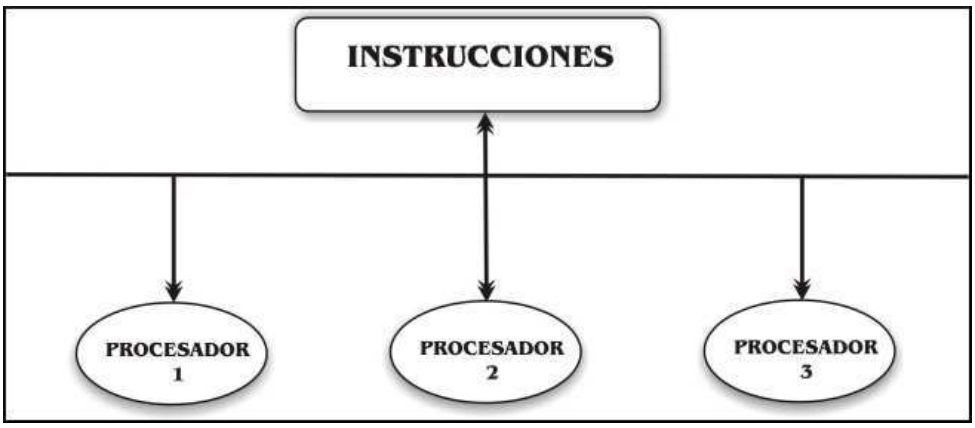

Figura 2.3. Modelo SIMD (Single Instruction Multiple Data).

Máquinas con arreglos de procesadores vectoriales como CRAY1 y CRAY2 son ejemplos de arquitecturas SIMD.

MIMD

El modelo **MIMD** es del tipo de computadoras paralelas al igual que las SIMD. La diferencia con estos sistemas es que MIMD es asíncrono: no tiene un reloj central (figura 2.4).

Cada procesador en un sistema MIMD puede ejecutar su propia secuencia de instrucciones y tener sus propios datos: esta característica es la más general y poderosa de esta clasificación.

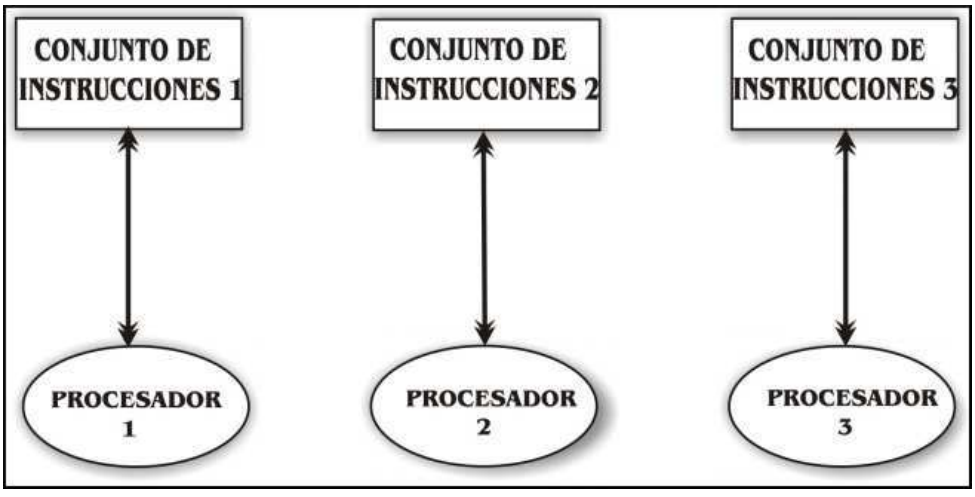

Figura 2.4. Modelo MIMD (Multiple Instruction Multiple Data).

Se tienen N procesadores, N secuencias de instrucciones y N secuencias de datos. Cada pro-

cesador opera bajo el control de una secuencia de instrucciones, ejecutada por su propia unidad de control, es decir cada procesador es capaz de ejecutar su propio programa con diferentes datos. Esto significa que los procesadores operan asincrónicamente, o en términos simples, pueden estar haciendo diferentes cosas en diferentes datos al mismo tiempo.

Los sistemas MIMD también tienen una subclasificación:

- **Sistemas de memoria compartida**:

  En este tipo de sistemas cada procesador tiene acceso a toda la memoria, es decir, hay un espacio de direccionamiento compartido (figura 2.5). Con esto se tiene tiempo de acceso a memoria uniformes, ya que todos los procesadores se encuentran igualmente comunicados con memoria principal, además el acceso a memoria es por medio de un solo canal. En esta configuración debe asegurarse que los procesadores no tengan acceso simultáneamente a regiones de memoria de una manera en la que pueda ocurrir algún error.

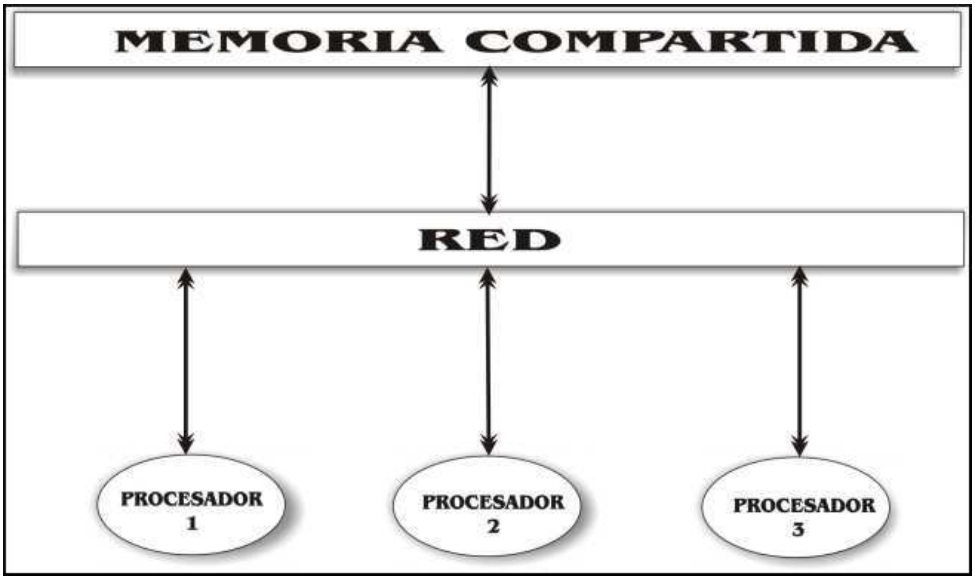

Figura 2.5. Sistemas de Memoria Compartida.

Ventajas:

- Facilidad de la programación. Es mucho más fácil programar en estos sistemas que en sistemas de memoria distribuida.
- Las computadoras MIMD con memoria compartida son sistemas conocidos como de multiprocesamiento simétrico (SMP), donde múltiples procesadores comparten un

mismo sistema operativo y memoria. Otro término con que se conoce es máquinas firmemente juntas o de multiprocesadores.

Desventajas:

- El acceso simultáneo a memoria es un problema.
- Poca escalabilidad de procesadores, debido a que se puede generar un cuello de botella en el número de CPU.
- En computadoras vectoriales como la Cray, todas las CPU tienen un camino libre a la memoria. No hay diferencia entre las CPU.
- La razón principal por el alto precio de Cray es la memoria.

Algunos ejemplos de estos sistemas son: **SGI/Cray Power Challenge, SGI/Cray C90, SGI/Onyx, ENCORE, MULTIMAX, SEQUENT** y **BALANCE**, entre otras.

- **Sistemas de memoria distribuida**:

  Estos sistemas tienen su propia memoria local. Los procesadores pueden compartir información solamente enviando mensajes, es decir, si un procesador requiere los datos contenidos en la memoria de otro procesador, deberá enviar un mensaje solicitándolos (figura 2.6). A esta comunicación se le conoce como **paso de mensajes**.

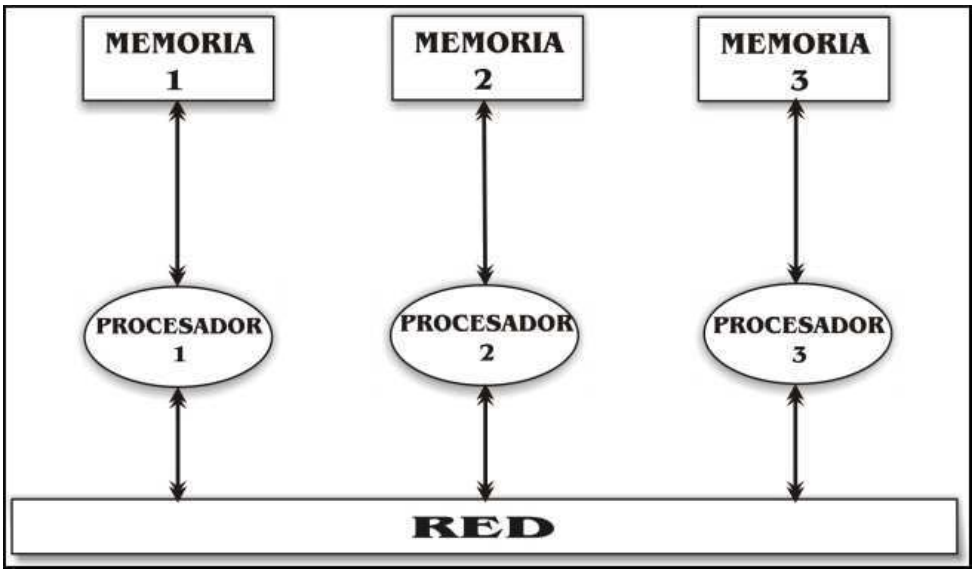

Figura 2.6. Sistemas de Memoria Distribuida.

Ventajas:

- La **escalabilidad**. Las computadoras con sistemas de memoria distribuida son fáciles de escalar, debido a la demanda de los recursos se puede agregar más memoria y procesadores.

Desventajas:

- El acceso remoto a memoria es lento.
- La programación puede ser complicada.

Las computadoras MIMD de memoria distribuida son conocidas como **sistemas de procesamiento en paralelo (MPP)**, donde múltiples procesadores trabajan en diferentes partes de un programa, usando su propio sistema y memoria. También se les llama multicomputadoras, máquinas libremente juntas o **cluster**.

MISD

El modelo **MISD** no tiene una aplicación real, pero se hace la descripción teórica. En este modelo, secuencias de instrucciones pasan a través de múltiples procesadores. Diferentes operaciones son realizadas en diversos procesadores, es decir, N procesadores, cada uno con su propia unidad de control, comparten una memoria común (figura 2.7).

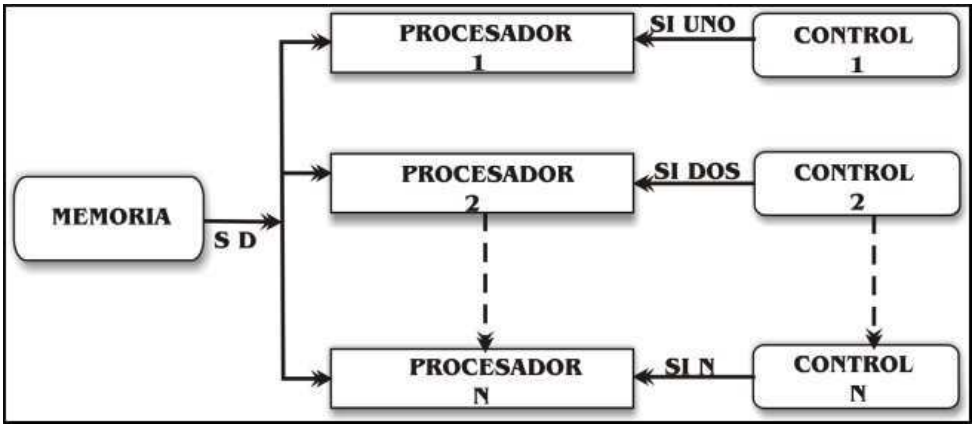

Figura 2.7. Modelo MISD (Multiple Instruction Single Data).

Aquí hay N secuencias de instrucciones y una secuencia de datos. El paralelismo es alcanzado dejando que los procesadores realicen cosas al mismo tiempo en el mismo dato.

2.3.2. Tipos de clusters

Los clusters se pueden clasificar tomando en cuenta diferentes aspectos como la aplicación, disponibilidad, servicio, hardware, sistema operativo, configuración y el número de nodos.

Para cuestiones prácticas, en este trabajo se tomó en cuenta la siguiente clasificación con respecto al servicio que proveen los cluster:

- **Tolerante a fallas (Fail-Over)**: Utiliza una conexión de alto desempeño entre las computadoras; ésta conexión es utilizada para monitorear cuál(es) de los servicios están en uso, así como la sustitución de una máquina por otra cuando uno de sus servicios ha caído.
- **Balanceo de carga (Load-Balancing)**: Cuando los recursos del nodo son insuficientes para el procesamiento de los datos, el cluster distribuye entre los demás nodos las tareas para un mejor desempeño.
- **Alto Desempeño (High Performance Computing)**: Estas máquinas han estado configuradas especialmente para centros de datos que requieren una potencia de computación extrema.

2.3.3. Modelos de clusters

- **NUMA (Non-Uniform Memory Access)**: Tiene acceso compartido a la memoria donde se puede ejecutar el código del programa.
- **MPI (Message Passing Interface)**: Es el estándar abierto de librerías de paso de mensajes. MPICH y LAM (Local Area Multicomputer) son dos implementaciones de MPI de código abierto.
- **PVM (Parallel Virtual Machine)**: Al igual que MPI, también es usado en clusters Beowulf y se basa en paso de mensajes, aunque ya está cayendo en desuso, pues carece de varias características y no cumple con los nuevos estándares para desarrollo de programas paralelos.
- **Beowulf**: Es un cluster de componentes commodity[6] dedicados a la solución de un problema paralelo. El primer cluster de este tipo fue desarrollado por Thomas Sterling, de

[6]Hardware disponible en los centros de cómputo que no necesariamente necesita adquirirse específicamente para el propósito de la construcción de un cluster.

la división de Ciencias de la Tierra de la NASA en JPL California. Esta solución popular ha sido ampliamente aceptada en varios ambientes de producción, principalmente laboratorios de investigación y sitios académicos.

En una configuración tradicional de un cluster Beowulf, los nodos se conectan por medio de una red privada, y sólo el nodo maestro es visible desde el exterior. El nodo maestro está reservado para acceder, compilar y manejar las aplicaciones a ejecutar.

2.4. OpenMosix como un cluster de alto desempeño y balanceo de carga

En la sección 2.3 se ha hablado de los tipos y modelos de clusters. Aquí describiremos las características de OpenMosix para entender su funcionamiento y la utilidad de éste.

OpenMosix es un parche (patch) al kernel de Linux que proporciona compatibilidad con el estándar de Linux versión 2.4.x para plataformas IA32[7] [MCatalan04], el algoritmo interno de balanceo de carga migra los procesos entre los nodos del cluster de forma transparente para el usuario. La principal ventaja es una mejor compartición de los recursos entre nodos, así como un mejor aprovechamiento de los mismos.

