UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"CONSTRUCCIÓN DE UN CLUSTER MOSIX: PRUEBAS CON SIMULACIÓN DE HALOS"

TESIS

PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

PRESENTA:

LEONARDI MARTÍN TORRALBA MORALES

ASESORES:

M.C. GABRIEL GERÓNIMO CASTILLO
DR. JOSÉ JAVIER BAEZ ROJAS
DR. FÉLIX AGUILAR VALDEZ
DR. ALEJANDRO CORNEJO RODRIGUEZ

Huajuapan, Oaxaca., Diciembre de 2002.

Dedicatoria

A mis Padres ***Margarita y Francisco***†, como agradecimiento por la ayuda y confianza que siempre me han brindado y por haberme dado la mejor herencia, mi educación.

En especial a la memoria de mi Padre ***Francisco***† que espiritualmente siempre está con migo.

A mi Tía ***Mary*** por su gran apoyo que me ha brindado en todo momento.

A mis hermanos por haberme apoyado, aconsejado y por ser parte importante de una familia unida.

Agradecimientos

- En primera instancia agradezco a la ***Universidad Tecnológica de la Mixteca,*** por haberme formado como Ingeniero en Computación.
- Quiero agradecer a mi asesor ***M.C. Gabriel Gerónimo Castillo***, por su gran ayuda y paciencia durante el tiempo del desarrollo de mi tesis.
- Agradezco al ***Dr. Javier Báez, Dr. Félix Aguilar y Dr. Alejandro Cornejo,*** por haberme dado la oportunidad de trabajar con ellos y por haberme brindado su ayuda y paciencia.
- Agradezco también al ***Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)*** por haberme brindado la oportunidad y facilidad de desarrollar buena parte de mi tesis en sus instalaciones.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

PREFACIO

El ***objetivo*** de esta tesis, es construir un cluster usando el sistema operativo Linux y la herramienta MOSIX para realizar distribución de procesos entre los nodos del cluster.

Para probar el funcionamiento de la distribución de procesos se ha desarrollado una aplicación que consiste en la simulación de un efecto óptico llamado "halo". Cabe mencionar que el cluster MOSIX no está dedicado a esta aplicación solamente si no que puede ejecutarse cualquier otra aplicación, siempre y cuando las necesidades de los usuarios así lo requieran, es decir, el cluster una vez configurado es capaz de funcionar con cualquier otra aplicación.

La ciencia de la computación en los últimos años ha tenido un avance de manera considerable a tal grado que algunas computadoras en poco tiempo llegan a ser obsoletas para ciertos usos. Este avance se ha dado tanto en la parte de hardware como en la parte software.

La ***justificación*** de esta tesis es la siguiente, en la actualidad cada vez más se necesitan resolver problemas complejos desde el punto de vista computacional que requieren mayor poder de cómputo del que puede ofrecer una computadora personal, debido a esta razón, se ha tenido la necesidad de construir computadoras con mucha mayor capacidad de procesamiento y almacenamiento, la solución más obvia es utilizar una supercomputadora, pero existe el problema que además de ser demasiado costosa su mantenimiento también es caro. Estas son las principales razones por las que surgió la idea de la tesis de construir un cluster ya que es una forma de hacer procesamiento distribuido o paralelo en forma económica.

Un cluster puede ofrecer un alto rendimiento en la ejecución de aplicaciones ya que todas las computadoras (nodos) trabajan al mismo tiempo para resolver un mismo problema. Una ventaja más de un cluster es que pueden tenerse nodos remotos y configurados de tal manera que en las horas que se encuentran ociosos se activen como parte del cluster y se desactiven en horas en las que el usuario trabaja, lo cual nos da cierta flexibilidad en la forma de trabajar.

Como ***antecedentes*** tenemos que antes de la construcción del primer cluster, la única forma de trabajar en la solución de problemas que necesitaban demasiado poder de cómputo solamente era usando supercomputadoras, las cuales son máquinas que tienen arreglos de procesadores que trabajan en forma conjunta para resolver cualquier problema.

Los ***clusters en los últimos años*** han tomado cierta fuerza tanto en universidades como en las empresas debido a que es una forma barata de hacer cómputo paralelo. Se afirma que es una forma barata por dos razones principales; con respecto al hardware, actualmente las PC's son más económicas, su rendimiento aumenta cada vez más, el material de interconexión de una red es más económico y más sofisticado, con respecto al software, existen piezas de software, tales como el sistema operativo Linux y herramientas para

administración, análisis, monitoreo y balanceo de cargas que se pueden obtener de forma gratuita.

Los clusters están presentes en muchas aplicaciones, ya que no son construidos para alguna aplicación en especial, por ejemplo, son usados en simulaciones nucleares, investigaciones sobre predicción del clima, comportamiento de materiales peligrosos e investigaciones sobre biotecnología, entre otros.

Actualmente un cluster no solamente se puede usar de forma separada si no que a la vez puede formar parte de otro grupo de clusters para que todos en conjunto proporcionen un mayor rendimiento, a este concepto de agrupación de clusters se le llama Superclusters o Clusters GRID.

En nuestro país, los primeros en hacer investigaciones sobre clusters fueron investigadores de la UNAM del Departamento de Supercómputo en el año de 1997 y en la actualidad los clusters más importantes en México se encuentran en PEMEX, en el IMSS, en la empresa Elektra y forman parte de la lista de los 500 clusters más rápidos del mundo [13], aunque ya también diversas universidades y otros centros de investigación han construido sus propios clusters.

Existen diversas herramientas (software) para construir un cluster, pero en particular, para el objetivo de esta tesis se utiliza la herramienta llamada MOSIX. Por consiguiente, la estructura de la tesis presenta el siguiente orden.

En el capítulo 1, se da un panorama general de los sistemas distribuidos, de cómo están constituidos, características, ventajas, desventajas y aspectos que se deben tomar en cuenta en su diseño. La razón por la que se incluye un estudio sobre los sistemas operativos distribuidos, es porque un cluster también es un sistema distribuido.

En el capítulo 2, se amplía la información de manera más detallada sobre lo que es un cluster, su clasificación, las actividades más comunes que realiza, ventajas, desventajas, los aspectos que se deben tomar en cuenta al construir un cluster así como aplicaciones que pueden tener.

En el capítulo 3, se describen las herramientas (MOSIX, CONDOR y LUI), con las que se pueden construir un cluster, se da un panorama ligeramente detallado de cada una de ella, pero dando mayor énfasis en MOSIX, debido a que el objetivo de la tesis es utilizar esta herramienta para construir el cluster.

En el capítulo 4, se detalla exactamente cómo se instala y configura MOSIX en cada uno de los nodos del cluster, abarca las especificaciones del kernel de Linux, la instalación de MOSIX, su configuración y manejo una vez que se ha instalado.

En el capítulo 5, se describe en forma detallada la aplicación que se desarrolló y la distribución de procesos entre los nodos del cluster, obteniendo así un rendimiento, este comportamiento se describe gráficamente. También se muestran los resultados obtenidos en

cada uno de los nodos. Finalmente se muestran algunas imágenes obtenidas de la aplicación, la cual consiste en la simulación de un fenómeno óptico llamado “halo”.

El objetivo principal que se cubrió y las bases documentadas en los capítulos descritos en este documento fueron, la construcción de un cluster utilizando el sistema operativo Linux y la herramienta MOSIX.

CAPÍTULO 1

INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

1.1 INTRODUCCIÓN

Este capítulo tiene la finalidad de dar un panorama general de lo que son los sistemas distribuidos, sus características, ventajas, desventajas y aplicaciones, esto con la finalidad de entender algunos conceptos que son básicos para el desarrollo del cluster que se tiene como objetivo en esta tesis, debido a que también un cluster es un sistema distribuido.

Aproximadamente entre los años de 1980 y 1985 cuando comenzó la era de computadora moderna suceden dos grandes avances tecnológicos, el primer avance fue el desarrollo de poderosos microprocesadores de 8, 16, 32 y 64 bits, el segundo avance significativo fue la invención de las redes de área local (Local Area Networks) que permiten conectar un gran número de computadoras personales (PC´s) permitiendo transferir e intercambiar datos entre ellas en tiempos muy pequeños. Posteriormente aparecen las redes de área amplia **(***Wide Area Networks-WAN***),** las cuales permiten conectar o están constituidas por redes de área local.

Con el resultado de estas tecnologías, en la actualidad es posible reunir sistemas de cómputo compuestos por un gran número de PC's, interconectadas mediante una red de alta velocidad y mediante software especial. Estos sistemas reciben el nombre de ***sistemas distribuidos*** [1].

1.2 SISTEMAS DISTRIBUIDOS

Un sistema distribuido es una colección de computadoras independientes enlazadas por una red y que aparece ante los usuarios como una sola computadora [1].

Las tareas principales de un sistema distribuido son, coordinar actividades y compartir recursos entre las computadoras que conforman al sistema. Se pueden notar también dos aspectos importantes. Uno es el hardware: las máquinas son autónomas y el otro es el software: un sistema distribuido bien diseñado hace que el usuario perciba que sólo es una máquina.

1.2.1 Ventajas y Desventajas de los sistemas distribuidos

Las principales ventajas de los sistemas distribuidos con respecto a los sistemas centralizados son: economía, velocidad, distribución inherente, confiabilidad y crecimiento por incrementos.

En los sistemas distribuidos, los datos no es lo único que se puede compartir, también se comparten diferentes dispositivos periféricos. Aunado a esto se tienen las siguientes ventajas sobre las PC's independientes: datos compartidos, dispositivos compartidos, comunicación y flexibilidad.

Si bien es cierto que los sistemas distribuidos presentan una serie de ventajas también presentan algunas desventajas, algunas de ellas son las siguientes:

- ***El software,*** ya que a veces no es fácil saber algunas cosas tales como, los lenguajes apropiados para realizar un sistema operativo distribuido o que tanto le corresponde hacer el sistema y que tanto le corresponde hacer a un usuario, entre otros.