El cluster OpenMosix asigna por sí mismo la utilización óptima de los recursos que son necesarios en cada momento y de forma automática, y a esto nos referimos cuando decimos que el balanceo de carga es transparente al usuario, pues éste no necesita preocuparse por hacer uso de algunas tareas o herramientas extra para poder trabajar en este sistema.

Los algoritmos de OpenMosix son dinámicos, lo que contrasta y es una fuerte ventaja frente a los algoritmos estáticos de PVM/MPI. Responde a las variaciones en el uso de los recursos entre los nodos, migrando procesos de un nodo a otro, con y de forma transparente para el proceso para balanceo de carga y para evitar falta de memoria en un nodo, no necesita una adaptación de la aplicación, ni siquiera que el usuario sepa algo sobre el cluster.

[7]IA32 es la arquitectura de microprocesadores de 32 bits de Intel (Intel Architecture 32). Son los microprocesadores más usados en los ordenadores personales (PC). En febrero de 2006 ya existe una nueva versión de OpenMosix para kernels 2.6.x, aún en fase beta; en esta nueva versión ya se contemplan arquitecturas de 64 bits.

OpenMosix puede balancear una única aplicación, si esta está dividida en procesos. Esto ocurre en gran número de aplicaciones hoy en día: lo que balancea son procesos. Cuando un nodo está muy cargado por sus procesos y otro no, se migran procesos del primer nodo al segundo.

La característica de migración de procesos transparente hace que el cluster funcione de forma "similar" a un sistema SMP(Symmetric Multi Processing), donde las tareas originadas en un nodo del cluster pueden distribuirse en los diferentes "procesadores" de este sistema.

El proyecto OpenMosix está respaldado y siendo desarrollado por personas muy competentes y respetadas en el mundo del Open Source, trabajando juntas en todo el mundo, tratando de crear un estándar en el entorno de clustering para todo tipo de aplicaciones HPC (High Performance Computing).

2.4.1. Características de OpenMosix

Pros de OpenMosix:

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

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 (hilos) no ganan demasiada eficiencia. Tampoco se obtiene mucha mejora cuando se ejecuta un solo proceso, como por ejemplo un navegador web.

2.4.2. Subsistemas de OpenMosix

Actualmente podemos dividir los parches de OpenMosix en cuatro grandes subsistemas.

- **Mosix File System (MFS)**:

  En cuanto a líneas de código el primer subsistema es MFS que permite un acceso a sistemas de archivos remotos, por ejemplo, como si estuviese localmente montado.

  El sistema de archivos del nodo raíz y de los demás podrán ser montados en el directorio $/mfs$ y se podrán acceder por ejemplo: para acceder al $/home$ del nodo 2 dentro del directorio $/mfs/2/home$ desde cualquier nodo del cluster.

- **Migración de procesos**:

  Con OpenMosix se puede lanzar un proceso en una computadora y ver si se ejecuta en ésta, en otra, o en el seno del cluster (nodo maestro).

  Cada proceso tiene su único nodo raíz (UHN, Unique Home Node) que 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. Mientras tanto OpenMosix se encarga de establecer la comunicación entre estos dos procesos.

- **Direct File System Access(DFSA)**:

  OpenMosix proporciona MFS con la opción DFSA, que permite acceso a todos los sistemas de archivos tanto locales como remotos.

- **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 separan 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 memoria ushering es un subsistema independiente del subsistema de equilibrado de carga, y por ello se le considera por separado.

2.4.3. El algoritmo de migración

El proceso de migración basado en el modelo $fork-and-forget$[8], en el que puede migrarse cualquier proceso a cualquier nodo del cluster de forma completamente transparente. La migración también puede ser automática: el algoritmo que lo implementa tiene una complejidad del orden de $O(n)$, siendo n el número de nodos del cluster [MCatalan04].

La ventaja del modelo es que la distribución de tareas en el cluster está determinada por OpenMosix de forma dinámica, conforme se van creando tareas. Así, cuando un nodo está demasiado cargado y las tareas que se están ejecutando puedan migrar a cualquier otro nodo del cluster. Es así como desde que una tarea se ejecuta hasta que ésta termina, podrá migrar de un nodo a otro, sin que el proceso sufra mayores cambios.

El nodo raíz:

Cada proceso ejecutado en el cluster tiene asignado un único nodo raíz, en el cual se lanza originalmente el proceso y donde éste empieza a ejecutarse.

Desde el punto de vista del espacio de procesos de la maquina del cluster, cada proceso con su correspondiente Identificador de Proceso (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 si no que el proceso migrado se ejecutara en este como una *thread* (hebra o hilo) del kernel. La interacción con un proceso, por ejemplo enviarle señales a cualquier proceso migrado, se puede realizar exclusivamente del nodo raíz.

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 el administrador de cualquiera de los dos sistemas.

2.4.4. El mecanismo de migrado

La migración de procesos en OpenMosix es completamente transparente. Esto significa que al proceso migrado no se le avisa que ya no se ejecuta en su nodo de origen, y si tuviera que

[8]fork-and-forget hace referencia a que el sistema cuando reconoce un subproceso, se encarga de ejecutarlo en otro nodo en paralelo, sin ningún efecto ni notificación.

leer o escribir algo, lo haría en el nodo origen, hecho que supone leer o grabar remotamente en este nodo.

2.4.5. ¿Cúando podrá migrar un proceso?

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

- El proceso no puede ejecutarse en modo de emulación VM86 (Modo Virtual 8086), donde se requieren ejecutar instrucciones antiguas para procesadores de arquitectura 8086.
- 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.
- Y una condición muy importante: el proceso no puede usar segmentos de memoria compartida.

Cumpliendo todas estas condiciones el proceso puede migrar y ejecutarse migrado. Como se puede sospechar OpenMosix no sabe nada de esto y en un principio migra todos los procesos que puedan hacerlo si por el momento cumple todas las condiciones, y en caso de que algún proceso deje de cumplirlas, lo devuelve al nodo raíz para que se ejecute en él mientras no pueda migrar de nuevo.

2.4.6. Comunicación entre las dos áreas

Un aspecto interesante es 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 donde la tarea migrada se ejecuta, los accesos al kernel se hacen de forma local, es decir, que se atiende en el nodo donde 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 se comunican 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 donde 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 necesitara procesar.

El nodo raíz procesará la llamada y enviara de vuelta al nodo dónde se esta ejecutando realmente el proceso migrado cualquiera de estos datos:

- El valor del éxito o fracaso de la llamada.
- Aquello que necesite saber para actualizar sus segmentos de datos.
- El estado en el que estará si el proceso se ejecuta en el nodo raíz.

Esta comunicación también puede ser generada por el nodo raíz. Es el caso, por ejemplo, del envió 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 él 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 corresponderá a la señal asíncrona.

Finalmente, una vez realizada toda la operación necesaria del 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 si 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 resuelve como en el primer caso comentado, es decir, como una llamada al kernel.

Este mecanismo de comunicación entre áreas es el que nos aseguran 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 mas 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.

Capítulo 3

Análisis para el diseño del cluster OpenMosix

Como se habló en la sección 2.4, la distribución de procesos en los nodos del cluster es la principal característica de funcionalidad de OpenMosix. De una forma ideal, los procesos buscarán ejecutarse en un ambiente igual o muy similar del nodo donde provienen, tal vez con mejores recursos de memoria y procesador. Hablando en términos del programa, éste estará ejecutándose en nodos externos, por lo que es indispensable que el procesador externo tenga el conjunto de instrucciones necesarias para poder procesar el código compilado en el nodo anfitrión, de otra forma el programa no podría ejecutarse.

Con base en estas observaciones, se concluye que la construcción de un cluster OpenMosix como en otros tipos de clusters es muy recomendable que se haga con hardware homogéneo, aunque muchas de las veces debido a los recursos con los que se pueden disponer en nuestro centros de investigación no es posible cumplir con este requerimiento, no indispensable pero si recomendado, pues además de proveer un ambiente de ejecución adecuado para los procesos, facilita la administración del software instalado en los nodos del cluster.

3.1. Requerimientos de Hardware

La elección del hardware a utilizar en un cluster puede definirse primeramente en base a conocer la magnitud del problema o problemas que se quieren resolver, el hardware que puede adquirirse en el mercado, los recursos económicos y humanos con los que se cuentan, tanto para administrar el cluster cómo para programar y ejecutar las aplicaciones. Cuando se planea la adquisición, se propone la compra del "mejor" equipo de cómputo.

La descripción de estos nodos es una propuesta ideal para ensamblar el cluster, en ésta también se describe el tipo de red o canal de comunicación que se recomienda, aunque por supuesto es hardware que cumple con los requerimientos básicos. Cabe aclarar que no se propone la construcción de un cluster especifico, pues no se ensamblará éste para un propósito, problema o proyecto en particular, si no por el contrario, el cluster que se ha propuesto es para la ejecución de programas de propósito general que ya existen o se usan dentro del Instituto de Ingeniería, o de aquellos que pueden ser adaptados con cierta facilidad a este tipo de ambientes. En el capítulo 7, en las pruebas de rendimiento se mostrará la utilidad de OpenMosix y el tipo de programas que pueden aprovechar ésta tecnología y cuales no.

En las coordinaciones donde se desarrolló este trabajo se ejecutan programas con diferentes tipos de requerimientos, por tanto se propondrá el diseño de un cluster que pueda en el mayor de los casos responder a las necesidades generales.

3.1.1. Características de los nodos

Se identifican como nodos aquellas unidades individuales de procesamiento en el cluster. Para la implementación de un cluster OpenMosix, el hardware soportado es para las arquitecturas IA32 y compatibles, es decir, en términos del hardware disponible en el mercado estamos hablando de los procesadores Intel y AMD con tecnología para procesamiento de palabras de 32 bits. OpenMosix ha sido desarrollado y probado para funcionar en estos dos tipos de procesadores y como se ha mencionado ya, es recomendable no ensamblar clusters con ambos tipos de procesadores.

Los factores en la elección de uno u otro tipo de procesador son principalmente su costo y también por supuesto el software que se podrá aprovechar al máximo. Como ejemplo, se puede mencionar que por cuestiones de mercado, los procesadores AMD han sido más económicos sin querer decir con esto que tengan un menor rendimiento o tecnologías más obsoleta o atrasada, pero en cuanto al desarrollo de aplicaciones de software comerciales y no comerciales, como por ejemplos los compiladores, estas han sido desarrolladas y optimizadas para los procesadores Intel. Esto no quiere decir que sea imposible de implementar un cluster de bajo costo con hardware de bajo costo, simplemente se requiere en la mayoría de los casos un estudio de las herramientas disponibles para optimizar las aplicaciones de los usuarios.

Con base en estas observaciones se hace una propuesta de equipo de cómputo *ideal*, pensando en que la adquisición de este equipo es exclusivamente para el uso de un cluster dedicado.

- Procesador:

  Intel(R) Pentium (R) 4.
  Velocidad del procesador (Processor Speed) 3.6 GHz.
  Cache Nivel 2 (L2) de 2 MB.

- Memoria (RAM):

  2 GB DDR-2 667 MHz.

- Disco Duro (Hard Disk):

  120 GB, SCSI de 15,000 rpm.

- Tarjeta Madre (Motherboard)

  Intel Corporation.
  Velocidad del Bus (System Bus Speed) 800 MHz.
  Velocidad de la memoria (System Memory Speed) 667 MHz.

- Tarjeta de red

  Intel Gigabit Ethernet.

Se ha listado el hardware básico de forma ilustrativa, pues no se mencionan detalles como dispositivos periféricos (teclado, mouse, etc.) y tarjetas adicionales (tarjeta SCSI para los discos duros, etc.), éstas quedan a criterio de las necesidades reales que se les puedan dar a estos dispositivos en el cluster.

3.1.2. Características de la red de computadoras

Como se ha venido mencionando, el canal de comunicación es uno de los subsistemas más importantes en la construcción de un cluster, pues de este dependen todo tipo de comunicaciones que haya entre los nodos. Cuando no se tiene el conocimiento sobre la importancia de éste, el tener la última tecnología en cuanto a procesadores no ayuda en nada al cluster, pues la comunicación entre ellos echa por la borda el trabajo de cooperación que pudieran tener de una forma eficiente.

En el mercado, desde la aparición de las redes de computadoras han surgido diversos tipos de tecnologías. Las más destacadas por su bajo costo relativo y la facilidad de uso e implementación son las redes Ethernet. Aunque una red de este tipo no siempre satisface las necesidades de comunicación veloz de un cluster, por el costo y la reutilización de las redes ya instaladas se hace casi siempre uso de éstas.

En ésta propuesta, considerando que las redes tipo Ethernet se hacen cada vez más accesibles por la reducción de costos, se considera implementar Gigabit Ethernet. En el caso OpenMosix, es el tipo de red de computadoras más veloz con el que se conoce puede trabajar, pues aunque existen tecnologías de red mucho más veloces, el kernel y el tipo de comunicaciones con las que trabaja OpenMosix, no están programadas para aprovecharlas aún.

3.2. Requerimientos de Software

El fin que lleva a la construcción de un cluster es por supuesto la ejecución de algún programa, entonces es de suma importancia conocer que tipo de software que se debe y se puede instalar para que las tareas que se ejecuten lo hagan con el mejor desempeño posible.

3.2.1. Características de la distribución Linux

Una alternativa importante para atender soluciones a ciertas demandas de cómputo científico se deriva de componentes de software libre, como el sistema operativo Linux; el proyecto que originó todos los derivados actuales de clusters. Principalmente por que es de código abierto, su código fuente esta disponible para poder adaptarlo a distintos ambientes de hardware y software, entre otras características. Derivado de esto, varios grupos han desarrollado lo que se les conoce

como distribuciones de Linux. Como es conocido, existe una gran variedad de distribuciones, y muchas de estas pueden ser tomadas como base para poder implementar un cluster OpenMosix.

La elección de la distribución para implementar un cluster OpenMosix depende de factores básicos como lo es el software con el que cuenta la distribución, el soporte a actualizaciones para todo este tipo de software, las herramientas de administración del sistema, su facilidad de uso y aprendizaje entre otros.

Para el proyecto OpenMosix, de acuerdo con la documentación y foros de discusión de usuarios de ésta tecnología, las distribuciones más utilizadas son Red Hat, Debian, Suse y Gentoo. Todas ellas utilizadas por su gran reputación en el mercado, facilidad de uso y estabilidad.

3.2.2. Características de los compiladores

Dentro de las aplicaciones finales que serán ejecutadas en el cluster, tenemos dos casos a saber:

- Aquellas que por ser aplicaciones comerciales o de terceros no se dispone del código fuente.
- Aquellas de las que sí se dispone del código fuente.