- ***Las redes,*** dado que un sistema distribuido está constituido por un grupo de computadoras interconectadas por una red, puede darse el caso de que la red se vea saturada, entonces, cuando un sistema operativo distribuido llega a depender de las características físicas de la red, la perdida o saturación de mensajes puede negar algunas de sus ventajas.

- ***La seguridad,*** otro problema que se presenta frecuentemente, es la falta de seguridad en los datos, debido a la facilidad que se tiene de poder compartirlos.

1.3 HARDWARE EN UN SISTEMA DISTRIBUIDO

Aunque todos los sistemas distribuidos constan de varias computadoras, existen diversas formas de organizarlas, en particular, la forma de interconectarlas y comunicarlas entre sí.

A través de los años se han propuesto diversas taxonomías de clasificación para sistemas distribuidos. Aunque ninguna ha sido adoptada de manera amplia, la taxonomía más citada y aceptada es la de Flynn (1972). Flynn eligió dos características consideradas esenciales: ***el número de flujo de instrucciones y el número de flujos de datos*** [1][3]. De este modo se tiene:

1. La arquitectura SISD (*Single Instruction, Single Data*), con un flujo de instrucciones y un flujo de datos, que en esencia es la arquitectura de Von Neumann. Se trata de todas las computadoras con un procesador y que procesan una sola instrucción a la vez.

2. La arquitectura SIMD (*Single Instruction, Multiple Data*), con un flujo de instrucciones y varios flujos de datos, se conoce como procesador de arreglos. Esta categoría se refiere a procesadores vectoriales, en los que una instrucción vectorial primero indica la operación que se va a realizar y posteriormente especifica la lista de operadores (denominada ***vector***) sobre los que operará.

3. La arquitectura MISD (*Múltiple Instruction, Single Data*), con un flujo de varias instrucciones y un flujo de datos. Ninguna de las computadoras conocidas se ajusta a este modelo, no tiene aplicaciones industriales.

4. Por último, la arquitectura MIMD (*Multiple Instruction, Multiple Data*), que es un verdadero procesador paralelo, con múltiples instrucciones y múltiples flujos de datos, que significa un grupo de computadoras independientes, cada una con su propio contador de programa, su propia memoria, sistema operativo y sus propios datos. Todos los sistemas distribuidos son MIMD. Por lo que se profundizará más en ésta categoría.

En la figura 1.3-1 se muestra una división de las computadoras MIMD en dos grupos; aquellas que tienen memoria compartida, que por lo general se llaman ***multiprocesadores*** y aquellas que no tienen memoria compartida, que reciben el nombre de ***multicomputadoras***.

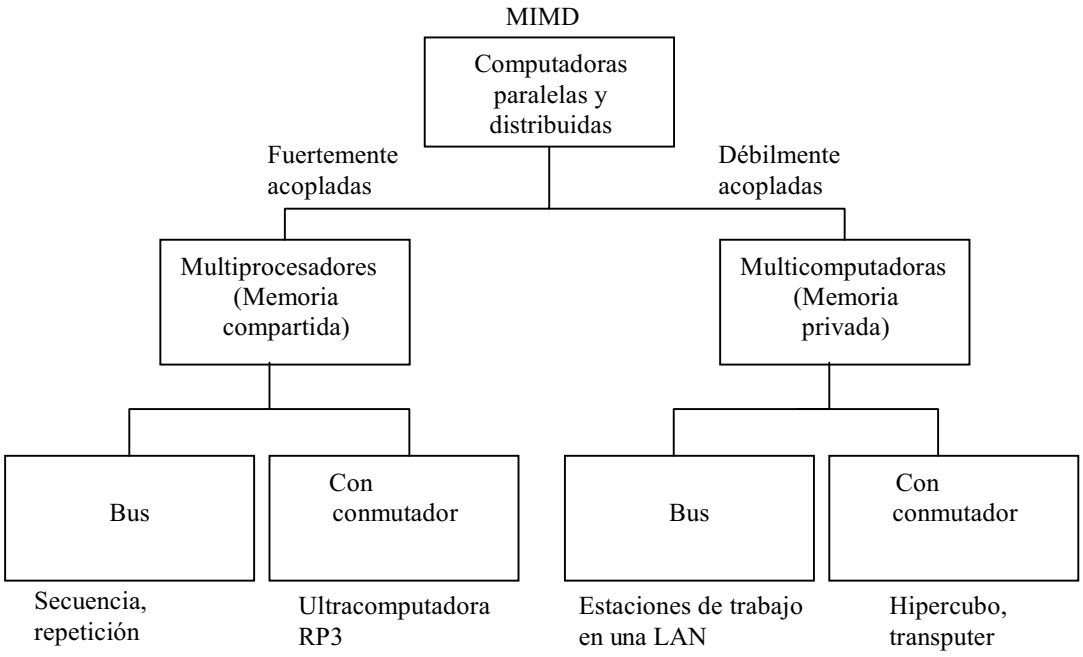


Figura 1.3-1 Una taxonomía de los sistemas de cómputo paralelo y distribuido.

1.3.1 Sistemas fuertemente y débilmente acoplados

En ciertos sistemas, con respecto al hardware, se dice que las máquinas están ***fuertemente acopladas*** y en otras están ***débilmente acopladas*** [3].

En sistemas fuertemente acoplados se utiliza un almacenamiento único compartido por los diferentes procesadores y un sólo sistema operativo que controla todos los procesadores y el hardware del sistema, como se muestra en la figura 1.3-2. La comunicación se realiza mediante memoria compartida y el retraso que se experimenta al enviar un mensaje de una computadora a otra es corto y la tasa de transmisión de los datos, es decir, el número de bits por segundo que se pueden transferir es grande.

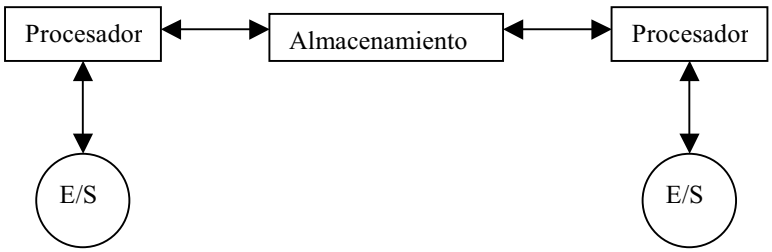


Figura 1.3-2 Sistema fuertemente acoplado.

Un sistema débilmente acoplado consiste en conectar dos o más computadoras independientes mediante una red, como se muestra en figura 1.3-3. Cada computadora tiene su propio sistema operativo, su propia memoria y su almacenamiento, además de que pueden funcionar en forma independiente y pueden comunicarse entre sí. También pueden tener acceso a los archivos de otras computadoras por medio de la red. La comunicación entre los procesadores se realiza por medio de ***transferencia de mensajes*** o de llamadas a ***procedimientos remotos***. El retraso de las máquinas es grande y la tasa de transmisión de los datos es baja.

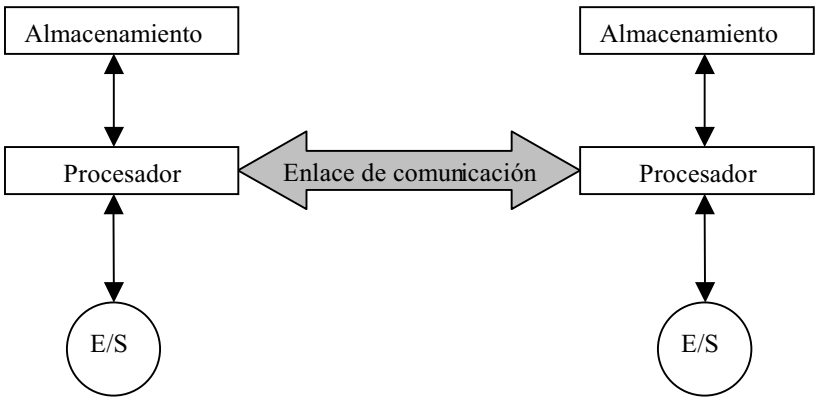


Figura 1.3-3 Sistema débilmente acoplado.

Los sistemas fuertemente acoplados tienden a utilizarse más como sistemas paralelos (para trabajar en un problema en específico) y los débilmente acoplados tienden a utilizarse como sistemas distribuidos (para trabajar con varios problemas relacionados o no relacionados entre sí).

En general los multiprocesadores tienden a estar más fuertemente acoplados que las multicomputadoras, puesto que comparten la memoria, un reloj global y sus tiempos de acceso a los datos son similares para todos sus procesadores. Por lo tanto, el intercambio de datos se realiza a una velocidad semejante a la velocidad de acceso a memoria, aunque

algunas multicomputadoras interconectadas mediante fibra óptica pueden funcionar también con velocidad aproximada a sus memorias, la ejecución de algunas tareas puede ser más tardado, debido a que no se comparte ni memoria, ni reloj y cada computadora cuenta con su memoria local. Los términos "fuertemente acoplados" y "débilmente acoplados" no son estrictamente apropiados, pero sí son muy útiles para entender el concepto que se quiere dar a conocer.

1.3.2 Organización simétrica

Aunque el multiprocesamiento simétrico es la organización más compleja al llevarla a la práctica, suele ser muy poderosa. Todos los procesadores son idénticos y el sistema operativo puede utilizar cualquiera de ellos para controlar cualquier dispositivo de E/S o para hacer referencia a cualquier unidad de almacenamiento.

Los sistemas de multiprocesamiento simétrico generalmente son los más confiables ya que si existe alguna falla en un procesador hace que el sistema operativo expulse el procesador de su conjunto de procesadores disponibles y tome nota del problema. El sistema puede seguir funcionando aunque a niveles menores (es decir, existe una degradación paulatina).

1.4 SOFTWARE EN LOS SISTEMAS DISTRIBUIDOS

Si bien es cierto que el hardware es importante, muchos expertos opinan que el software lo es mucho más. La imagen que presenta y la forma de pensar de los usuarios del sistema, queda determinada en gran medida por el software y no por el hardware [1].

Los sistemas operativos no se pueden clasificar tan fácilmente como el hardware. Por su propia naturaleza el software es vago y amorfo. Pero aún así, es más o menos posible distinguir los tipos de sistemas operativos para los varios CPU's conectados en la red: los ***débilmente acoplados*** y los ***fuertemente acoplados***.

El software débilmente acoplado permite que las máquinas y los usuarios de un sistema distribuido sean independientes entre sí en lo fundamental, pero que puedan interactuar en cierto grado cuando sea necesario.

Por otro lado, tenemos el caso de un multiprocesador (software fuertemente acoplado), el cual, es un arreglo de procesadores que comparten la memoria, comparten el mismo sistema operativo, todos los procesadores tienen un reloj común y el tiempo de acceso a los datos es el mismo por todos los procesadores.

1.5 ASPECTOS PRINCIPALES EN EL DISEÑO DE UN SISTEMA DISTRIBUIDO

Para el diseño de un sistema operativo distribuido existen ciertos aspectos que son fundamentales y que deben ser tomados en cuenta.

1.5.1 Transparencia

Una de las principales metas del un sistema distribuido es la transparencia, es decir, la manera de hacer que las múltiples computadoras interconectadas que forman un sistema distribuido aparezcan ante los usuarios como si fuese sólo una máquina. Un sistema que logre este objetivo se conoce como transparente [1].

La transparencia en un sistema distribuido es una tarea difícil de lograr, por esta razón se considera necesario también tomar en cuenta los siguientes aspectos: transparencia de localización, transparencia de migración, transparencia de réplica y transparencia de concurrencia.

1.5.2 Heterogeneidad

Debido a que las computadoras que constituyen un sistema distribuido pueden ser máquinas heterogéneas, es decir, computadoras de diferentes arquitecturas, debido a esta razón, se debe evitar en al máximo la dependencia entre un sistema operativo y la plataforma del hardware.

1.5.3 Flexibilidad

La flexibilidad es una de las características más importantes tanto en los sistemas distribuidos como en cualquier otro sistema ya que por características propias un software nunca es perfecto. Por tal motivo existen dos razones principales para que un software sea flexible:

- Que pueda modificarse fácilmente, causando los mínimos cambios posibles en todo el sistema, ya que en ocasiones se encuentran errores (bugs) en el diseño o los usuarios tienen nuevos requerimientos.

- Que pueda mejorarse, para hacerlo cada vez más robusto, más fácil de usar y la forma de hacerlo es mantener abiertas las opciones de poder corregir.

1.5.4 Confiabilidad

Uno de los objetivos originales de la construcción de sistemas distribuidos es hacerlos más confiables que los sistemas con un procesador, debido a la vulnerabilidad que existe entre las diferentes máquinas remotas que forman el sistema distribuido.

Un sistema ampliamente confiable debe ser siempre muy disponible, debido a que los datos confiados al sistema no deben perderse o mezclarse de manera alguna.

Otro aspecto importante en un sistema confiable, es la seguridad en los archivos y otros recursos que deben ser protegidos contra el uso no autorizado.

1.5.5 Desempeño

La construcción de un sistema distribuido transparente, flexible y confiable no será bueno si es muy lento. En particular, cuando se ejecuta una aplicación en un sistema distribuido no debe parecer peor que su ejecución en un sistema centralizado, al menos debe ser igual.

La métrica estándar para medir el desempeño de un sistema es el tiempo de respuesta mediante la ejecución de un determinado número de trabajos y la capacidad de la red consumida.

Una de las desventajas de un sistema distribuido con respecto a un sistema de un sólo procesador, es el tiempo consumido en el envío y recepción de mensajes a través de la red entre las diferentes máquinas.

En particular, los trabajos que necesitan de un gran número de pequeños cálculos y que necesitan interactuar en gran medida con otros, en general son la causa de algunos problemas en los sistemas distribuidos con una comunicación lenta.

1.5.6 Escalabilidad

En un sistema distribuido, la escalabilidad se refiere a la capacidad del sistema para adaptarse a la creciente demanda de conectar más computadoras si así se requiere. La escalabilidad es algo natural e inevitable debido a que las necesidades de los usuarios crecen con el paso del tiempo, por eso al diseñar un sistema distribuido se debe tomar en cuenta este aspecto para evitar problemas de comunicación o pérdida de información.

1.5.7 Tolerancia a fallas

Una de las características más importantes de los sistemas operativos distribuidos es la capacidad para tolerar fallas en los procesadores individuales y continuar trabajando [1].

Cuando un sistema distribuido es tolerante a fallas significa que puede seguir trabajando aun cuando fallen algunas partes del sistema.
Los diseños tolerantes a fallas son importantes sobre todo en *sistemas críticos* en los que quizá no puedan intervenir los seres humanos para solucionar los problemas, también en sistemas en donde las consecuencias de las fallas son demasiado graves que podrían sobrevenir tan rápido que los seres humanos no podrían intervenir a tiempo.

Existen algunas técnicas para fomentar la tolerancia a fallas:

- Se deben mantener varias copias de los datos críticos para el sistema y para los diferentes procesos.

- El sistema operativo debe estar diseñado de forma tal que pueda trabajar de manera eficiente con la configuración máxima del hardware y con subconjuntos del hardware cuando ocurran fallas.

- Las funciones de detección y corrección de errores en el hardware deben estar implantadas de tal forma que se efectúen comprobaciones exhaustivas.

- Se debe aprovechar la capacidad de los procesadores ociosos para tratar de detectar fallas potenciales antes de que ocurran.

CAPÍTULO 2

CLUSTERS

2.1 INTRODUCCIÓN

El objetivo de éste capítulo es dar una descripción de los clusters, lenguajes y librerías de programación mas usados, clasificación y aplicaciones, todo esto con la finalidad de ir comprendiendo y familiarizándose con la forma o manera de cómo está constituido y cómo funciona un cluster.

En la actualidad, los clusters han tenido mucho auge en centros de investigación y en las empresas, debido a que ciertos problemas que se desean resolver rebasan la capacidad de cómputo de una computadora personal. La solución más obvia para resolver estos problemas pues sería comprar una supercomputadora, pero existe un problema a esta solución debido a que una supercomputadora cuestan en ocasiones varios millones de dólares, cantidad que a veces va más allá de los presupuestos de inversión tanto de las empresas como de los centros de investigación.

Pero la necesidad de solucionar los problemas con los recursos que se cuentan, han hecho que personal académico de diversas universidades y centros de investigación se han a la tarea de construir su propias supercomputadoras conectando computadoras personales y desarrollando software para resolver dichos problemas

El primer cluster (llamado *Beowulf*) que se construyó fue en el año de 1994, en el Centro de Vuelos Espaciales Goddard de la NASA, con la finalidad de resolver problemas que aparecen en las ciencias de la Tierra y el Espacio.

Posteriormente en el año de 1996, se construye otro cluster llamado *Hyglac* desarrollado por el Instituto Tecnológico de California (CalTech) y el Laboratorio de Propulsión Jet (JPL), el otro cluster es, *Loki* construido en el Laboratorio Nacional de Los Álamos.

2.2 ESTRUCTURA DE UN CLUSTER

Antes de continuar con este capítulo se dará una definición de un cluster.

Un cluster es la interconexión de dos o más computadoras independientes a través de una red, usadas como un recurso unificado de cómputo con el fin de aumentar el rendimiento en la ejecución de tareas [4].

De todos los nodos que forman parte del cluster, alguno de ellos tiene que actuar como servidor debido a que debe existir algún nodo que asigne tareas y administre los procesos internos de paso de mensajes en el cluster.

La arquitectura del cluster es uno de los diversos caminos para explotar el procesamiento en paralelo. Existen diferentes métodos importantes que se utilizan para realizar cómputo paralelo entre los cuales se encuentran: Multiprocesamiento Simétrico (*Simetric Multiprocessing-SMP*), Acceso de Memoria No Uniforme (*Nonuniform Memory Acces-NUMA*), Procesamiento Masivamente Paralelo (*Massively Parallel Processing-MPP*), entre otros. Los sistemas Tolerantes a Tallas (*Fault Tolerant-FT*) explotan el procesamiento en paralelo para conseguir una buena integridad entre los diferentes componentes que forman el cluster.

Cada diseño de estos clusters tiene como fin mejorar el desempeño, la integridad de los sistemas o ambas cosas. En la figura 2.2-1 se resumen las relaciones de las diferentes alternativas de diseño en un cluster, desempeño, integridad y escalabilidad.

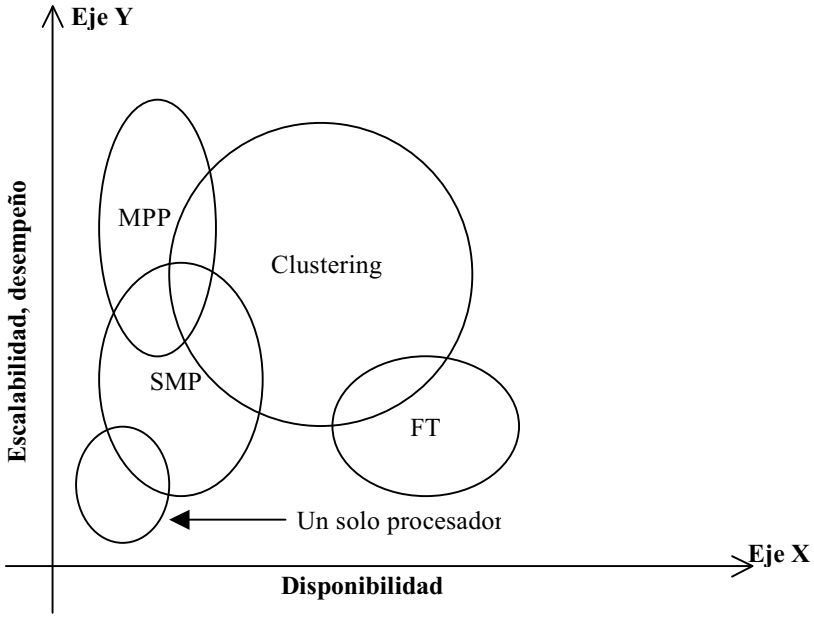


Figura 2.2-1 Escalabilidad, desempeño contra Disponibilidad.