Para que las primeras puedan aprovechar el cluster OpenMosix, sólo es necesario revisar que cumplan con los requisitos que se han mencionado en la sección 2.4.5.

Para los programas que se generará un ejecutable a partir del código fuente, el compilador es una pieza muy importante, pues aunque pareciera ser trivial, no todos los compiladores hacen optimizaciones de la misma calida. Esto quiere decir que un compilador puede generar un código objeto o también llamado ejecutable que puede ser ejecutado más eficientemente y con mayor velocidad en la misma arquitectura del procesador donde se trabaja.

Dependiendo del programa que se compile y del tipo de resultados que se desea obtener en cuanto a velocidad de ejecución, es la decisión en el uso de uno u otro compilador. También en esta decisión influye conjuntamente el costo monetario de la adquisición del compilador.

En proyectos de cómputo científico pequeños, para programas en lenguaje C o Fortran, la mayoría de las veces basta con el compilador de software libre como GNU/GCC y librerías de

programación como PVM y LAM/MPI, que también son software libre. En otros casos, por el tipo de proyecto, es conveniente gastar en la adquisición de compiladores comerciales como los que vende Intel y Portran Group entre otros. Estos compiladores como se ha mencionado tienen la capacidad de hacer optimizaciones muy fuertes, además de proveer en la mayoría de los casos herramientas para depurar el código.

3.2.3. Herramientas de administración

Además de las herramientas de administración de procesos y sistemas de archivos locales con las que cuentan todas las distribuciones de Linux, se hace necesario la utilización de otras herramientas para la administración y el monitoreo de los procesos distribuidos en el cluster. Estas herramientas tendrán la tarea de reportar:

- El estado actual de los procesos.
- El estado actual de la carga de cada nodo del cluster.
- El estado actual de tráfico de red.
- El estado actual de los sistemas de archivos distribuidos si se cuenta con ellos.
- Resultados parciales del proceso o procesos en ejecución.

Cada tipo de cluster y dependiendo de las aplicaciones y software que se utilice, hace uso de herramientas que faciliten tareas como las listadas anteriormente. En el caso de un cluster OpenMosix, éste posee un conjunto de herramientas llamadas User Land Tools, éstas son las herramientas básicas que nos ayudan en la administración básica de los procesos que se distribuyen en los nodos así como de su supervisión. En los siguientes capítulos se habla con más detalle sobre el uso de éstas.

Capítulo 4

Diseño del cluster OpenMosix

En este capítulo se describen los componentes de hardware y de software que componen el cluster OpenMosix antes de comenzar con el procedimiento de instalación.

Para este diseño se tomaron en cuenta los mejores equipos y de igual arquitectura con los que se cuenta para obtener un mejor rendimiento de las aplicaciones.

4.1. Organización de los nodos

4.1.1. Hardware

Nodo maestro: Este nodo se eligió por tener el mejor procesador de las computadoras disponibles, tomando en cuenta que los demás fueran de la misma arquitectura.

Las principales características son:

- Procesador:

  Intel(R) Pentium (R) 4.
  Velocidad del procesador (Processor Speed) 2.8 GHz.
  Cache Nivel 2 (L2) de 512 KB.

- Memoria (RAM):

  512 MB DDR 400 MHz.

- Disco Duro (Hard Disk):

  Segate ST31200022A 120 GB, IDE de 7,200 rpm.

- Tarjeta Madre (Motherboard)

  Intel Corporation D865GLC.
  Velocidad del Bus (System Bus Speed) 800 MHz.
  Velocidad de la memoria (System Memory Speed) 400 MHz.

- Tarjeta de red

  Intel (R) Pro/100 Network Conecction.
  Fast Ethernet.

Nodos esclavos: Los tres nodos esclavos son casi idénticos al nodo principal, la única diferencia radica en la velocidad del procesador que es menor a la del nodo maestro.

Las principales características de estos nodos son:

- Procesador:

  Intel(R) Pentium (R) 4.
  Velocidad del procesador (Processor Speed) 2.4 GHz.
  Cache Nivel 2 (L2) de 512 KB.

- Memoria (RAM):

  512 MB DDR 400 MHz.

- Disco Duro (Hard Disk):

  Segate ST38001A 80 GB, IDE de 7,200 rpm.

- Tarjeta Madre (Motherboard):

  Intel Corporation D865GBF.
  Velocidad del Bus (System Bus Speed) 800 MHz.
  Velocidad de la memoria (System Memory Speed) 400 MHz.

- Tarjeta de red:

  Intel (R) Pro/100 Network Conecction
  Fast Ethernet

4.1.2. Software

Sistema Operativo:

Suse 9.0 con kernel OpenMosix 2.4.24: Distribución de Linux que soporta versiones de kernels 2.4.x compatible con openMosix, además de ser la distribución que se utiliza para servidores y estaciones de trabajo de los usuarios en el Instituto de Ingeniería.

Sistema de archivos:

Reiserfs: Sistema de archivo utilizado para las particiones Linux, que al igual que el sistema de archivos ext3, incluye la característica de *journaling* que previene el riesgo corrupciones del sistema de archivos y es de los más utilizados por presentar mejor desempeño en el manejo de archivos.

Mosix File System (MFS): Necesario en OpenMosix para proveer de acceso al sistema de archivos remotos y que ayuda a mantener la consistencia de cache.

Herramientas de administración

- User Lan Tools: Las herramientas de área de usuario son lo mínimo que necesita cualquier administrador o usuario de OpenMosix para el manejo de los procesos.
- OpenMosixView: Aplicación gráfica de monitoreo de OpenMosix, con ella puede realizarse parte de las tareas comunes que se harían con los comandos de las User Lan Tools.

Estas herramientas para administrar, supervisar y migrar los procesos en el cluster OpenMosix se describen en detalle en el capítulo seis.

4.2. Descripción de la conexión de los nodos

En esta sección se da una descripción de la topología de la red y de los componentes de la conexión entre nodos del cluster, como son el switch y el cable de red utilizados. Cabe mencionar que esta configuración es parte de la red actual del Instituto de Ingeniería.

4.2.1. Topología

La topología de red utilizada es la estrella, se caracteriza por tener todos sus nodos conectados a un controlador central, en este caso, un switch de 48 nodos de los cuales 4 nodos pertenecen al cluster. Todas las comunicaciones pasan a través del switch, siendo éste el encargado de controlarlas. Por este motivo, el fallo de un nodo en particular es fácil de detectar y no daña el resto de la red, pero un fallo en el controlador central desactiva la red completa.

4.2.2. Switch

El switch es marca 3Com SuperStack 3, serie 4400.

Una característica importante es que soporta telefonía sobre IP y tiene rendimiento de ancho de banda de 13.6 Gbps (forwarding bandwidth) y 10.1 millones de paquetes por segundo.

El switch es de 48 nodos y también soporta tráfico de telefonía o también llamado VoIP, tecnología con que cuenta el Instituto de Ingeniería, a la cual se le da mayor prioridad cuando es necesario.

Los equipos conectados a este switch se conectan a una velocidad auto negociable de 10Base-T/100Base-TX, según la tarjeta de red.

Cuenta con un puerto Gigabit 1000 Base SX para salir a otro equipo: 3Com 7700, que es el que provee de red ethernet a la Torre de Ingeniería, donde se encuentra físicamente el cluster, como lo muestra la figura 4.1.

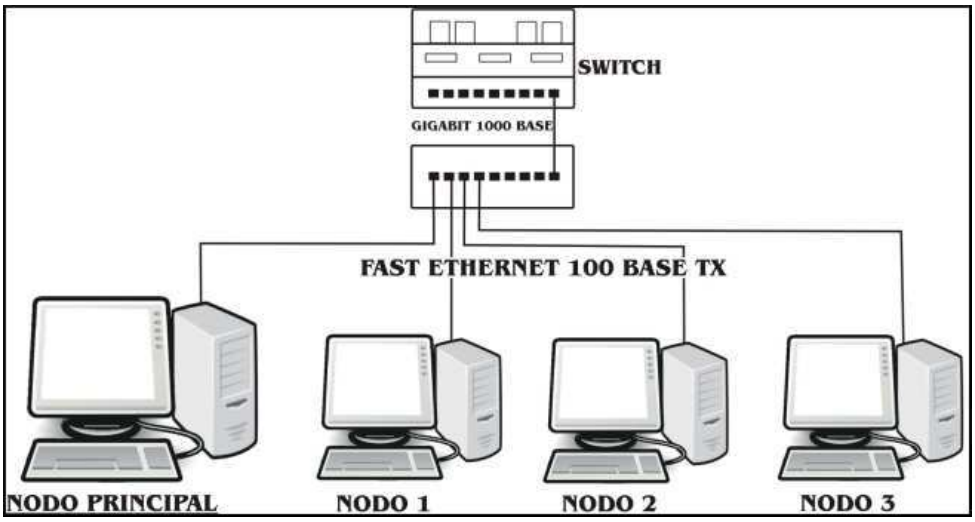

Figura 4.1. Diagrama del cluster

4.2.3. Cable

El cable con el que están conectados los nodos del cluster al switch que los une, es estructurado de la marca Kron categoría 6, para velocidades de transmisión de datos Ethernet 10/100/1000.

4.2.4. Tarjetas de red

Las tarjetas de red con que cuenta cada nodo, son las especificadas anteriormente, todas estas marca Intel, e integradas a las tarjetas madre.

Capítulo 5

Implementación del cluster OpenMosix

Conociendo la terminología básica sobre clusters y habiendo formulado un diseño específico con OpenMosix, ahora se plantea la forma en que se lleva a cabo la implementación.

5.1. Instalación de los nodos

De una forma práctica, en este capítulo se explica la instalación de Suse Linux 9.0 de forma breve, puesto que se hace mayor énfasis en la instalación de OpenMosix.

Descripción de particiones del disco duro:

Los nodos con los que se trabajó son compartidos, es decir se utilizan tanto en ambientes Windows como en Linux y no son de uso exclusivo del cluster. En cada nodo se tiene instalado Windows XP Profesional, Suse Linux 9.3 Profesional y se instaló Suse Linux 9.0 Profesional para configurar el cluster OpenMosix, todo en un mismo disco duro, particionado de la siguiente forma para el nodo maestro y de manera similar para los nodos esclavos:

- 80 GB con Windows XP Profesional SP2 con un sistema de archivos NTFS.
- 1 GB con swap (Memoria de intercambio virtual) para Linux.
- 30 GB con Linux Suse 9.3 Profesional con un sistema de archivos reiserfs.
- 6 GB con Linux Suse 9.0 Profesional con un sistema de archivos reiserfs.

Se muestra con mayor detalle esta configuración con ayuda del comando *fdisk*:

```
# fdisk -l
   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1     10199  81923436    7  HPFS/NTFS
/dev/hda2         10200     10329   1044225   82  Linux swap
/dev/hda3         10330     13853  28306530   83  Linux
/dev/hda4         13854     14592   5936017+  83  Linux
```

Las primeras tres particiones son de uso particular de los usuarios de las computadoras utilizadas, la cuarta partición es la que se usa para la instalación y configuración de OpenMosix.

5.1.1. Instalación de Suse 9.0

Se hace la instalación estándar a través de red con el CD número uno de instalación y la herramienta de instalación y administración Yast [9] la cual nos guiará haciendo las particiones necesarias, y se instalan los módulos de los controladores necesarios según el hardware con el que se cuenta; se utiliza como fuente de instalación el espejo (mirror) http://linux.iingen.unam.mx, donde se cuenta con los archivos fuente de instalación; también se puede hacer ésta con los 5 CDs de instalación de Suse 9.0.

5.1.2. Configuración de la red

Se hizo una configuración estática de direcciones IP, mascara de red, gateway y DNS para una red privada; en este caso, la configuración elegida fue de uso exclusivo del cluster, pues los usuarios de estos equipos no la usan para acceso a recursos de red, como impresiones, trabajo en grupo o Internet.

5.1.3. Actualizaciones (patchs) del sistema

La actualización de los paquetes instalados se hace utilizando la herramienta YOU (Yast On Line Update), necesario para mantener seguro el sistema. Esta actualización se hace también por medio de la red del espejo mencionando.

[9] http://www.novell.com/es-es/documentation/suse/index.html [Última consulta: febrero 2006]

5.2. Configuración de los nodos

En los nodos esclavos sólo se instala el kernel OpenMosix, las User Lan Tools y se configuran los nodos para que arranquen en nivel 3 (multiusuario sin modo gráfico) para mejorar el rendimiento de éstos en el cluster, pues no necesitan un ambiente gráfico para las aplicaciones de ventanas.

En el nodo maestro se instala el kernel OpenMosix, las User Lan Tools y las herramientas adicionales de administración y monitoreo (OpenMosixView); se configura para que arranque en nivel 5 (modo gráfico) y poder aprovechar las herramientas de monitoreo.

5.2.1. Compilación del kernel OpenMosix

Ésta es la parte central del trabajo. Aquí se describe la forma de instalar el kernel o núcleo con el que funciona OpenMosix.

Comenzamos con la instalación del kernel openmosix-kernel-source-2.4.24-OpenMosix2.i386.rpm que soporta el sistema de archivos MFS.

- Se instalan los archivos fuente de OpenMosix con los parches incluidos, los cuales aún no están compilados:

  ```
  # rpm -ivh OpenMosix-kernel-source-2.4.24-OpenMosix2.i386.rpm
  ```

  El código fuente al instalar este rpm, lo coloca en /usr/src/linux-2.4.24-OpenMosix2, por lo que hay que cambiarse a esa ruta.

  ```
  # cd /usr/src/linux-2.4.24-OpenMosix2
  ```

- Es en este momento cuando se procede con la compilación del kernel OpenMosix, limpiando los archivos de código objeto obsoletos que pudieran encontrarse:

  ```
  # make mrproper
  ```

- Se configuran las opciones de compilación del kernel basándose en el archivo de configuración de un kernel de Suse 9.0 precompilado, es decir, con el que ya se está trabajando, pues en este archivo se listan todos los módulos para que nuestro nodo funcione adecuadamente.

```
# make xconfig
```

- Se personaliza de acuerdo al hardware con el que se cuenta, y si es necesario, se quitan los módulos del hardware con el que no se cuenta y se asegura que se encuentren seleccionados los módulos del hardware de los equipos utilizados (figura 5.1).