En resumen, un cluster es un grupo de computadoras interconectadas para proveer rapidez e integridad de servicio.

2.2.1 Ventajas y desventajas de los clusters

Las ventajas que presenta la construcción de un cluster son:

1. Cada una de las máquinas en un cluster, puede ser un sistema completo para usarlo en un amplio rango de aplicaciones.

2. El hardware de interconexión de red ha venido experimentando un constante decremento de precio, considerando que además se pueden lograr ahorros adicionales empleando un solo monitor, teclado y ratón.

3. Los clusters de computadoras pueden crecer hasta formar sistemas verdaderamente grandes, es decir, desde dos hasta varios cientos, lo cual no es posible con los sistemas SMP.

4. Se puede reemplazar fácilmente una computadora del cluster que no esté funcionando correctamente.

5. Si algún componente falla, el o los procesos pueden seguir ejecutándose en los demás nodos.

6. El cluster se puede interconectar a una red de área local permitiendo dar servicio a múltiples usuarios internos y externos a través de Internet.

7. Existe en la red software gratis como el sistema operativo LINUX y software de aplicación para clusters como: MOSIX, CONDOR, LUI, BEOWULF, HPVM entre otros.

Las desventajas principales en la utilización de un cluster son:

1. Si falla el nodo maestro se pierde la ejecución de toda la tarea.

2. Puede haber inconsistencia en los datos, provocada por alguna falla en el software o en el hardware.

Una pregunta puede surgir entonces, si el software es gratis, el hardware barato y los clusters pueden crecer de manera significante. ¿Por qué no cualquier institución o empresa tiene su propio cluster?

Pues bien la respuesta es que:

- No todo el hardware de red está diseñado para realizar procesamiento paralelo. Generalmente porque la latencia es alta y el ancho de banda de la red es baja.

- Hay muy poco software que soporte a un cluster como un sólo sistema.

2.3 CLASIFICACIÓN DE LOS CLUSTERS DE ACUERDO A SU HOMOGENEIDAD Y A SU TAMAÑO

Debido a que los clusters están formados por un grupo de computadoras y estas pueden tener diferente arquitectura, existe la siguiente clasificación:

2.3.1 De acuerdo a su homogeneidad

En los ***clusters homogéneos,*** todas las computadoras conectadas en red que forman los nodos del cluster tienen la misma arquitectura y utilizan un mismo sistema operativo (Linux, Unix, Windows), por ejemplo computadoras IBM, COMPAQ, HP, SUN, etc.

En los ***clusters heterogéneos,*** todas las computadoras conectadas en red que forman los nodos del cluster tienen diferentes arquitecturas y usan sistemas operativos diferentes. Por ejemplo utilizar supercomputadoras como SDSC, computadoras HP, computadoras IBM, computadoras COMPAQ y todas corriendo un sistema operativo diferente. Controlar este entorno de un cluster heterogéneo es mucho más difícil que controlar un cluster homogéneo [5].

2.3.2 De acuerdo a su tamaño

Los ***meta clusters,*** son clusters de clusters, es decir, los meta clusters son usualmente un grupo de clusters que están geográficamente distribuidos, ya sea a nivel nacional o alrededor del mundo, pero pueden ser tratados como un sólo recurso por software especial y muy avanzado. Los meta clusters pueden ser también homogéneos o heterogéneos. Los meta clusters son llamados también super clusters [10].

2.4 CARACTERISTICAS DE LOS CLUSTERS

Un cluster proporciona tres beneficios principales: alta disponibilidad, buena escalabilidad y bajo costo [7].

La ***alta disponibilidad***, es la capacidad de permanecer funcionando ante diferentes fallas que pudieran existir mientras se encuentra trabajando el cluster. Por ejemplo una falla podría ser un procesador estropeado, una falla en disco duro, alguna falla en la conexión de red o una falla de corriente en algún nodo. Entonces si un nodo falla, los demás nodos captan toda la carga de trabajo.

Para muchas aplicaciones de empresas, el cluster es una buena solución, debido a que el tiempo de recuperación es aceptable. Frecuentemente se programan los tiempos para realizar otras aplicaciones en algún nodo del cluster.

La ***escalabilidad,*** es la capacidad de un sistema o sistemas de poder crecer de acuerdo con las necesidades de los usuarios. Los clusters son escalables ya que se les puede agregar o sustraer componentes a la computadora, por ejemplo, agregar memoria, discos duros, tarjetas de red, procesadores o cambiar una computadora completamente.

Las arquitecturas de clusters pueden ser escaladas de dos maneras. La primera, los diseñadores seleccionan las computadoras que formarán los nodos del cluster, para que de esta forma puedan ser escaladas, aumentándoles memoria, disco duro, etc. La segunda, los diseñadores aumentan el número de computadoras que forman el cluster.

El ***bajo costo,*** en una arquitectura cluster es relativamente bajo debido a que un cluster con sistema operativo Linux provee una gran uniformidad en todos los nodos y como resultado de esto se puede instalar diverso software que la empresa necesite como son: ORACLE, SYBASE o INFORMIX, las cuales son bases de datos que pueden ser instaladas en un cluster sin ningún problema de compatibilidad con Linux, así también las empresas no tendrían que hacer un gasto extra para comprar diferente software para cada plataforma.

En resumen los clusters proveen una alta disponibilidad en relación a la optimización en el tiempo que tardan las aplicaciones y los bajos costos económicos, también es económico un cluster con sistema operativo LINUX debido a la facilidad con que se puede conseguir el software y la disponibilidad de técnicas en el manejo, debido a que es muy similar al sistema operativo UNIX.

2.5 ACTIVIDADES COMUNES EN UN CLUSTER

Las actividades esenciales de los clusters son las siguientes [5]:

El ***soporte de paso de mensajes,*** es la habilidad de pasar datos entre los diferentes procesos mediante algún método estandarizado. Esta comunicación entre los diferentes procesos permite resolver un problema en forma paralela.

La ***migración de procesos,*** es la habilidad para mover un proceso de un nodo a otro nodo del cluster sin tener que reiniciar el programa, de este modo se pueden balancear las cargas entre los diferentes nodos. La migración de procesos puede ser usada idealmente si la carga de una máquina es demasiado grande.

El ***balanceo de cargas,*** se refiere a la distribución de la carga de trabajo de manera equivalente entre los diferentes nodos del cluster. En un cluster se puede dar el caso de que algún nodo esté ocioso debido a que ha terminado su tarea, mientras que otros nodos pueden estar procesando su carga de trabajo todavía, entonces al tener nodos ociosos se hace nuevamente una reasignación de procesos.

La ***interfaz gráfica de usuario (GUI),*** un buen diseño de una interfaz gráfica, puede ayudar al usuario a interactuar o entender de manera más fácil como funciona el sistema. Si un usuario no entiende el sistema de manera intuitiva puede provocar que el sistema no se utilice. Una buena interfaz también implica algún método para monitorear el estado de trabajo de cada nodo y el estado completo del cluster.

Se pueden ***redirecionar entradas y salidas (I/O)*****,** en un cluster con sistema operativo Linux se tiene esta posibilidad. Es muy similar a Unix, por ejemplo utilizar (>, >>, <, <<). Esta característica permite a los usuarios el manejo o ejecución de tareas de manera más fácil.

2.6 LENGUAJES DE PROGRAMACIÓN MÁS UTILIZADOS EN CLUSTERS

Los lenguajes de programación más utilizados son: lenguaje C o Fortran con las librerías de ambiente paralelo (Interface de Paso de Mensajes-MPI, Maquina Virtual Parelela-PVM) que tienen funciones para envío y recepción de información hacia o desde cualquier nodo del sistema.

Cuando se tiene un cluster con la herramienta MOSIX, no es necesario usar las librerías PVM o MPI, sólo se debe programar en lenguaje C o Fortran dividiendo la tarea principal en varios procesos y MOSIX realiza la distribución automáticamente, o bien, si el usuario lo desea puede hacer la distribución de los procesos en forma manual.

Debido a que el objetivo de la tesis es construir un cluster con MOSIX, la aplicación que se desarrollará es en lenguaje C bajo Linux.

A pesar de que estos sistemas se dedican a reducir el tiempo de procesamiento de la información, hay que tener presente que no todos los problemas requieren de un procesamiento en ambientes paralelos, se debe tomar en cuenta: primero, tener la herramienta correcta, el algoritmo correcto y segundo, por increíble que parezca tener el problema correcto.

2.7 TECNOLOGÍA DE UNA ARQUITECTURA CLUSTER

La principal tecnología de un cluster es la interconexión entre las computadoras ordinariamente llamadas “nodos del cluster”, el software y el hardware adicional que es requerido para la comunicación entre los nodos interconectados.

La tecnología de interconexión fundamentalmente es responsable de coordinar las actividades de la red que está formada por los nodos, también de hacer que el cluster parezca ante los usuarios como una sola máquina y no como un conjunto de computadoras interconectadas por una red donde se encuentran instaladas una serie de herramientas de software.

Generalmente una red o un bus son dedicados para este propósito únicamente. Para que un cluster tenga un alto desempeño se requiere de una red con un ancho de banda amplio, frecuentemente en un rango de 100 Mbps o más, dependiendo de las aplicaciones. En consecuencia los requerimientos exactos de la red varían con respecto a las necesidades del usuario y objetivo del cluster. Por ejemplo una Red de Área local (*Local Area Network LAN*) es adecuada para interconectar un cluster. Esta red LAN debe ser privada, para los nodos que forman el cluster.