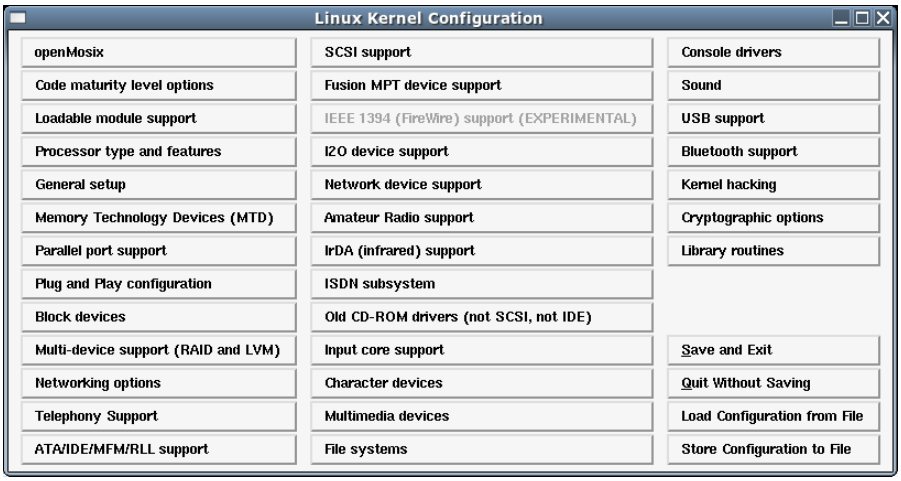

Figura 5.1. Configuración del kernel OpenMosix

- Se habilita el trabajo del kernel con OpenMosix y el Sistema de archivos OpenMosix File-System (5.2).

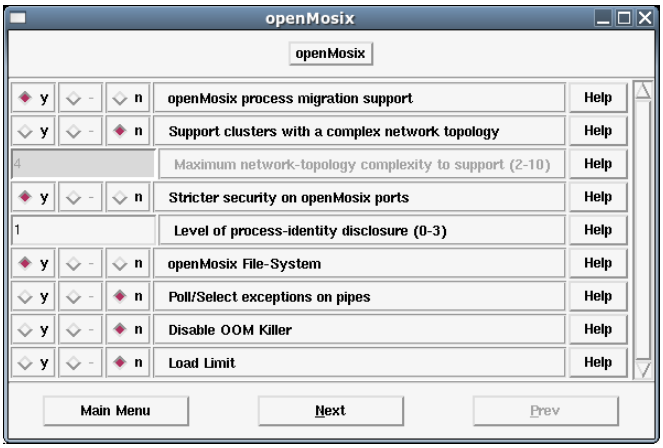

Figura 5.2. Configuración de OpenMosix

- Se resuelven las dependencias de todos los módulos seleccionados para compilarse en el kernel.

```
# make dep
```

- Se crea(compila) el rpm[10] del nuevo kernel con OpenMosix.

```
# make rpm
```

5.2.2. Instalación del kernel

Una vez compilado el kernel se comienza con su instalación.

- Se instala el rpm creado con el siguiente comando:

```
# rpm -ivh /usr/src/packages/RPMS/i586/kernel-2.4.24OpenMosix2-1.i586.rpm
```

- Se crea el initrd para el kernel OpenMosix compilado, necesario para el correcto arranque del sistema.

```
# mkinitrd
```

- Se configura Grub (Sistema gestor de arranque de Suse 9.0) para arrancar con el nuevo kernel OpenMosix, con una configuración como la siguiente en el archivo /boot/grub/menu.lst.

```
title Cluster OpenMosix 2.4.24
kernel (hd0,3)/boot/vmlinuz-2.4.24-OpenMosix2 root=/dev/hda4\
        vga=0x31a splash=silent desktop hdc=ide-scsi hdclun=0 showopts
initrd (hd0,3)/boot/initrd-2.4.24-OpenMosix2
```

- Se agrega la siguiente línea al archivo /etc/fstab para dar de alta el sistema de archivos MFS, tomando en cuenta que el directorio /mfs ya fue creado anteriormente:

```
# vi /etc/fstab
mfs_mnt         /mfs            mfs     dfsa=1          0 0
```

5.2.3. Instalación de las User Land Tools

Una vez instalado el kernel se procede a instalar las herramientas de administración User Land Tools; como requisito, es necesario tener instaladas las librerías ncurses, contenidas en los CDs de instalación de Suse 9.0.

- Se crea el rpm, que es equivalente a compilar las Herramientas de Administración (User Lan Tools), y para esto nos cambiamos al directorio /usr/src:

[10]Sistema de empaquetamiento de Software que facilita la distribución de Software en algunas Distribuciones Linux como RedHat, Mandrake y Suse que tiene entre otras ventajas, tener un mayor control de los archivos en el sistema a la hora de instalarlos o desinstalarlos, además de resolver dependencias de paquetes.

```
# cd /usr/src/
```

- Se crea una liga simbólica, necesaria para la correcta compilación de estos códigos fuente.

```
# ln -s linux-2.4.24-OpenMosix2 linux-OpenMosix
```

- Nos cambiamos al directorio donde se tienen el código fuente de las User Land Tools.

```
# cd /root/srcCluster/toolsOM
```

- Se construye el rpm, equivalente a compilar las librerías, y de esta manera se resuelven dependencias entre paquetes.

```
# rpmbuild --rebuild OpenMosix-tools-0.3.6-2.src.rpm
```

- Se instala el rpm creado.

```
# rpm -ivh /usr/src/packages/RPMS/i586/OpenMosix-tools-0.3.6-2.i586.rpm
```

5.2.4. Instalación de OpenMosixView

Instaladas las herramientas de usuario, se procede a instalar las herramientas de monitoreo OpenMosixView:

- Instalación por medio del paquete rpm.

```
# cd /root/srcCluster/toolsOM
# rpm -ivh openmosixview-1.5-redhat90.i386.rpm
```

  Esta versión precompilada se obtuvo del sitio de internet http://www.openmosixview.com/ donde es mantenido este proyecto.

5.2.5. Configuración de OpenMosix

Después de terminar con la instalación del kernel y de las herramientas, procedemos a configurar el cluster.

- Se configura el archivo /etc/openmosix.map en el cual se listan los nodos pertenecientes al cluster, para esto, se necesita editar este archivo y agregar a la lista cada uno de los nodos, quedando de la siguiente forma:

```
# vi /etc/openmosix.map
1          192.168.1.1          1
2          192.168.1.2          1
3          192.168.1.3          1
4          192.168.1.4          1
```

Como se observa, la primera columna enumera las direcciones IP de los nodos que pertenecen al cluster (columna dos) y la última columna indica el número consecutivo de nodos que pueden pertenecer al cluster, pudiendo con esto hacer abreviaciones de configuración, es decir, la configuración anterior podría haberse escrito de la siguiente manera:

```
# vi /etc/openmosix.map
1         192.168.1.1         4
```

5.2.6. Prueba mínima de funcionamiento

Una vez terminada la configuración del cluster, sólo resta hacer una prueba de funcionamiento. Con el siguiente script en bash shell podemos hacer esta prueba. El funcionamiento de esta se basa en la ejecución de 10 tareas simultaneas. El programa no tiene ningún fin práctico, sólo es para probar la migración de los procesos en el cluster.

Para ejecutar el siguiente script, se escribe este código en un archivo llamado pruebaOM.sh y se ejecuta.

```
#!/bin/bash
#-------------------------------------------------------
# DESCRIPCION: Script que lanza 10 tareas simultaneas.
# Cada tarea itera en un ciclo doble de 10000 unidades.
# El objetivo es probar que el cluster esta migrando las
# tareas a los demás nodos
#-------------------------------------------------------
for x in 1 2 3 4 5 6 7 8 9 10
do
   awk 'BEGIN {for(i=0;i<10000;i++)for(j=0;j<10000;j++);}' &
done
```

Para poder observar si los procesos están migrando, se utiliza el comando mosmon:

```
# mosmon
```

5.3. Instalación de los compiladores

Ahora se describe la instalación del compilador Intel, recordando que ya tenemos el compilador de código libre GNU/GCC, y que sin él no habríamos podido compilar los paquetes antes mencionados. Cabe mencionar que además del compilador comercial Intel existen otros como LAHEY y Portran Group.

Se instala el compilador de Intel debido a que es el que adquirieron investigadores del instituto, al observar algunas ventajas de este sobre GNU/GCC, LAHEY y el de Portran Group.

5.3.1. Compilador Intel

- Una vez obtenido este compilador se descomprime el paquete l_fc_p_9.0.021.tar.gz:

```
linux:~/Intel # gunzip l_fc_p_9.0.021.tar.gz
linux:~/Intel # tar -xvf l_fc_p_9.0.021.tar
linux:~/Intel # cd l_fc_p_9.0.021
```

- Para obtener detalles sobre el producto y la instalación leemos el manual de instalación.

```
linux:~/Intel/l_fc_p_9.0.021 # more INSTALL.txt
```

- Ahora se comienza con la instalación.

```
linux:~/Intel/l_fc_p_9.0.021 #./install.sh

============================================================
"Welcome to Installation"
Please make your selection by entering an option:
1. "Intel(R) Fortran Compiler 9.0 for Linux*" - install
        1a. Readme
        1b. Release Notes
        1c. Installation Guide
        1d. Product Web Site URL
        1e. Intel(R) Support Web Site URL
x. Exit.
Please type a selection  :   1
============================================================
Please select an option to continue:
        1. Proceed with Serial Number to install and register.
         [Recommended]
        2. Provide name of an existing license file.
        x. Exit.
Please type your selection  :   2
============================================================
Please provide the license file name with full path (*.lic)
        x.Exit
License file path  :
   /root/Intel/noncommercial_for_l_NJ24-L2HSGXXP.lic
============================================================
Which of the following would you like to do?
1.    Typical Install (Recommended - Installs All Components).
2.    Custom Install (Advanced Users Only).
x.    Exit.
Please type a selection:   2
Which of the following would you like to install?
1. Intel(R) Fortran Compiler for 32-bit applications, Ver 9.0
2. Linux Application Debugger for 32-bit applications, Ver 9.0
x. Exit.
Please make a selection   :  1
============================================================
Intel(R) Fortran Compiler for 32-bit applications, Ver 9.0
------------------------------------------------------------
Please carefully read the following license agreement.  Prior
to installing the software you will be asked to agree to the
terms and conditions of the following license agreement.
------------------------------------------------------------
Press Enter to continue   :
============================================================
Do you agree to be bound by the terms and conditions of this
license agreement? 'accept' to continue, 'reject' to return
to the main menu   :   accept
============================================================
Processing Intel(R) Fortran Compiler for 32-bit applications,
Version 9.0
Values in [...] are the default values.
You can just hit the Enter key where you want to use the
```

```
default values.
Where do you want to install to?  Specify directory starting
with '/'. [/opt/intel/fc/9.0]: /usr/local/Intel/9.0
============================================================
Where do you want to install to?  Specify directory starting
with '/'. [/opt/intel/fc/9.0]: /usr/local/Intel/9.0
What rpm install options would you like?
[-U --replacefiles --force]:
============================================================
Intel(R) Fortran Compiler for 32-bit applications, Ver 9.0
Installing...
Installation successful.
Press Enter to continue:
============================================================
Which of the following would you like to install?
1.  Intel(R) Fortran Compiler for 32-bit applications, Ver 9.0
2.  Linux Application Debugger for 32-bit applications, Ver 9.0
x.  Exit
Please make a selection:2
============================================================
Values in [...] are the default values.
You can just hit the Enter key where you want to use the
default values.
Where do you want to install to?  Specify directory starting
with '/'. [/opt/intel/idb/9.0]: /usr/local/Intel
============================================================
Where do you want to install to?  Specify directory starting
with '/'. [/opt/intel/idb/9.0]: /usr/local/Intel
What rpm install options would you like?
[-U --replacefiles --force]:
============================================================
------------------------------------------------------------
Linux Application Debugger for 32-bit applications, Version 9.0
Installing...
Installation successful.
------------------------------------------------------------
Press Enter to continue:
============================================================
Which of the following would you like to install?
1.  Intel(R) Fortran Compiler for 32-bit applications, Ver 9.0
2.  Linux Application Debugger for 32-bit applications, Ver 9.0
x.  Exit
Please make a selection:x
============================================================
"Installation is complete "
Thank you for using Intel(R) Software Development Products,
tools for improving application performance.
```

5.3.2. Librerías Intel

Se instalan también las librerías matemáticas del compilador Intel que serán necesarias para la ejecución de uno de los programas de un investigador del Instituto de Ingeniería.

```
linux:~ \# cd Intel
linux:~/Intel # ls
l_fc_p_9.0.021      lib_Intel  link_l_fc_p_9.0.021.txt
l_fc_p_9.0.021.tar  link1.txt  noncommercial_for_l_NJ24-L2HSGXXP.lic
linux:~/Intel/lib_Intel # gunzip *.gz
linux:~/Intel/lib_Intel # tar -xvf *.tar
linux:~/Intel/lib_Intel # cd l_mkl_p_7.2.1.003
linux:~/Intel/lib_Intel/l_mkl_p_7.2.1.003 # ls
install  mklEULA.txt
```

- Después de la lectura del manual de instalación ejecutamos el instalador.

```
linux:~/Intel/lib_Intel/l_mkl_p_7.2.1.003 # ./install

Would you like to install the following?
Intel(R) Math Kernel Library Version 7.2.1
<<Enter>> to continue, or x to exit:
================================================
Where do you want to extract temporary files to?
Specify directory starting with '/' [/tmp/mkl]:
================================================
Intel(R) Math Kernel Library Version 7.2.1
------------------------------------------------
Please read the following license agreement carefully.Prior
to installing the software, you will be asked to agree to
the terms and conditions of thefollowing license agreement.
------------------------------------------------
Please press Enter to continue.
================================================
se lee toda la licencia
================================================
Enter 'accept' to continue, 'reject' to return to the
main menu: accept
================================================
extracting......
================================================
Where do you want to install to?
Specify directory starting with '/'.
[ /opt/intel ]: /usr/local/Intel
Installing Intel(R) Math Kernel Library Version 7.2.1...
================================================
Note:  To uninstall the Intel(R) Math Kernel Library
Version 7.2.1 run
/usr/local/Intel/mkl721/uninstall.sh file.
Intel(R) Math Kernel Library Version 7.2.1 was
successfully installed.
Press Enter to continue.
================================================
<<Enter>> to continue, or x to exit: x
================================================
```

5.3.3. Librerías LAM/MPI

Recordemos que las librerías LAM/MPI son necesarias para compilar códigos paralelos, en el capítulo siete se harán unas pruebas con estos tipos de códigos.

La versión con la cual se trabajo es LAM 7.1.1, 19-Julio-2005.

- Primero cambiarse al directorio donde se encuentra el paquete.

```
# cd /root/srcCluster/LAM
```

- Desempaquetar con el comando tar.

```
# tar -jvxf lam-7.1.1.tar.bz2
# cd lam-7.1.1/
```

- Configurar el paquete para que funcione con el protocolo TCP y compilar.

```
# ./configure --with-rpi=tcp
# make
```

- Una vez que terminó de procesar y compilar podemos instalar.

```
# make install
```

5.4. Seguridad lógica del cluster

Un aspecto importante dentro de cualquier sistema informático es la seguridad lógica de este, y una vez instalado lo necesario para que funcione OpenMosix se procede con la configuración del firewall de los nodos, esto se hace configurando el archivo /etc/sysconfig/SuSEfirewall2.

- Se edita el archivo SuSEfirewall2.

```
# cd /etc/sysconfig/

# vi SuSEfirewall2
```

- En el apartado 10.), se cambia la línea.

```
$FW_TRUSTED_NETS=""
Por:
$FW_TRUSTED_NETS="192.168.1.1/24 192.168.1.2/24\
192.168.1.3/24 192.168.1.4/24"
```

  Las direcciones IP listadas son los nodos del cluster, con esto se logra una comunicación total entre los nodos con una relación de confianza y negando las conexiones entrantes a cualquier otro equipo no autorizado.

- Por último es necesario reiniciar el firewall en cada uno de los nodos para que se actualicen los cambios hechos.

```
# SuSEfirewall2 stop
# SuSEfirewall2 start
```

Para esta configuración del cluster se hace notar que se implementó un modelo de seguridad básica, pues se trabajó dentro de una red interna con direcciones IP privadas en la cual se supone no habría problemas de intrusiones. Idealmente un cluster dedicado debería de tener esta configuración en su red interna, pero si cuidar más de la seguridad del nodo maestro o nodo bastión, el cual normalmente es el que da la cara a las redes públicas.

Capítulo 6

Administración del cluster

Una vez que se ha instalado el cluster, tanto físicamente como lógicamente, es necesario tener el control de éste para su mejor aprovechamiento posible. En este capítulo se describen algunas de las herramientas para administrar, supervisar y migrar los procesos en el cluster OpenMosix.

6.1. Herramientas de área de usuario de OpenMosix

Las herramientas de área de usuario son lo mínimo que necesita cualquier administrador o usuario de OpenMosix para poder trabajar y aprovechar de manera eficiente el cluster.

6.1.1. Monitorizando el cluster

La herramienta usada para monitorizar un cluster OpenMosix es mosmon. Esta herramienta nos permite ver un gráfico de barras con la carga asociada a cada nodo del cluster. De una forma tradicional esta información puede obtenerse mediante el comando *top* en cada uno de los nodos, pero para clusters de más de ocho nodos la solución del top es no viable, y la solución de mosmon es especialmente buena.

Mosmon es una utilidad especialmente interesante por varios puntos adicionales, que no todos sus usuarios conocen, y que lo hacen indispensable para cualquier administrador de sistemas OpenMosix: con ella puede verse todos los nodos definidos en el cluster, estén o no activos, podemos ver el número de nodos activos e inactivos. Aunque este número de nodos sea mayor del que se puede ver en una pantalla, podemos con el cursor derecho y el cursor izquierdo

movernos un nodo a la derecha o un nodo a la izquierda, con lo que podrán verse grandes cantidades de nodos en una pantalla cualquiera. Todo esto hace a mosmon una herramienta realmente imprescindible para el administrador del cluster (figura 6.1).

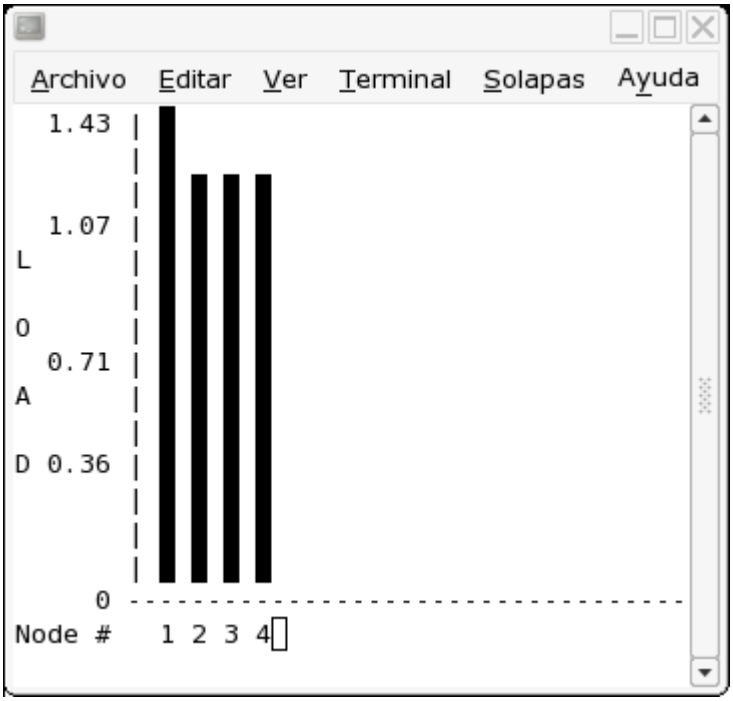

Figura 6.1. Herramienta mosmon

6.1.2. Configurando los nodos en OpenMosix

La herramienta para configurar los nodos de un cluster OpenMosix es **setpe**. Esta herramienta es ejecutada por los scripts de inicialización y parada de OpenMosix, así como por numerosos scripts de OpenMosix y herramientas auxiliares. A pesar de que habitualmente no la llamaremos directamente, es interesante su estudio.

Setpe se encarga de determinar la configuración de nodos del cluster. Su parámetro principal es el archivo donde estará especificado el mapa de nodos. Setpe habitualmente es llamado de tres formas; la primera es con el modificador *-f* nombrearchivo, que tomará como archivo de configuración nombrearchivo; la segunda forma es pasando como parámetro -, en cuyo caso leerá el archivo de configuración de la entrada estándar. Esto es útil para hacer pruebas de configuración. Por último, puede ser llamado con un único parámetro *-off*, para sacar el nodo del cluster.

6.1.3. Controlando los nodos con mosctl

Del mismo modo que *setpe* nos permite ver la configuración de un nodo OpenMosix y modificarla, **mosctl** nos permite auditar el comportamiento de un nodo ya configurado y verlo.

De entre las opciones del comando mosctl, se destacan:

- block: en el nodo local, bloquea la entrada de los procesos generados en otro nodo.
- noblock: deshace los efectos de block.
- mfs: activa el soporte MFS en OpenMosix.
- nomfs: inhabilita el soporte MFS en OpenMosix.
- lstay: bloquea la migración automática hacia fuera de los procesos generados localmente.
- nolstay: deshace los efectos de lstay.
- stay: bloquea la migración automática hacia fuera de cualquier proceso.
- nostay: deshace los efectos de stay.
- quiet: el nodo local no informará a los otros nodos de su estado.
- noquiet: deshace los efectos de quiet.

6.1.4. Migraciones

Para forzar una migración de un proceso en un cluster OpenMosix debemos usar el comando **migrate**. El comando migrate toma dos parámetros: el primero es el PID (Identificador de proceso) que queremos hacer migrar, y el segundo parámetro es donde queremos que migre. Este segundo parámetro debe ser el identificador valido de un nodo en el cluster.

Existen, sin embargo, dos parámetros que podemos colocar en lugar del identificador válido de un nodo del cluster. Estos dos modificadores modelan dos casos especiales, que son:

- home: forza la migración de un proceso al nodo donde fue generado.
- balance: forza la migración de un proceso al nodo donde la migración suponga minimizar el desperdicio de recursos dentro del cluster OpenMosix.

Es una forma de indicar que se evalúe el algoritmo de migración automática de carga de OpenMosix, pero dando preferencia a la migración del proceso del que hemos indicado el PID.

A la hora de lanzar esta migración, en caso de que el proceso sea un proceso lanzado en la máquina donde ejecutamos el comando migrate, debemos ser el administrador de la máquina, el usuario propietario del proceso, el usuario efectivo del proceso, miembro del grupo propietario del proceso o miembro del grupo efectivo del proceso.

Por otro lado, el administrador del sistema de un nodo cualquiera del cluster siempre puede lanzar este comando sobre cualquier proceso que se ejecute en dicho nodo, independientemente de que se haya generado en el nodo local o en un nodo remoto.

En principio, el proceso puede no migrar aunque le lancemos la orden migrate. En caso de que no migre, algunas veces recibiremos un mensaje de error avisando que el comando no funcionó, pero unas pocas veces no migrará, y no recibiremos dicho mensaje. Particularmente esto se da cuando forzamos una migración posible pero pésima: el proceso será mandado de vuelta al nodo local incluso antes de que salga, porque el algoritmo de optimización de carga considerará inaceptable la migración.

La única migración que realmente podemos forzar es la de vuelta a casa, siempre que el nodo de origen no acepte salidas de su nodos con mosctl lstay y no bloqueemos la entrada en el nodo de destino con mosctl block.

6.1.5. Recomendando nodos de ejecución

Cuando lanzamos un proceso, podemos indicar cómo se va a comportar frente a la migración, o dónde se prefiere que se ejecute; para ello, contamos con el comando **mosrun**. Este comando no se suele llamar directamente, sino a través de un conjunto de scripts que facilitan su uso. Con este comando podemos transmitir a OpenMosix la información sobre qué hace el proceso, información que será fundamental para que OpenMosix minimice el desperdicio de recursos del cluster. También podemos indicar un conjunto de nodos entre los cuales estará el

nodo donde migrará el proceso después de lanzado, si esta migración es posible.

En un sistema que no sea OpenMosix, mosrun lanza el proceso en la máquina local de forma correcta.

El comando mosrun siempre tiene la misma estructura de llamada.

```
# mosrun  donde  migración  tipo  comando  argumentos
```

Donde los parámetros son:

- **donde**: nodo al que el proceso va a migrar inmediatamente después de ser lanzado, si esto es posible.
- **migración**: si se bloquea o no el proceso en el nodo de destino.
- **tipo**: tipo de proceso que se lanzaría.
- **comando**: nombre del proceso que se va a lanzar.
- **argumentos**: argumentos del proceso que se va a lanzar.

A continuación se muestran dos ejemplos de cómo utilizar mosrun.

```
# mosrun -h ./programa
```

Obliga a ejecutar el *programa* en el nodo local.

```
# mosrun -j2-4 ./programa
```

Obliga a ejecutar el *programa* en los nodos 2, 3 y 4.

6.2. Herramientas de monitoreo

6.2.1. OpenMosixView

No podemos hablar de OpenMosix pasando por alto OpenMosixView, que es una cómoda y amigable aplicación de monitoreo de un cluster OpenMosix.

OpenMosixView no está en las herramientas de área de usuario de OpenMosix por defecto. Y la razón es muy simple: las herramientas de área de usuario son lo mínimo que necesita cualquier administrador o usuario de OpenMosix para poder trabajar. En la mayoría de las

instalaciones de OpenMosix, los nodos son cajas sin monitor, ratón o teclado con una instalación mínima de Linux, por lo que en principio OpenMosixView sólo será un problema para el administrador, que puede no tener interés en instalar las librerías QT y KDE en una máquina que sólo va a servir procesos.

A diferencia de las herramientas de área de usuario, que tienen una dependencia mínima de bibliotecas y compiladores preinstalados, OpenMosixView necesita otras bibliotecas instaladas para ejecutarse y compilarse.

Sin embargo, esto no quita que OpenMosixView sea una excelente herramienta y de gran utilidad, que todo administrador de OpenMosix podría tenerla instalada en la máquina desde la que inspecciona todo el cluster. Por ello, aunque no la haya incluido, se recomienda a todo administrador que la descargue y la instale en los nodos que quiera utilizar como terminal gráfico del cluster.

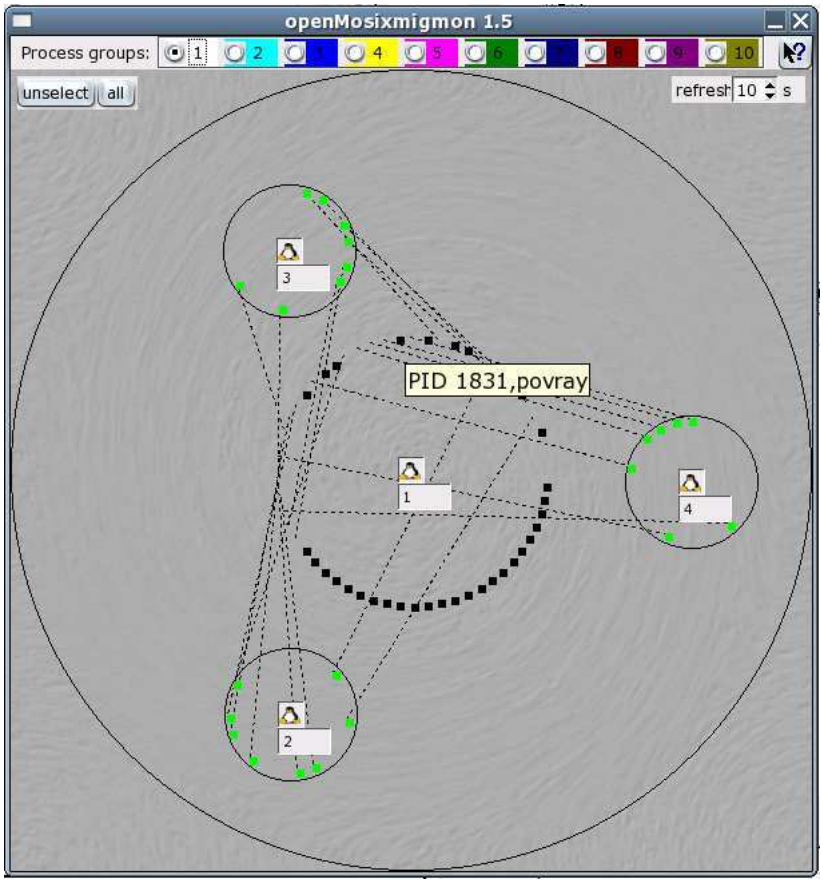

Figura 6.2. Herramienta openMosixMigmon

La suite OpenMosixView contiene siete aplicaciones altamente útiles y eficaces tanto para la administración como para la monitorización del cluster.

- OpenMosixView: principal aplicación de monitorización y administración.
- OpenMosixprocs: aplicación para la administración de procesos.
- OpenMosixcollector: captura la información del cluster proporcionada por los demonios.
- OpenMosixanalyzer: analizador de la información capturada por OpenMosixcollector.
- OpenMosixhistory: historial de monitorización de procesos del cluster.
- OpenMosixmigmon: visor que representa la migración de procesos (figura 6.2).
- 3dmosmon: visor para monitorización de datos en 3D.

Todos los componentes son accesibles desde la ventana de la aplicación principal. Este entorno facilita la interacción con el usuario puesto que le permite ejecutar los comandos de consola más comunes con unos pocos clic de ratón.