La interconexión entre los nodos del cluster en la red LAN, puede ser mediante un hub (concentrador) o mediante un switch, ya que estos dispositivos pueden trabajar en una red LAN a una velocidad de 100 Mbps o más.

En la figura 2.7-1, se muestra gráficamente la estructura de un cluster.

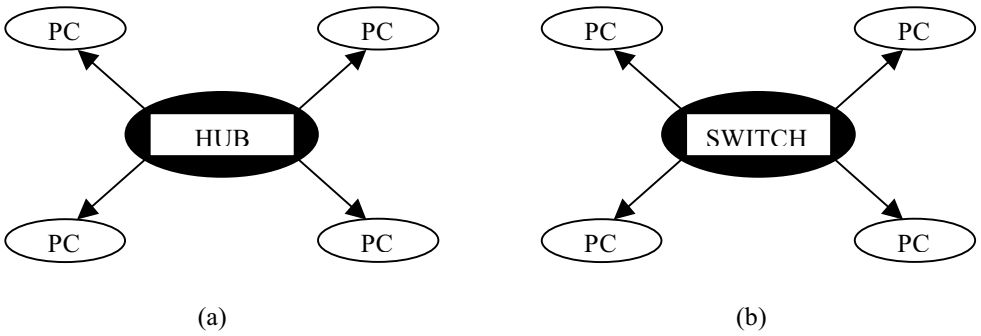


Figura 2.7-1: (a) Cluster utilizando un hub. **(b)** Cluster utilizando un switch.

Aunque un concentrador o un switch puede ser usado indistintamente para interconectar una LAN, es más recomendable un switch debido a que puede trabajar en modo full duplex, además de su desempeño y escalabilidad. En ocasiones un concentrador no es recomendable para interconectar los nodos de un cluster, sobre todo en los casos cuando el tráfico de información es demasiado grande, debido a que muchos concentradores trabajan en modo half duplex y no detectan colisiones entre la red, entonces este problema puede afectar en el rendimiento del cluster. Pero, en caso de que el cluster conste de pocos nodos, un concentrador también es muy aceptable.

2.7.1 Almacenamiento compartido y Almacenamiento distribuido

Cuando se diseña un cluster, también se diseña también la forma de almacenamiento de la información que puede ser: almacenamiento compartido y almacenamiento distribuido, como se muestra en la figura 2.7-2 [12].

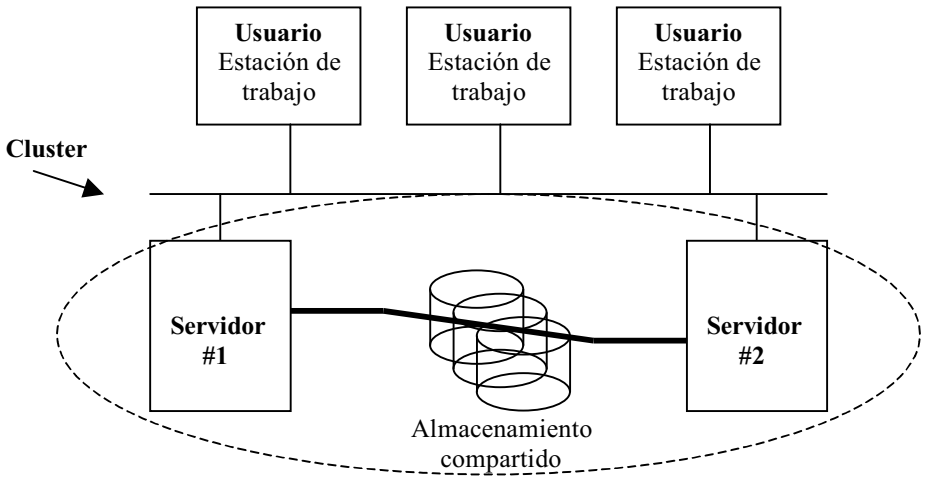


Figura 2.7-2 Cluster con almacenamiento compartido.

En una arquitectura de ***almacenamiento compartido***, la interconexión entre los nodos provee acceso a un disco o arreglo de discos, a éste arreglo de discos se le conoce convencionalmente como tecnología RAID. Los diferentes nodos que componen el cluster, comparten el espacio del arreglo de discos, obviamente se debe tener un software que controle el acceso a este arreglo, debido a que constantemente se lee y se escribe información por la ejecución de diferentes procesos en forma paralela. Este tipo de almacenamiento compartido, es mejor para sistemas que manejan gran cantidad de bases de datos y para sistemas que se encuentran en un mismo lugar.

En una arquitectura de ***almacenamiento distribuido***, cada nodo accesa a su propio disco duro. Cuando un nodo del cluster necesita alguna información que se encuentra almacenada en otro nodo el acceso se realiza mediante un mecanismo de paso de mensajes. Este mecanismo de paso de mensajes es responsable también de sincronizar y actualizar toda la información en los diferentes nodos que forman el cluster. El almacenamiento distribuido tiene mejor sentido en sistemas que necesitan tener un acceso constante e independiente a ciertos datos y para sistemas que se encuentran ampliamente dispersos o para clusters con pocos nodos.

Como se puede ver las arquitecturas de los clusters son bastante flexibles, tanto es así, que se pueden mezclar ambos métodos de almacenamiento (compartido y distribuido), de acuerdo a las necesidades que se tengan.

2.7.2 Memoria compartida y Memoria distribuida

Cuando se usa ***memoria compartida*** en un cluster cada procesador tiene acceso a toda la memoria logrando así un gran desempeño debido a que hay un espacio de direccionamiento compartido y el acceso puede llevarse a cabo uniendo accesos de todos los nodos que componen el cluster a la memoria principal [6][12], como se muestra en la figura 2.7-3. Los clusters con memoria compartida son muy similares a las computadoras que realizan multiprocesamiento simétrico (SMP), donde múltiples procesadores comparten un mismo sistema operativo y memoria.

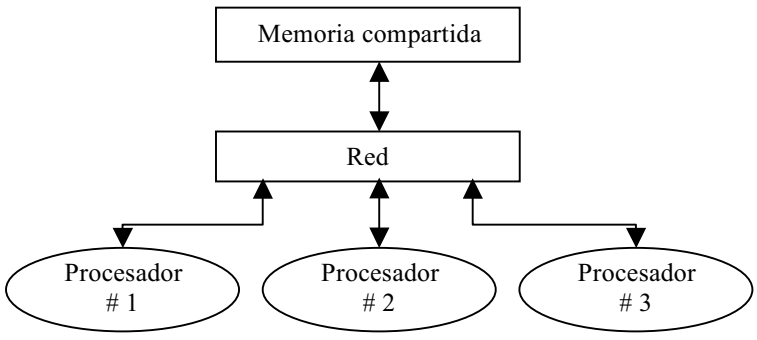


Figura 2.7-3 Memoria compartida.

Las ventajas y desventajas que presenta este tipo de memoria se muestran en la tabla 2.7-1

Tabla 2.7-1 Ventajas y desventajas de la memoria compartida.

Ventajas	Desventajas
• La facilidad de programación; es mucho más fácil programar en estos sistemas que en sistemas de memoria distribuida.	• El acceso simultáneo a memoria es un problema. • Todos los CPU's comparten el camino a memoria y el CPU que accesa la memoria, bloquea el acceso de todos los otros CPU's. En computadoras vectoriales. • Todos los CPU's tienen un camino libre a la memoria y existe interferencia entre CPU's.

Comúnmente la ***memoria distribuida*** se usa cuando la comunicación entre los nodos es establecida por *paso de mensajes* entre la red que forma el cluster, es decir, si un nodo requiere los datos contenidos en la memoria de otro procesador, deberá enviar un mensaje solicitándolos, figura 2.7-4

Estos sistemas tienen su propia memoria local y es favorecida cuando la aplicación es capaz de correr dentro de la memoria que tiene cada nodo.

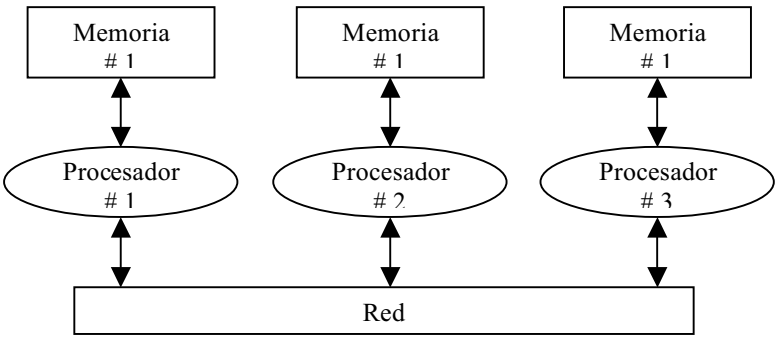


Figura 2.7-4 Memoria distribuida.

Las ventajas y desventajas que presenta este tipo de memoria se muestran en la tabla 2.7-2.

Tabla 2.7-2 Ventajas y desventajas de la memoria distribuida.

Ventajas	Desventajas
• Muy económico. • Escalabilidad; los sistemas con este tipo de memoria son fáciles de escalar, mientras la demanda de los recursos crece, se puede agregar más memoria y procesadores.	• El acceso remoto a memoria es lento. • La programación puede ser complicada.

2.7.3 Detección de problemas en los clusters

Los clusters tienen la habilidad de detectar problemas en un nodo y asignar el proceso del nodo dañado a otro nodo que se encuentre ocioso. Esta asignación puede hacerse automáticamente sin la intervención del operador. Pero también puede el operador tomar el control del proceso y reasignarlo a otro nodo disponible.

Lo fundamental en la detección de problemas, es la comunicación entre nodos, señalando que todos estén funcionando correctamente o que existe un problema en alguno de ellos. En este proceso de comunicación, existe un software que monitorea a todos los nodos del cluster, este software comúnmente es llamado latido (*heartbeat*). Normalmente con este software, cada nodo escucha que estén activos los demás nodos que forman parte del cluster

Cuando un nodo falla, en el caso más simple el operador es alertado. En un cluster más sofisticado el software reacciona ante el problema cambiando las aplicaciones rápidas y automáticamente a los nodos ociosos del cluster. Una falla catastrófica es que un nodo se dañe o se detenga por completo.
Los diseñadores de arquitecturas de clusters deben considerar los costos de cada una de las fallas más probables que pudieran presentarse, para preparar las contingencias apropiadamente.