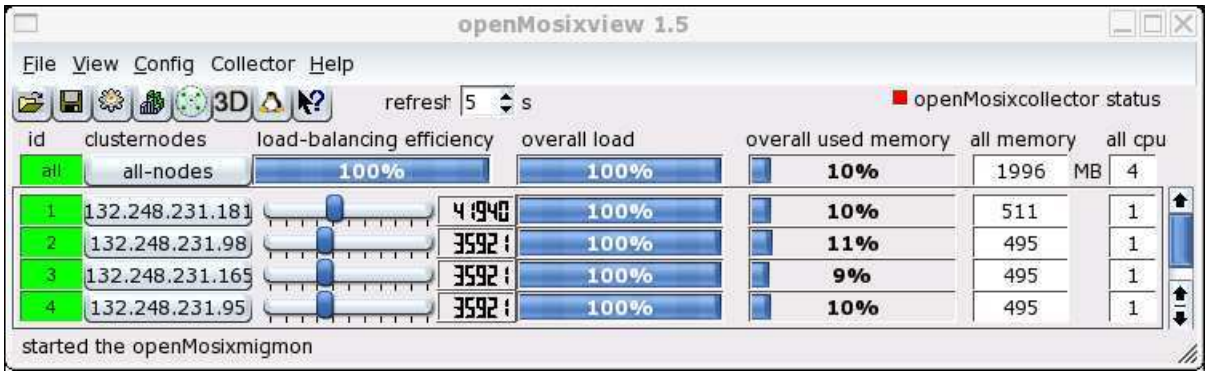

Figura 6.3. Herramienta openMosixView

La figura 6.3 muestra la ventana de la aplicación. El usuario podrá interactuar con OpenMosix a través de sus controles. Para cada nodo del cluster (cada fila): una luz, una barra de velocidad, un número que indica la velocidad de procesamiento, dos barras de progreso porcentual que indican la eficiencia de balanceo de carga y de uso de memoria, también un par de etiquetas que indican la cantidad de memoria y el número de procesadores por nodo.

6.2.2. Programando OpenMosix

Para programar OpenMosix a bajo nivel, es decir, tomando el control del cluster y de la migración, podemos emplear tres mecanismos:

- Hacer uso de las **herramientas de área de usuario**. Este es el mecanismo recomendado para scripts en Perl, o para usar en shell-scripts.
- Hacer uso del **sistema de archivos /proc**. En Linux tenemos un sistema de archivos virtual en /proc. Este sistema de archivos no existe físicamente en el disco, y no ocupa espacio; pero los archivos y los directorios que en el encontramos, modelan distintos aspectos del sistema, tales como la memoria virtual de cada proceso, la red, las interrupciones o la memoria del sistema, entre otras cosas. OpenMosix no puede ser menos, y también podemos obtener información sobre su comportamiento y darle órdenes a través de /proc. Este método de operación es ideal en cualquier lenguaje que no tiene un método cómodo para llamar a procesos externos, y nos permite acceder con facilidad a archivos, leyendo y escribiendo su contenido. Este es el caso de la práctica totalidad de los lenguajes de programación compilados.
- Hacer uso de la **biblioteca de OpenMosix**. El área de usuario de OpenMosix incluye una biblioteca en C que puede ser utilizada para hacer todo aquello que pueden hacer las herramientas de área de usuario. Este mecanismo sólo funciona en C, pero es el más cómodo para los programadores en este lenguaje.

Capítulo 7

Desempeño y pruebas de rendimiento

Ya se ha terminado con la implementación de OpenMosix, la instalación de los compiladores con las librerías necesarias y también se han hecho todas las configuraciones necesarias para el correcto funcionamiento. Ahora se muestran los resultados de pruebas realizadas con algunos programas de uso general y particular que se ejecutaron en el cluster.

7.1. Pruebas generales de rendimiento del cluster

Para cada una de las pruebas, se describe la herramienta o programa que se utiliza para tener un panorama general de la aplicación, posteriormente se explican los resultados que se han tenido para cada una de ellas.

7.1.1. POV-Ray / Povmosix

POV-Ray (Persistence of Vision Ray Tracer), es un avanzado software gratuito para trazado de rayos, el cual es un método para generar imágenes fotorealistas por computadora y se basa en el algoritmo de determinación de superficies visibles de Arthur Appel denominado Ray Casting.

Para describir los objetos se hace uso de un modelador gráfico o mediante un archivo de texto en el que van descritos los objetos que forman la escena. El resultado es un archivo donde se especifican los objetos, texturas, iluminación y la colocación de la cámara virtual que sacará la foto de la escena.

Povmosix[11] es un programa que puede proveer renderizado en paralelo para POV-Ray en

[11] http://povmosix.sourceforge.net/ [Última consulta: febrero 2006]

un ambiente OpenMosix. Éste divide la escena llamada tarea en un conjunto de "subtareas" y éstas pueden migrar en el cluster. Cuando todas las subtareas terminan, la imagen resultante es generada de las salidas parciales.

En este trabajo, se hicieron pruebas con el archivo benchmark.pov, el cual es una demostración elaborada de lo que se puede llegar a generar con POV-Ray y está incluido con los archivos de instalación de este mismo software. Como ya se mencionó, en este archivo de texto, se describe a los objetos de la escena. Por ejemplo, con el código que se muestra a continuación, se describe la colocación y el tipo de letra de un texto en la escena:

```
#declare POV_Text =
text {ttf
      "timrom.ttf"
      "OPENMOSIX-II"
      0.25,0
      scale 0.3
      rotate 90*x
      rotate -90*z}
```

Figura 7.1. Imagen Obtenida con PovMosix

El resultado es una imagen con resolución de 1024x768 píxeles, la cual se muestra en la figura 7.1

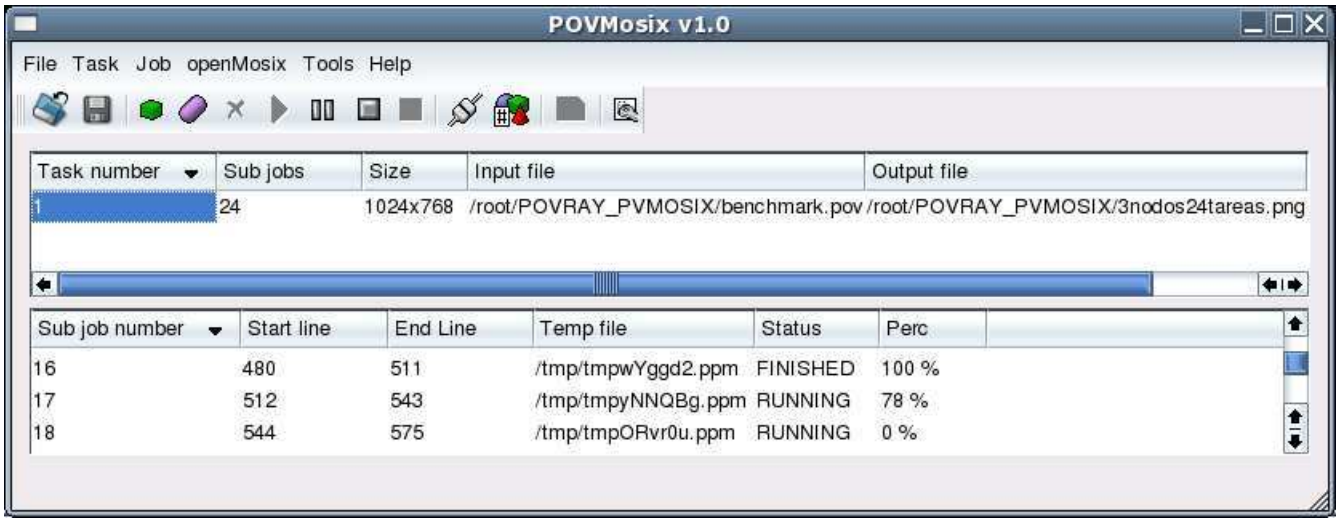

Figura 7.2. Interfaz de ejecución Povmosix.

En la figura 7.2 se muestra la interfaz de ejecución de Povmosix, en ella podemos iniciar y administrar la ejecución de los procesos y subprocesos para la generación de una escena POV-Ray. Con esta herramienta se hizo una serie de ejecuciones con diferente número de nodos y de procesos. Estas se hicieron de acuerdo al cuadro 7.1 y con los resultados mostrados ahí mismo.

Número de procesos	Tiempos de ejecución (min.) 1 Nodo	2 Nodos	3 Nodos	4 Nodos
1	62.5	-	-	-
2	63.4	46.9	-	-
3	-	-	34.29	-
4	65.14	43.42	-	33.44
8	68.31	37.41	29.48	25.58
12	-	39.29	27.37	23.49
16	-	40.5	28.59	22.09
20	-	42.32	29.19	22.49
24	-	-	30.46	23.43
28	-	-	-	24.15

Cuadro 7.1. Comparación de tiempos Povmosix.

Para una mejor comprensión del cuadro 7.1, se muestra la figura 7.3. Como puede observarse en ella, en el caso específico de Povmosix, el rendimiento del cluster OpenMosix es mejor cuando el número de tareas es mayor que el número de nodos activos, pero esto sólo hasta cierto límite. Esta particularidad es debida a que cada subtarea generada por Povmosix tiene

diferente carga de trabajo y por tanto no tiene el mismo tiempo de ejecución.

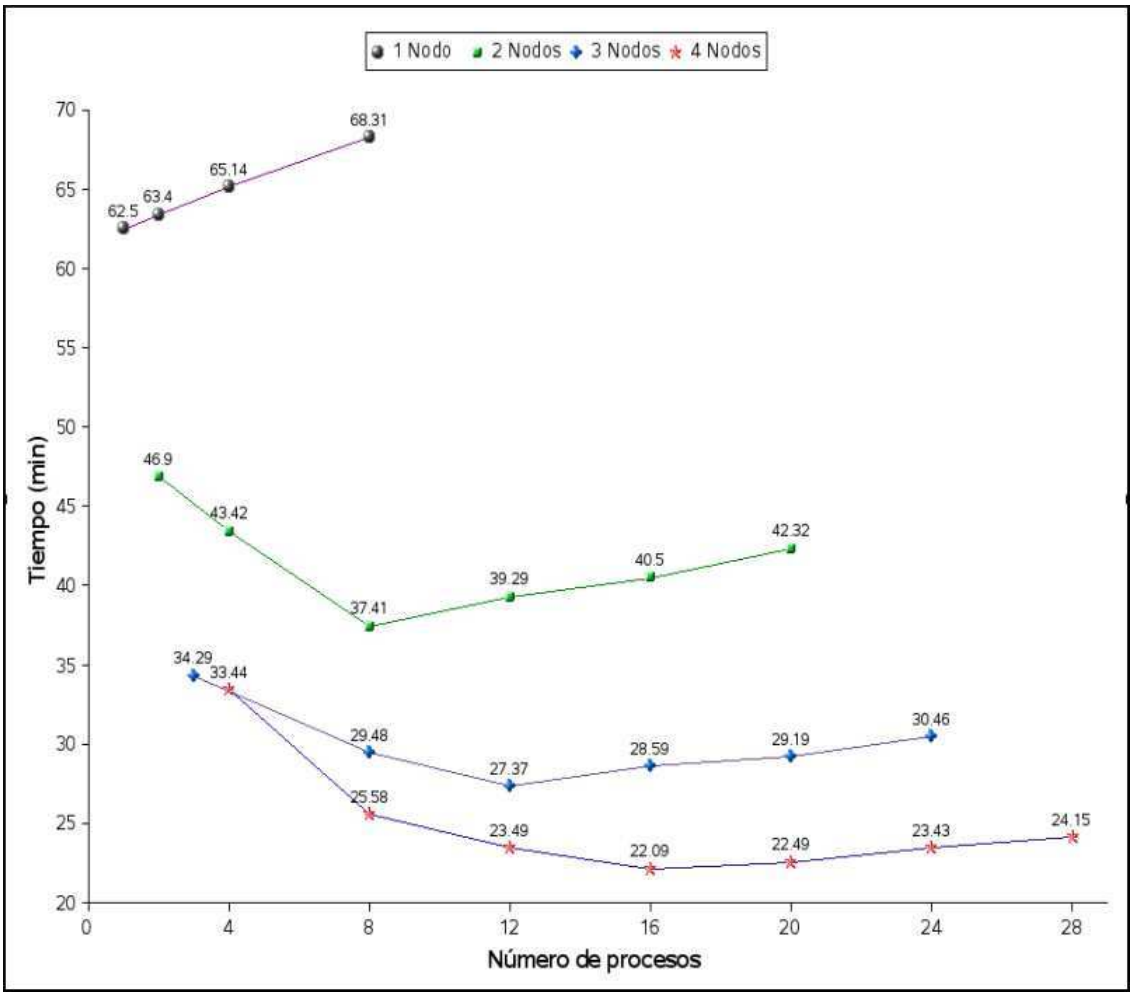

Figura 7.3. Resultados y tiempos de ejecución para Povmosix.

Para la generación de estas imágenes fotorealistas, con la configuración del cluster y con ayuda de Povmosix, podemos tomar ventajas para optimizar tiempos de generación de escenas.

En esta primera aproximación al uso del cluster OpenMosix se observa la utilidad que se puede dar a una herramienta como ésta y se muestra como pueden obtenerse mejores resultados dependiendo del número de tareas que se ejecuten en los nodos disponibles. Otra observación importante que se hace, es que esta mejora en rendimiento al aumentar el número de tareas es debida muy particularmente a la forma en que trabaja Povmosix. Hay que tomar en cuenta que no todos los programas están diseñados de igual manera y por lo tanto no todos pueden tomar las mismas ventajas.

7.1.2. Programas paralelos con LAM/MPI, ejemplo del cálculo del área bajo la curva de una función

Se hicieron también pruebas de rendimiento con un programa paralelo con MPI, el código de este puede leerse en el apéndice A.

El programa consiste en calcular el área bajo la curva definida por el polinomio $f(x) = 10.5 + 1.2x + 0.8x^2 - 2.25x^3 + 0.5x^4$ con limite inferior en 0.0 y superior en 9.5. Cabe mencionar que este programa no tiene utilidad inmediata en este contexto más que para pruebas del cluster.

Para compilar y ejecutar este programa se utilizan los siguientes comandos:

```
$lamboot
 LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University
$mpicc mpi_integral2.c -lm -o mpi_integral2.out
$mpirun -np 4 mpi_integral2.out
Integrando la función Y = 10.5 + 1.2x + 0.8xˆ2 - 2.25xˆ3 + 0.5xˆ4 en
  150000000 intervalos...
Limite Inferior: 0.000000
Limite Superior: 9.500000
proc 1 computes: 884.686245
proc 2 computes: 884.686281
proc 0 computes: 884.686210
proc 3 computes: 884.686316
The integral is 3538.745052
```

En este caso, el número 4 es el número de subprocesos que generará el programa.

Número de procesos	Tiempos de ejecución (seg.)			
	1 Nodo	2 Nodos	3 Nodos	4 Nodos
1	96.09	-	-	-
2	96.23	57.95	-	-
3	96.24	56.29	40.15	-
4	96.26	56.39	43.61	32.25
8	96.13	57.73	44.84	34.95
12	96.21	59.07	45.18	37.93
16	96.15	61.04	48.4	41.34
20	96.28	61.58	50.66	43.13
24	96.34	64.86	53.57	47.16
28	96.29	66.64	56.92	51.16
32	96.35	69.01	61.11	60.04

Cuadro 7.2. Comparación de tiempos para el programa paralelo.