En un cluster el tiempo de mantenimiento en un nodo se reduce de manera considerable, ya que el trabajo asignado al nodo al cual se dará mantenimiento puede cambiarse o puede asignarse a otro nodo y una vez terminado el mantenimiento, el trabajo es regresado nuevamente al nodo original.

2.8 APLICACIONES DE LOS CLUSTERS

Inicialmente los clusters se habían usado para aplicaciones de ciencia e ingeniería o como pruebas experimentales en la educación. Pero debido a su bajo costo, escalabilidad y generalidad se ha ampliado su campo de aplicación. Algunos ejemplos de nuevas aplicaciones se describen a continuación [4].

2.8.1 Bases de datos

Han aparecido diversas distribuciones de software comercial que realizan manipulación de información en bases de datos muy complejas. Las distribuciones comerciales de bases de datos más comunes son: Oracle, Sybase, Informix, las cuales han anunciado su compatibilidad y soporte con el sistema operativo Linux. Por ejemplo un cluster Beowulf presenta la oportunidad de soportar bases de datos distribuidas en las cuales se pueden realizar múltiples transacciones para múltiples usuarios.

2.8.2 Servidores Web

Actualmente uno de los crecimientos más rápidos ha sido el uso del World Wide Web y ha traído como consecuencia un incremento de búsquedas en algunos sitios, por lo que ha sido necesario también incrementar el ancho de banda, debido en parte a la gran cantidad de imágenes que son un componente esencial de los sitios Web. Con un apropiado software, un cluster puede dar un bajo costo y un alto desempeño a futuros servidores Web.

2.8.3 Simulación interactiva dinámica

La simulación interactiva dinámica (DIS) es una clase de aplicación que presenta a múltiples usuarios un entorno de simulación con una respuesta en tiempo real. DIS ha sido aplicada para simulaciones con armamento de guerra y en los videojuegos multiusuarios. DIS requiere de un escenario complejo de simulación, tiempo rápido de respuesta e interfaces sofisticadas.

2.8.4 Realidad virtual

Un escenario de realidad virtual puede ser generado explotando en un cluster el entorno paralelo. Diferentes partes de un escenario de realidad virtual pueden ser manipuladas y ejecutadas en nodos diferentes con cierta sincronización y al final el sistema puede entregar una imagen ensamblada.

2.8.5 Inteligencia artificial

En inteligencia artificial se puede tener cómputo muy intensivo. Como estructuras para controlar robots, resolver rompecabezas, jugar ajedrez, análisis de imágenes, entre otras. Los lenguajes dominantes para inteligencia artificial son LISP, Common LISP, Allegro Common LISP ya disponibles para el sistema operativo Linux.

2.9 AVANCES EN LA TECNOLOGÍA DE CLUSTERS

La tecnología de hardware interconectado continuará reforzándose, mejorando el ancho de banda de comunicación entre los nodos de un cluster y para conexiones de grandes distancias, incrementando así la velocidad entre las redes de computadoras. Por ejemplo una vez que hayan incrementado las conexiones de fibra óptica, incrementará la velocidad

de intercomunicación entre las computadoras y así también incrementará la rapidez de poder compartir datos y detectar fallas rápidamente.

Las tecnologías SMP y NUMA proporcionarán arquitecturas de clusters con nodos más potentes. Como se puede notar la extraordinaria fuerza de los clusters se debe en gran medida al número de nodos que se pueden conectar.

En conclusión, los avances futuros en la tecnología de los clusters será minimizar la pérdida de tiempo debido a que el tiempo juega un papel importante en algunas transacciones de algunas empresas.

CAPÍTULO 3

SOFTWARE PARA REALIZAR DISTRIBUCIÓN EN LINUX: MOSIX, CONDOR, LUI

3.1 INTRODUCCIÓN

Para construir un cluster existen muchas herramientas, cada una con cierta particularidad en la forma de trabajar, debido a esta razón el capítulo 3 tiene como finalidad, dar una descripción de algunas herramientas, para que un usuario en algún momento dado pueda evaluar y escoger la herramienta que más se acople a sus necesidades

Como un cluster requiere de un conjunto de herramientas (software) para realizar una distribución equitativa de la carga de trabajo entre sus nodos. Dentro del software desarrollado se encuentra: MOSIX, CONDOR y LUI, entre otros. Pero en particular se describirán estas tres herramientas, dando mayor énfasis en MOSIX debido a que esta herramienta se utilizó para construir el cluster que se tenía como objetivo.

La característica principal de MOSIX es distribuir procesos ya sea de forma automática o que el usuario realice de forma manual esta distribución, los procesos en un cluster MOSIX se ejecutan en forma simultánea, más no en forma paralela.

Los recientes avances en los clusters con la posibilidad de generar y migrar procesos a muchas máquinas ha permitido la integración y gestión de recursos de cómputo permitiendo así que los usuarios se beneficien de un mayor nivel de eficiencia, rentabilidad y de una forma rápida de obtención de los resultados.

En general, se puede decir que las tecnologías de cluster fueron inicialmente motivadas por la necesidad de tener pequeñas máquinas paralelas dedicadas a un sólo cálculo de gran escala.

3.2 SOFTWARE MOSIX

MOSIX es una herramienta diseñada para realizar balanceo de cargas en un cluster de forma totalmente transparente a tal grado que los nodos del cluster se comportan como si fuera solamente una máquina y de esta manera incrementar el aprovechamiento de cada uno de los nodos.

El núcleo de la herramienta MOSIX está formado por un conjunto de algoritmos diseñados para responder dinámicamente a las variaciones de carga entre los nodos que forman el cluster, migrando los procesos de un nodo a otro, con el fin de mejorar el desempeño.

Los usuarios pueden correr aplicaciones paralelas inicializando múltiples procesos en un nodo, entonces, la tarea de MOSIX es asignar estos procesos a los nodos con mejores recursos disponibles (procesador más rápido, más memoria RAM, mayor tamaño en disco duro, etc). Los algoritmos de MOSIX están diseñados para monitorear constantemente los nodos, para asignar y reasignar procesos, esto es fundamentalmente importante porque da un uso eficiente de todos los recursos con los que cuenta cada nodo [16].

La propiedad más notable de la ejecución de aplicaciones en MOSIX son las políticas de distribución de procesos y la flexibilidad de configuración de los nodos del cluster. Estas aplicaciones pueden ser ejecutadas creando muchos procesos de tal forma que parezca ante el usuario como si fuera solamente un nodo el que ejecuta todos los procesos [18].

3.2.1 Características de MOSIX

Las características de MOSIX que se muestran a continuación están implementadas a nivel del kernel del sistema operativo, en este caso de Linux [18].

- **Algoritmos de dispersión de información probabilística**, mediante estos algoritmos se tiene conocimiento suficiente de los recursos con que cuenta cada nodo. Además todos los nodos envían en cada determinado tiempo información a cerca de sus recursos disponibles en ese instante, para que de esta forma el nodo principal (*home node*) les pueda asignar o reasignar procesos. Al mismo tiempo cada nodo mantiene un buffer con información enviada a éste.

- **Migración de procesos preventivos**, esto significa que un usuario puede migrar procesos de forma manual a un nodo disponible, ya que la otra forma es mediante la asignación automática.

- **Balanceo de carga dinámica**, continuamente se intenta balancear y reducir la carga entre pares de nodos, es decir, cada vez que un nodo tiene un exceso de carga siempre se busca otro nodo que esté ocioso para asignarle trabajo. Este mismo algoritmo es ejecutado por cada uno de los nodos del cluster MOSIX.

- **Acomodo de memoria**, es un algoritmo que se activa cuando un nodo no tiene la memoria suficiente (se detecta una saturada paginación) para ejecutar los procesos que se le han asignado. En estos casos se activa también el algoritmo de balanceo de carga dinámica para asignar los procesos a otros nodos.

- **Comunicación eficiente de kernel,** esta característica fue especialmente diseñada para reducir la sobrecarga de comunicación (migración de procesos) entre los diferentes núcleos (kernel) de todos los nodos del cluster. Con esta característica la asignación e intercomunicación de procesos se hace más eficiente.

- **Autonomía y control descentralizado**, cada nodo es capaz de operar de manera independiente, de esta forma se permite una configuración dinámica donde los nodos pueden entrar o salir de la red cuando existe alguna interrupción.

- **Consideraciones de escalabilidad**, consiste en asegurarse que un sistema que funciona perfectamente en configuraciones pequeñas, funcione también para configuraciones grandes (clusters con un gran número de nodos).

La combinación de características hace que los programas no necesiten saber el estado actual del sistema de configuración para poder ejecutarse.

3.2.2 Tecnología de MOSIX

La tecnología de MOSIX consiste de un mecanismo llamado Migración de Procesos Preventivos (*Preemptive Process Migration-PPM*) y una serie de algoritmos para compartir recursos. Ambas partes son implementadas en el nivel de kernel usando módulos cargables [16].

La PPM puede migrar algunos procesos a un nodo disponible en un determinado instante de tiempo. Usualmente la migración automática de procesos, es basada en información proporcionada por los algoritmos de compartición de recursos, aunque los usuarios pueden también tomar el control de la migración de procesos manualmente y de esta forma la asignación de procesos queda a criterio del usuario.

La migración manual de procesos puede ser útil para implementar políticas propias de los usuarios. Al entrar al sistema como super usuario se tienen privilegios a tal grado que se puede hacer caso omiso al mecanismo de migración de procesos y el usuario puede decidir que nodos utilizar de acuerdo a sus necesidades.

Cada proceso tiene un único nodo principal (*Unique Home Node-UHN*) donde es creado y es asignado. Normalmente este nodo es donde el usuario se autentifica para entrar al sistema. Los procesos que son migrados usan los recursos propios disponibles de los nodos pero la interacción con el usuario siempre será a través del nodo principal (UHN), por ejemplo, si el usuario ejecuta el comando "*ps*" en el UHN, el sistema reportará todos los procesos que se están ejecutando tanto en el nodo principal como en todos los nodos.

El mecanismo de MPP, es la principal herramienta para el manejo de recursos en los nodos. Cuando los requerimientos de los procesos exceden los recursos de algún nodo, entonces, en este momento es cuando son migrados a otros nodos para balancear el trabajo ya que la meta principal es optimizar al máximo las tareas, los recursos de los nodos y los recursos de la red.

3.2.3 Algoritmos para compartir recursos

La función principal de los algoritmos para compartir recursos en MOSIX, es balancear la carga, es decir, constantemente intentan reducir la carga en un nodo. Este procedimiento se dice que es descentralizado, debido a que todos los nodos del cluster ejecutan los mismos algoritmos de manera independiente, el número de procesadores (por ejemplo, máquinas duales) en cada nodo y su velocidad son factores importantes para estos algoritmos de balanceo de carga.

Existen también algoritmos de compartición de memoria. Estos algoritmos son activados cuando un nodo no tiene la suficiente memoria para ejecutar los procesos que se le han asignado. Cuando esto sucede, los algoritmos de balanceo de carga son anulados y los algoritmos de compartición de memoria entran en acción de manera que reasignen el exceso de carga hacia otro nodo que tenga mayor cantidad de memoria [17].

3.2.4 Migración de procesos

Debido a los recientes avances en los clusters, a la posibilidad de migrar procesos hacia los diferentes nodos y a la necesidad de ejecutar aplicaciones que necesitan mucho tiempo de cómputo, se han desarrollado diferentes mecanismos de sistemas de archivos para poder controlar y sincronizar la migración de los procesos que son generados [15][17].

La principal característica que se debe tener en cuenta para diseñar un sistema de archivos, es la capacidad de migrar procesos y esta capacidad es soportada por MOSIX para Linux. MOSIX tiene implementado un sistema de archivos llamado MFS, el cual se describirá con más detalle posteriormente.

El mecanismo principal que usa MOSIX para la migración de procesos, como se mencionó anteriormente se llama: Migración de Procesos Preventivos. Para implementar el PPM, el proceso de migración es dividido en dos partes:

1. El ***contexto del usuario,*** llamado también ***remoto*** que contiene el código del programa, datos, mapas de la memoria, la pila y registros de los procesos.

2. El ***contexto del sistema,*** llamado también ***agente*** que contiene la descripción de los recursos de los nodos donde se ejecutan los procesos. El agente encapsula los procesos que están corriendo en el kernel y mantiene ciertos datos de ubicación de éstos en el UHN. Mientras los procesos pueden migrar entre los diferentes nodos, el agente nunca es migrado y se mantiene siempre en el UHN.

La interface entre el contexto del usuario y contexto del sistema está bien definida. Por lo tanto, es posible interceptar cada interacción entre los dos contextos. Esta interacción se encuentra implementada en la capa de enlace con un canal especial de comunicación. La figura 3.2-1 muestra un escenario de la comunicación entre dos procesos que comparten un UHN. En la parte izquierda se muestra un proceso local en Linux mientras que en el lado derecho se muestra un proceso remoto.

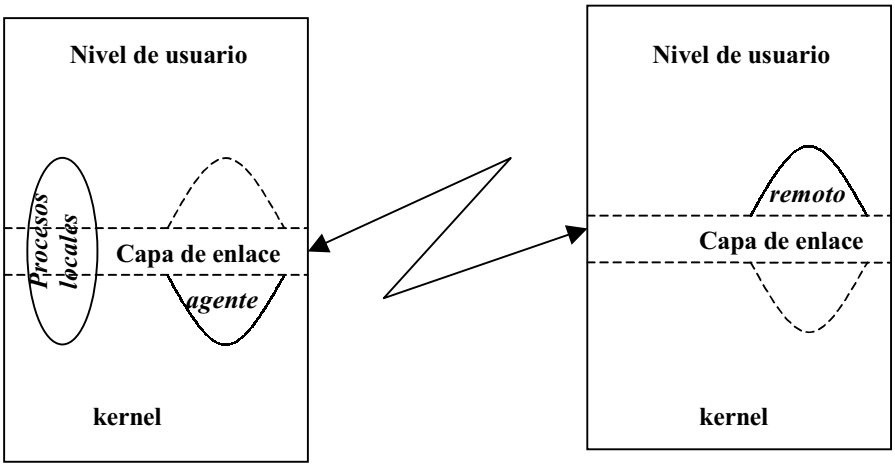


Figura 3.2-1 Un proceso local y un proceso migrado.

3.2.5 Acceso Directo al Sistema de Archivos

El Acceso Directo al Sistema de Archivos (*Direct File System Acces-DFSA*), es un mecanismo diseñado para reducir la sobrecarga de entradas y salidas (I/O) de los procesos migrados, por lo que se permite un mayor número de llamadas al sistema, es decir, se tiene un mejor control de un mayor número de procesos [15].

MOSIX, es un esquema distribuido que permite el acceso simultáneo de procesos a diferentes archivos en diferentes nodos, así que para que funcione correctamente el DFSA se requiere que todo su sistema de archivos (incluyendo ligas simbólicas) en todos los nodos sean idénticos y de preferencia sean montados en la misma partición.

En resumen, se puede decir que el DFSA optimiza y controla la intercomunicación de un mayor número de procesos, de tal manera que siempre exista una buena consistencia.

3.2.6 Sistemas de archivos MOSIX

Para que el DFSA pueda funcionar se ha implementado un Sistema de Archivos para MOSIX (*MOSIX File System-MFS*), el cual tiene como función principal proporcionar una vista global de todos los sistemas de archivos en todos los nodos del cluster y tratarlos como si fuera un sólo sistema, por ejemplo, si el MFS es montado en el punto ***/mfs,*** entonces, ***/mfs/1456/tmp/myfile*** se refiere al archivo de nombre ***/usr/tmp/myfile*** en el nodo ***#1456***. Por lo tanto, el MFS trataría de forma genérica en todos los nodos a partir de ***/usr/tmp.***

3.3 INTRODUCCIÓN DE CONDOR

Otra herramienta para distribuir cargas en un cluster se llama Condor y es desarrollado por un equipo de la Universidad de Winsconsin–Madison y fue instalado por primera vez en el departamento de Computación de esta Universidad [21].

Condor administra la información mediante un mecanismo de colas, mediante esquemas de prioridad de los trabajos y mediante la clasificación de recursos en los nodos que forman el cluster. Cuando se ejecuta algún trabajo coloca el trabajo en una cola de espera, posteriormente el trabajo se envía a un nodo para su ejecución y al final entrega los resultados al usuario.

3.3.1 Software Condor

Condor es un software que proporciona gran fuerza de cómputo (*High-Throughput Computing-HTC*) en todos los nodos que forman el cluster.

Cuando un usuario ejecuta un trabajo en el cluster, Condor lo primero que hace es encontrar la mejor máquina disponible en ese instante para ejecutar el trabajo. Condor también tiene la capacidad de detectar que computadora del cluster está ejecutando cierto trabajo, por lo que el usuario tiene la capacidad de mover (migrar) los trabajos a diferentes máquinas ociosas en caso de que sea necesario.

Dado que Condor puede migrar procesos entre los diferentes nodos, esta migración hace que el trabajo en general pueda ejecutarse en un mayor número de nodos, logrando así que los trabajos se ejecuten de manera más rápida.

Condor puede trabajar en tiempo real debido a que los trabajos se pueden ejecutar en diferentes máquinas al mismo tiempo y quizás con diferentes datos de entrada.

Condor tiene un nodo principal donde se pueden controlar todos los nodos. Por lo tanto, no se requiere de una cuenta en cada una de los nodos donde se está ejecutando algún trabajo, debido a que la comunicación se hace mediante una tecnología de llamadas remotas al sistema y esta comunicación es realizada con todos los nodos del cluster a través del nodo principal.

Todos los nodos ponen a disposición sus recursos, memoria RAM, tipo de CPU, velocidad de CPU, tamaño de la memoria virtual, localización física del nodo y la carga de trabajo actual de cada nodo.

3.3.2 Características de Condor

- **Punto de inspección y migración**, los programas que son ejecutados en un cluster Condor y que fueron migrados a una máquina en específico son monitoreados constantemente para que de esta manera el usuario se asegure que el trabajo se está realizando correctamente.

- **Llamadas remotas al sistema**, a pesar de que los trabajos se ejecutan en máquinas remotas, Condor preserva todas las llamadas que se han hecho a los diferentes nodos del cluster. Así los usuarios no tienen que preocuparse en que máquina se encuentra ejecutando su trabajo, para ellos aparecerá como si se estuviese ejecutando en la máquina donde él inicialmente lo ejecutó (nodo principal).

- **No es necesario que usuario tenga que modificar su código fuente**, debido a que el punto de inspección y migración le presenta al usuario la facilidad de que al momento de compilar los programas, estos se ligan con algunas librerías especiales de Condor, no se necesita programación especial, así que los programas podrán ser ejecutados en el cluster sin ningún problema.

- **Los trabajos pueden ser ordenados**, el orden de ejecución de los trabajos puede ser manipulado por el usuario. El conjunto de procesos que se ejecutan son mostrados en una gráfica y cada proceso mostrado corresponde a un nodo. Por lo tanto, los trabajos enviados a ejecución seguirán la secuencia mostrada en dicha gráfica.

- **ClassAds**, es un mecanismo flexible en Condor que ofrece al usuario una relación entre los recursos requeridos por el usuario para ejecutar alguna tarea y los recursos ofrecidos por cada uno de los nodos del cluster Condor, por ejemplo, si un usuario requiere que su trabajo se ejecute de preferencia en una máquina que tenga 128 Megabytes en memoria RAM, entonces el usuario asigna el trabajo a la mejor máquina disponible en ese momento.