En el cuadro 7.2 se muestran los resultados de los tiempos resultantes de las pruebas que se hicieron en el cluster.

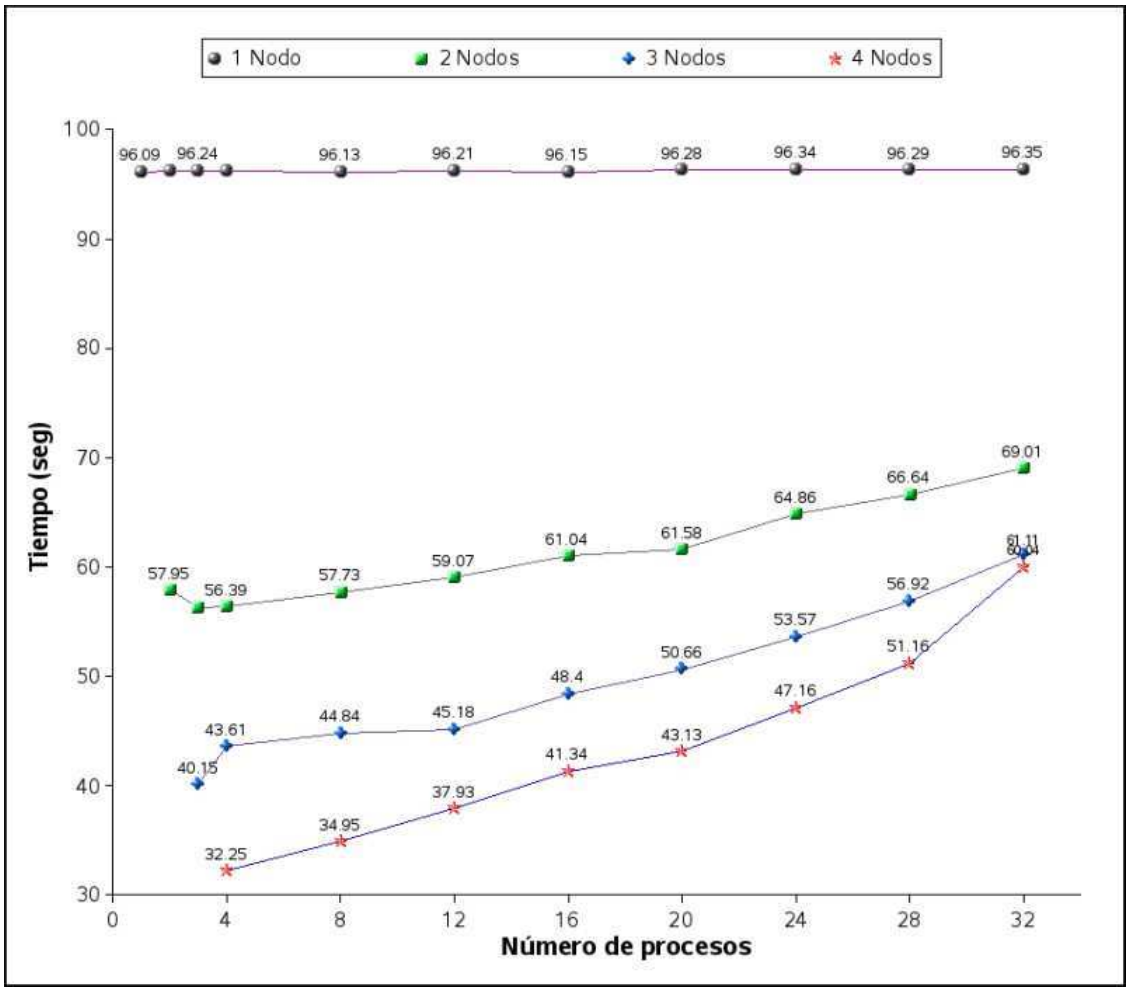

Figura 7.4. Comparación de tiempos del programa paralelo.

Para mejor comprensión de los resultados, se muestra la figura 7.4, con los tiempos obtenidos en las diferentes ejecuciones con uno, dos, tres y cuatro nodos. Como se observa en ella, los mejores tiempos de ejecución se dan cuando el número de subprocesos es igual al número de nodos, lo cual no sucede conforme aumenta el número de procesos en los que se divide la ejecución, en la mayoría de los casos cuando el número de subprocesos es mayor al número de nodos, el tiempo aumenta. Lo cual implica que este programa no es un buen candidato para dividirse en muchas tareas, pero sí para ejecutarse en muchos nodos.

También es importante mencionar que si el número de procesos es menor al número de nodos, se estaría desaprovechando al resto de los nodos.

7.2. Ejecución de programas del Instituto de Ingeniería

En esta sección se describe el funcionamiento de los programas utilizados en las coordinaciones en las cuales se ha colaborado y se hizo la implementación del cluster.

7.2.1. Coordinación de Ingeniería Sismológica

Debido a la necesidad de considerar los efectos destructivos que pudieran tener eventos sísmicos como los de 1985, en la Coordinación de Ingeniería Sismológica se procesa un modelo de la corteza terrestre mexicana con influencia de sismos generados en diferentes epicentros y su influencia sobre la Ciudad de México. Esto para ayudar a tomar medidas de prevención, analizando las zonas que pudieran sufrir los daños más severos.

El programa utilizado en esta etapa de pruebas de rendimiento, fue desarrollado por el M. en C. Hugo Cruz Jiménez. En este trabajo se aplicó el método pseudo-espectral para la obtención del campo de onda incidente en la Ciudad de México; el campo de ondas se expande en el espacio en términos de polinomios de Fourier, y las derivadas parciales espaciales se calculan en el dominio del número de onda. En este estudio se consideró conveniente simular el movimiento en 2D para evaluar los efectos de la estructura de la corteza heterogénea y el manto superior de México[HCruz04].

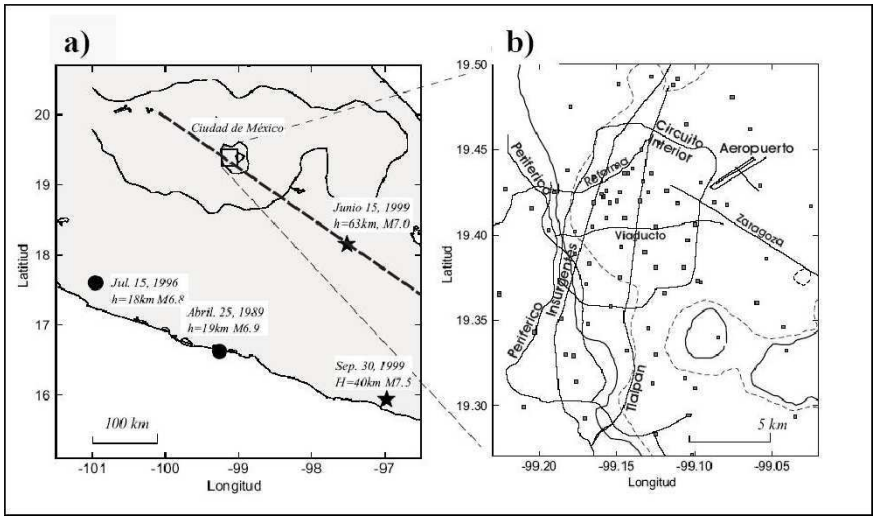

Figura 7.5. Mapa de localización de sismos.

En la figura 7.5 a, se muestra la localización de los sismos considerados. Los círculos negros y las estrellas muestran los epicentros y la Ciudad de México, y en la figura 7.5 b, se muestran las

localizaciones de las estaciones de movimiento fuerte (cuadros) y algunas avenidas importantes en la Ciudad de México.

La zona modelada tiene 512 Km. de largo por 128 Km. de profundidad. Se utilizó un espaciamiento uniforme de la malla de 0.125 Km., para minimizar las reflexiones artificiales de los bordes del modelo y se utilizó una zona absorbente con 20 puntos de la malla.

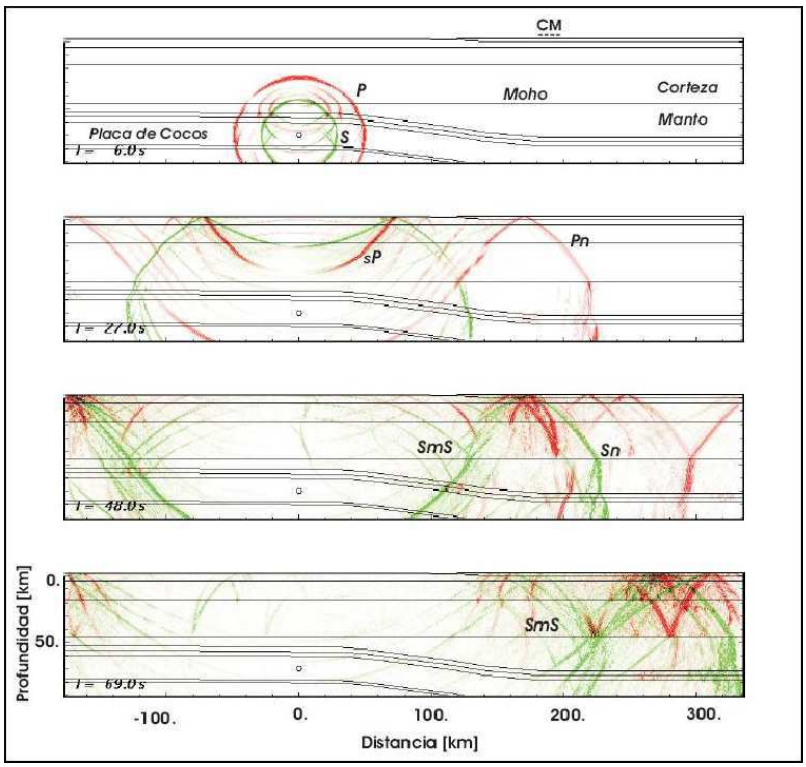

Figura 7.6. Instantáneas de modelado numérico en dos dimensiones.

En la figura 7.6, se muestran cuatro instantáneas obtenidas a partir del modelado numérico en dos dimensiones.

Este programa se ejecuto para 10000 y 20000 iteraciones, con las cuales las salidas arrojadas permiten al investigador tener diferentes resoluciones para observar sus resultados.

Los tiempos observados con las ejecuciones de este programa y cálculos promedio se pueden ver en el cuadro 7.3, y para una mejor comprensión de éste, se muestra la figura 7.7, con los tiempos obtenidos en las diferentes ejecuciones. Como puede observarse en ésta figura, los tiempos al ejecutar el proceso mejoran conforme se ejecuta en más procesadores y no al ejecutarse

Número de procesos	Tiempos de ejecución (min.)			
	1 Nodo	2 Nodos	3 Nodos	4 Nodos
1	217.7	212.14	218.4	215.2
2	-	235.56	236.15	224.45
3	-	-	237.80	235.51
4	-	-	-	236.25
Tiempo promedio	217.7	223.85	230.78	227.85

Cuadro 7.3. Comparación de tiempos promedio para 10000 iteraciones.

con un mayor número de procesos. Lo que implica que solo se ejecuta en un procesador a la vez y la única ventaja es la de poder ejecutar este programa con diferentes datos al mismo tiempo.

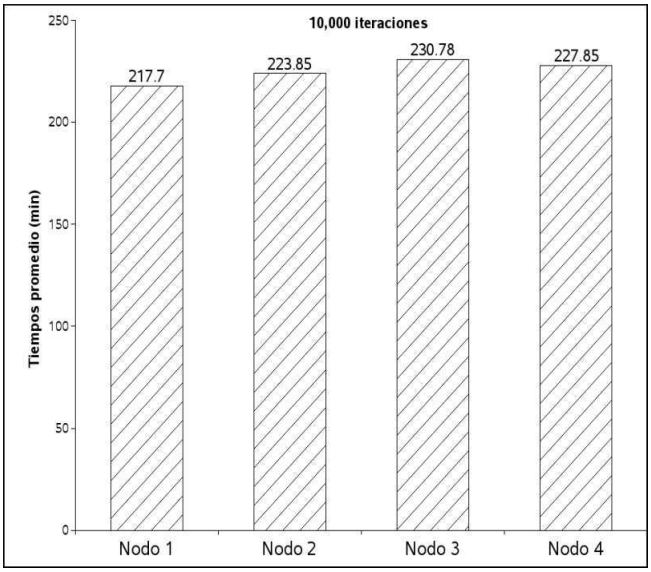

Figura 7.7. Comparación de promedios de tiempos de ejecución

Los tiempos observados con las ejecuciones de este programa y calculando promedios se pueden observar en el cuadro 7.4, y para una mejor comprensión, se muestra la figura 7.8, con los tiempos obtenidos de las diferentes ejecuciones. Como se observa en la figura, los tiempos al ejecutar el proceso mejoran conforme se ejecuta en más procesadores y no al ejecutarse con un mayor número de procesos, lo que implica que sólo se ejecuta en un procesador a la vez y la ventaja es la de poder ejecutar este programa con diferentes datos al mismo tiempo.

Número de procesos	Tiempos de ejecución (min.)			
	1 Nodo	2 Nodos	3 Nodos	4 Nodos
1	433.57	467.14	419.19	443.55
2	-	559.3	465.36	464.34
3	-	-	465.39	468.51
4	-	-	-	481.29
Tiempo promedio	433.57	513.22	449.98	464.42

Cuadro 7.4. Comparación de tiempos promedio para 20000 iteraciones.

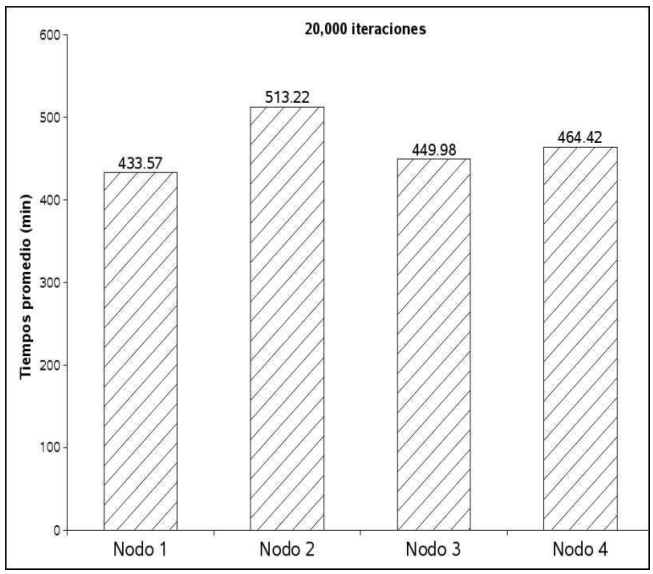

Figura 7.8. Comparación de promedios de tiempos de ejecución

7.2.2. Coordinación de Mecánica Aplicada

La evolución de la arquitectura de las estructuras a través de los tiempos ha estado subordinada a distintos factores que influyeron para que se desarrollara en un determinado periodo y lugar. Entre ellos se pueden señalar el tecnológico, el cual está ligado al grado de conocimiento y habilidad de los diseñadores y constructores de cada época y que está influenciado de manera importante por la cantidad y calidad del material para cada realización. Es así que la mampostería toma su lugar en la historia de la ingeniería[GRoeder04].

En la Coordinación de Mecánica Aplicada del Instituto de Ingeniería, trabaja un grupo dirigido por el Doctor Gustavo Ayala Milián, aplicando métodos de mecánica numérica. Es el caso del trabajo realizado por el Doctor en Ingeniería Guillermo Martín Roeder Carbo descrito a continuación (ver figura 7.9).

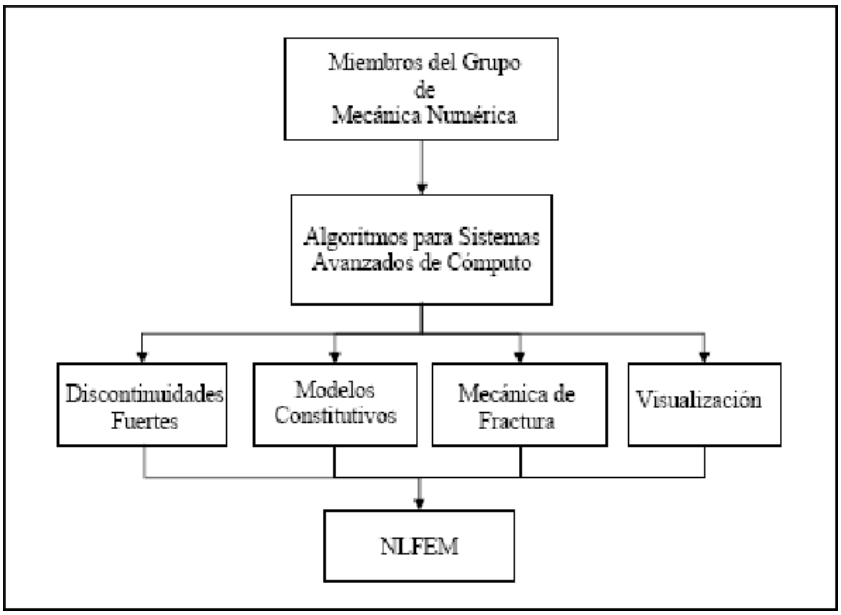

Figura 7.9. Estructura y organización del Grupo de trabajo para el desarrollo de NLFEM[GRoeder04]

Mediante modelos matemáticos y algoritmos, se han desarrollo e implementado herramientas numéricas para modelación de estructuras de mampostería por el método de elementos finitos, donde con el avance de las nuevas investigaciones se han ido incorporando nuevos procedimientos para la solución de sistemas de ecuaciones no-lineales. La meta de este trabajo fue el desarrollo de una herramienta computacional bajo un sistema robusto para el análisis no-lineal de estructuras de mampostería.

Así, el resultado es NLFEM(Non-Linear Finite Element Models), programa que permite investigar el comportamiento físico y/o mecánico de una gran variedad de sistemas estructurales gracias a la amplia variedad de tipos de elementos incorporados en el que pueden ser aplicados a estructuras de concreto simple y reforzado, asimismo como a estructuras de mampostería. También se puede realizar modelados de problemas de la mecánica de suelos.

Al igual que con el programa de la Coordinación de Ingeniería Sismológica se intentó aprovechar el poder de cómputo que ofrece el cluster OpenMosix. En este caso el resultado final no fue satisfactorio y con el objetivo de entender cuando un programa no puede hacer uso de este tipo de clusters, se hace la siguiente explicación:

NLFEM se comenzó en el año de 1999 y desde entonces ha sido extendido, adaptándose a los requerimientos de las investigaciones. Desde su comienzo, este sistema no fue planeado

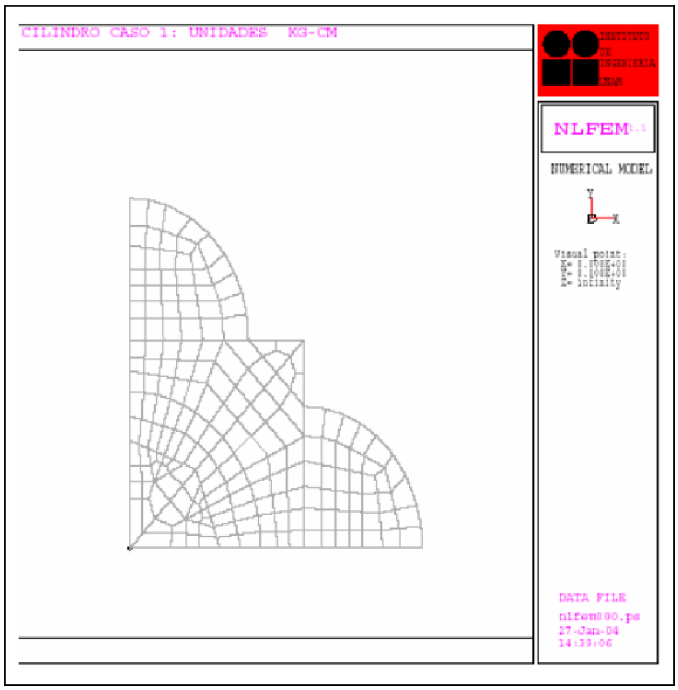

Figura 7.10. Pantalla de resultados, generada por NLFEM

para poder ejecutarse en un cluster, aunque se sabía, que con el tiempo tendría requerimientos más grandes de cómputo, los cuales serían satisfechos con computadoras monoprocesador o multiprocesador de vanguardia, situación que provoca grandes gastos en la compra de estos recursos.

Por otro lado, la programación del sistema se hizo con tres lenguajes: C, C++ y FORTRAN 77. El programa completo está constituido por otros subprogramas ejecutables escritos en FORTRAN 77 que son llamados entre si mediante el control de procesos padres a procesos hijos, lo cual pone en desventaja el aprovechamiento del cluster, pues como se menciona en la sección 2.4.5 sobre los requerimientos mínimos para la migración de procesos, éstos no deben de hacer uso de la memoria compartida, situación que se genera con la metodología usada en la programación de este sistema.

Otro análisis que se hizo con este sistema, fue sobre la duración en tiempo de ejecución de cada uno de los subprocesos generados por el programa principal, observando que estos tiempos, en el orden de segundos, son muy pequeños y debido al algoritmo de balanceo de carga que utiliza OpenMosix, no es posible que estos migren a otros nodos, pues tardaría más tiempo en trasladarse a otro nodo, que si se ejecutara en el nodo anfitrión.

Conclusiones

La ejecución de programas y los resultados obtenidos al final de este trabajo, muestran que no todos los programas son capaces de aprovechar la infraestructura en un cluster, ya sea que por su diseño no fueron pensados para este propósito o por que el tipo de problema no se presta para ello, esta conclusión se obtuvo a partir de la evaluación particular del programa de la Coordinación de Mecánica Aplicada, lo cual hace necesaria una fuerte interacción entre investigadores y profesionales de la computación. Además se observa como no necesariamente entre mayor número de procesos sea dividida una tarea, ésta deba ser más rápida en su ejecución como lo muestran los resultados obtenidos durante la ejecución de los programas con código paralelo y el de la Coordinación de Ingeniería Sismológica.

Al dar a conocer este tipo de tecnologías a los investigadores del Instituto de Ingeniería se ha observado el aumento en el interés, lo cual permite concluir que es necesario estar pendiente de este tipo de avances tecnológicos, no sólo por ser lo que marca la tendencia, si no también por los grandes beneficios que ofrecen en su utilización. Además con la construcción e implementación de este cluster, se concluye que no se necesitan muchos recursos económicos para iniciar el uso de este tipo de tecnologías y comenzar a disfrutar de los beneficios que estas ofrecen, así también, se pueden aprovechar los recursos de cómputo ya existentes.

El comenzar la ejecución de los programas de las coordinaciones de Ingeniería Sismológica y de Mecánica Aplicada, generó el interés por parte de los investigadores y algunos ya están trabajando para implementar sus sistemas con el paradigma de programación paralela, para el mejor aprovechamiento de los clusters en el Instituto de Ingeniería, así como también los clusters de otras instituciones con los que ellos colaboran.

Al inicio este trabajo de tesis en el Instituto de Ingeniería, esta institución no contaba con un cluster dedicado únicamente al cómputo científico, en el proceso de este trabajo de tesis se adquirió equipo para formar parte del primer cluster con el que contó esta institución, lo que

hace concluir que este trabajo fue de gran importancia en la toma de ésta y futuras decisiones.

Durante esta investigación se hizo visible la necesidad del trabajo multidisciplinario, lo cual es indispensable en una institución de la magnitud del Instituto de Ingeniería, con lo que se concluye que fomentar esta forma de trabajo es indispensable para mantener el nivel de este centro de investigación.

Pudo afirmarse que en el cluster OpenMosix también es posible la ejecución de programas de código paralelo, con la instalación de las respectivas librerías, lo que ofrece una ventaja sobre los clusters tipo Beowulf.

Como trabajo a futuro se pretende dar seguimiento a la implementación de nuevos programas paralelos y migración de los ya existentes, esto con el fin de aprovechar los sistemas de clusters más avanzados y/o complejos que hacen buen uso de los conceptos de programación paralela. Otro punto a considerar será la evaluación de las nuevas versiones de OpenMosix que ya aprovechan algunas arquitecturas de 64 bits.

También durante el desarrollo de esta investigación, y como parte del trabajo académico y de difusión que tiene como objetivo el Instituto de Ingeniería, se tuvo la oportunidad de presentar los avances de este trabajo en diversos congresos nacionales, así mismo se prestó asesoría a proyectos vinculados con este instituto, tal es el caso de la construcción de un cluster para la empresa PEMEX.

Apéndice A

Ejemplo de código paralelo MPI para el cluster OpenMosix

Código paralelo para el cálculo del área bajo la curva de una función f(x) [1].

```
 /**
 * Este programa calcula el área bajo la curva del polinomio
 * f(x)= Y = 10.5 + 1.2x + 0.8x^2 - 2.25x^3 + 0.5x^4 con límite
 * inferior 0 y límite superior 9.5 haciendo uso de la librería
 * MPI para código paralelo. La función f(x) puede ser cambiada
 * en f_de_x(), ver más abajo.
 * Código modificado de la fuente:
 * http://www.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml {Enero 2006}
 **/
#include <stdio.h>
#include <math.h>
#include <mpi.h>

#define PI 3.1415926535
#define LIM_INF 0.0 /*Limite Inferior de la Integral*/
#define LIM_SUP 9.5 /*Limite Superior de la Integral*/

double f_de_x(double x);
/*En el cuerpo de esta función puedes cambiar el calculo para f(x)*/

main(int argc, char **argv)
{
   int my_id, root_process, num_procs, ierr, num_intervals, i;
   double rect_width, area, sum, x_middle, partial_sum;
   MPI_Status status;

   /* 0 es el proceso raíz. */
   root_process = 0;

   /* Replicar el proceso para crear procesos paralelos. */
   ierr = MPI_Init(&argc, &argv);

   /* Encuentra MI ID de proceso y cuantos procesos fueron iniciados. */
   ierr = MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
   ierr = MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
   if(my_id == root_process) {
```

[1] http://www.cc.ku.edu/ grobe/docs/intro-MPI-C.shtml [Última consulta: febrero 2006]

```
         /*Modificar el código de las siguientes líneas si se quiere
          *preguntar al usuario cuantos procesos iniciar.
          *printf("Please enter the number of intervals to interpolate: ");
         scanf("%i", &num_intervals);*/
        num_intervals = 150000000;
        printf("Integrando la función \
        Y = 10.5 + 1.2x + 0.8x^2 - 2.25x^3 + 0.5x^4 \
        en %d intervalos...\n",num_intervals);
        printf("Límite Inferior: %f\n",LIM_INF);
        printf("Límite Superior: %f\n\n",LIM_SUP);
      }

      /* Se envía un mensaje a todos los subprocesos del número de
       * intervalos. */
      ierr = MPI_Bcast(&num_intervals, 1, MPI_INT, root_process,
            MPI_COMM_WORLD);

      /* cálculo de la base del rectángulo, y ... */

      rect_width = (LIM_SUP - LIM_INF) / num_intervals;

      /* se calcula la suma de las áreas de los rectángulos de las cuales el
       * subproceso es responsable. */
      partial_sum = 0;
      for(i = my_id + 1; i <num_intervals + 1; i += num_procs) {
         x_middle = (i - 0.5) * rect_width;
         area =  f_de_x(x_middle) * rect_width;
         partial_sum = partial_sum + area;
      }
      printf("proc %i computes: %f\n", my_id, partial_sum);

      /* Finalmente las sumas parciales son recogidas en la variable "sum" */
      ierr = MPI_Reduce(&partial_sum, &sum, 1, MPI_DOUBLE,
            MPI_SUM, root_process, MPI_COMM_WORLD);

      /* Si soy el proceso raíz, imprimo el resultado */
      if(my_id == root_process) {
         printf("The integral is %f\n", sum);
      }

      /* finaliza el proceso. */
      ierr = MPI_Finalize();
   }

double f_de_x(double x)
{
    double punto = 0.0, valor = 0.0;
    double poli[]={10.5,1.2,0.8,-2.25,0.5};
    int  i = 0;

    punto = x;
    for(i=0;i<5;i++)
      valor += poli[i]*pow(punto,i);
    return valor;

/*El usuario puede definir otras funciones, Ejemplos:
1) return sin(x);

2) return cos(x);

3) return x*x;

4) return x/2;
*/
}
```

Bibliografía

[ATanenbaum03] Andrew S. Tanenbaum, *Sistemas Operativos Modernos*, Pearson Educación, 2003.

[EGeorge03] George Em Karniadakis and Robert M. Kirby II, *Parallel Scientific Computing in C++ and MPI*, Cambridge University Press,2003.

[GRoeder04] Guillermo Martín Roeder Carbo, *Simulación numérica del comportamiento mecánico de la mampostería*, Tesis de Doctorado, UNAM 2004.

[HCruz04] Hugo Cruz J., *Simulación numérica del movimiento fuerte en la Ciudad de México*, Tesis de Maestría, UNAM 2004.

[HDavid00] David HM Spector, *Building Linux Clusters*, O'Reilly Media, Inc. 2000.

[MCatalan04] Miquel Catalán i Coït*El manual para el clustering con openMosix*, Copyright c miKeL a.k.a.mc2 & Kris Buytaert. Versión 1.0 - 6 de Septiembre de 2004.

[RichardS03] Ruichard S. Morrison, *Cluster Computing. Architectures, Operating Systems, Parallel Processing & Programming Languages*, 1998-2003.