3.3.3 Ejecución de un programa en Condor

Los lenguajes de programación usados en Condor son Fortran, Lenguaje C, y el usuario puede usar también las librerías PVM y MPI para realizar la programación en forma paralela.

Los pasos para ejecutar un programa en un cluster Condor son los siguientes:

1. **Preparación de código**, cuando un trabajo se ejecuta en un cluster Condor, el usuario define la forma en como quiere obtener su resultado, es decir, tal vez el usuario quiera que el resultado se le muestre en consola o bien que el resultado se guarde en algún archivo en especial, también puede definirse que la ejecución se lleve a cabo en segundo plano. En conclusión, el usuario puede definir todo en el programa.

2. **El universo de Condor,** para correr un programa en Condor se necesita especificar un universo (es una manera de indicarle a Condor la forma de como ejecutar un programa). En Condor existen cuatro universos (Standar, Vanilla, PVM y Globos), pero los que más se utilizan son:

    - **El universo estándar**, permite al usuario ejecutar un programa, de tal forma que dicho usuario tiene toda la libertad de indicar de forma manual la

distribución de procesos en los nodos. Para usar éste universo estándar, es necesario ligar el programa con algunas librerías de Condor usando el comando "***condor compile***".

- **El universo vanilla,** proporciona la opción de correr un programa en el cluster, sin necesidad de que el usuario tenga que hacer la asignación de los procesos manualmente, debido a que dicha asignación se hace de manera automática.

3. **Archivo de descripción propuesto,** el usuario define como será ejecutado el programa, por ejemplo, cuantas veces Condor tiene que ejecutar el mismo programa ya sea con los mismos datos de entrada o con datos totalmente diferentes, según las necesidades del usuario y puede indicarle también que al terminar todo su trabajo le envíe un correo al usuario.

4. **Enviar el trabajo**, una vez que los pasos anteriores han sido completados, se envía el programa a Condor para su ejecución mediante el comando*"**condor_submit**"*.

Cuando el programa ha terminado de ejecutarse en el cluster, Condor le indica al usuario el estado final del programa y algunas estadísticas acerca del desempeño o funcionamiento.

3.3.4 Limitaciones de Condor

Aunque Condor presenta varias características positivas, también tiene las siguientes limitaciones:

1. Condor no permite la intercomunicación de procesos como son: pipes, semáforos y memoria compartida.

2. La migración y asignación de los procesos debe llevarse en el menor tiempo posible, ya que por ejemplo, si se hace una llamada con un socket( ) y este permanece abierto por un largo periodo de tiempo, puede provocar que el proceso del punto de inspección y migración sea más lento.

3. Los procesos de alarmas, timers y sleeping no están permitidos en Condor, por la razón de que Condor maneja sus propias funciones como son: alarm( ), getitimer( ) y sleep( ).

4. Es preferible tener la misma distribución y versión de Linux instalada en todos los nodos.

5. Todos los archivos en Condor sólo permiten la opción de lectura o escritura pero no ambas opciones al mismo tiempo. Por razones de compatibilidad, si un archivo es abierto para lectura y escritura al mismo tiempo puede causar algún error.

6. Aunque no es necesario modificar el código de las aplicaciones para que puedan ser ejecutadas en el cluster, es preferible compilarlas y ligarlas con las librerías de

Condor para tomar las ventajas que ofrecen, tanto el punto de inspección y migración como el punto de llamadas remotas al sistema. El software comercial está excluido de tomar estas ventajas, debido a que las casas comerciales no ofrecen el código fuente, pero Condor ofrece otras herramientas para optimizar la ejecución este tipo de software.

3.3.5 Disponibilidad de Condor

Condor está disponible como software gratis en Internet vía World Wide Web [22]. Las distribuciones de Condor versión 6.x están disponibles para las siguientes plataformas que se muestran en la tabla 3.3-1.

Tabla 3.3-1 Arquitecturas aceptadas por Condor.

Arquitectura	Sistema Operativo
• Hewlett Packard PA-RISC(both PA70000 and PA8000 series)	• HPUX 10.20
• Sun SPARC Sun4m, Sun UltraSPARC	• Solaris 2.5.x, 2.6, 2.7, 2.8
• Silicon Graphics MIPS (R4400, R4600, R8000,R10000	• IRIX 6.5
• Intel x86	• RedHat Linux 5.2, 6.x, 7.1 • Solaris 2.5.x, 2.6, 2.7 • Windows NT 4.0
• Digital ALPHA	• OSF/1 (Digital Unix) 4.x to 5.0 • Linux 2.2.x

3.4 UTILIERÍA DE LINUX PARA INSTALAR UN CLUSTER (LUI)

Otra herramienta para hacer distribución de cargas se llama, Utilería de Linux para Instalar un Cluster (*Linux Utility for clusters Install-LUI*) [20]. LUI es un código fuente abierto (Open Source), y es desarrollado por IBM [19].

La herramienta LUI en un principio fue desarrollada para construir clusters con estaciones de trabajo RS/6000 SP de IBM. Pero en la actualidad LUI puede ser instalado en PC's con Sistema Operativo Linux (Red Hat), en estaciones de trabajo de IBM y en Servidores Dell.

En la clasificación de clusters existen: clusters homogéneos y clusters heterogéneos, tanto en software como en hardware. Con LUI se pueden construir clusters heterogéneos tanto en software como en hardware.

En este punto es necesario contestar a una pregunta que frecuentemente se hace un usuario: ¿Qué significa en LUI que un cluster sea heterogéneo, tanto en software como en hardware? o ¿Qué hace diferente un nodo de otro.? La respuesta son varios puntos y a continuación se mencionan:

- El disco puede ser SCSI, IDE O RAID.
- Puede tener diferente Kernel cada nodo.
- Algunos nodos requieren instrucciones especiales al momento de arrancar.
- Dispositivos de red diferentes.
- Diferentes paquetes de software instalados.
- Diferente tamaño en la partición swap.

Estos puntos que diferencían un nodo de otro, en LUI se les denomina “recursos”.

A toda la colección de nodos en el cluster, se les conoce como “máquinas objeto”, pero también se hace una distinción entre el servidor y los clientes, el servidor es llamado “Máquina Objeto Servidora”, y los clientes son llamados “Máquinas Objetos Cliente”. Por lo tanto, para construir un cluster LUI, todo lo que tiene que hacer un usuario es definir un servidor y definir una serie de nodos clientes.

3.4.1 Configuración de LUI

Existen dos formas de configurar LUI, una es por medio de la línea de comandos y la otra es mediante una interfaz gráfica (comúnmente conocida como GUI o GLUI para LUI).

Entonces para configurar un cluster LUI es necesario saber y conocer lo siguiente:

1. Todos los nodos clientes instalados, deben tener una buena comunicación con el nodo servidor.
2. Realizar la instalación de un servidor con Linux.
3. Obtener la distribución de LUI e instalarla en el servidor.
4. Escoger la forma de configurar los nodos (GLUI, Línea de comandos) y seguir paso a paso las instrucciones para la instalación.
5. Definir los nodos que funcionarán como clientes.
6. Personalizar todos los recursos para cada nodo (particiones de disco duro, etc).
7. Realizar la instalación por red de LUI en cada uno de los nodos.
8. Revisar que todos los nodos funcionen correctamente.

3.4.2 Formas de instalación de LUI

La instalación de LUI se puede realizar de dos maneras:

1. ***Desde disco***, el disco para inicializar los nodos se puede obtener del siguiente sitio de Internet [14].

2. ***Por red***, para este tipo de instalación se requiere que tanto el *firmware* de la tarjeta de red del nodo maestro coincida con el *firmware* de la tarjeta de red de los nodos clientes y que la intercomunicación entre todos los nodos funcione correctamente. El *firmware* que se usa frecuentemente se llama Ambiente de Ejecución de Preinicio (*Preboot eXecution Environment-PXE*), el cual es un estándar desarrollado por Intel.

La instalación que frecuentemente se usa en LUI es la instalación directa por red, para la cual, a continuación se muestran los pasos que se deben seguir [20]:

1. Debe haber una buena comunicación entre el servidor y los clientes.
2. El servidor está condicionado a escuchar (*broadcast*) a todos los clientes, mediante la red.
3. Los clientes están forzados a escuchar los mensajes del servidor, de esta manera saben los nodos en que momento hay algún mensaje de instalación.
4. La comunicación servidor-cliente se lleva a cabo por medio de direcciones IP.
5. El servidor envía el kernel a los nodos clientes mediante TFTP.
6. El kernel es leído en memoria por cada nodo cliente.
7. Una vez que el kernel ha sido cargado en cada uno de los nodos, se carga un sistema de archivos del servidor para obtener varios servicios de éste, mediante el NFS.
8. El kernel empieza a ser instalado en cada nodo.
9. Se crean las particiones de disco, se crean los archivos de sistema y se instala el sistema operativo en los clientes.
10. Los clientes completan la instalación en disco duro y están listos para inicializar por primera vez.

3.4.3 Instalación de un nodo LUI

Antes de realizar la instalación, es necesario definir de forma manual ciertos recursos en cada nodo. Para el funcionamiento de LUI se deben definir principalmente tres particiones en disco duro (boot, raíz, swap) y se debe hacer también una partición para instalar todos los paquetes RPM que contiene la distribución LUI.

Se tiene la posibilidad, de que LUI haga las particiones automáticamente, pero existe un problema, ya que se pueden hacer asignaciones muy grandes a las particiones de disco duro, lo que puede provocar que haya una mala distribución del tamaño.

LUI usa el comando *"**mklimm**"* para definir la máquina que funcionará como servidor

```
# mklimm –n hacker –t server -i 192.168.22.151 -m 255.255.255.0
```

la línea anterior se explica de la manera siguiente: al servidor se da el nombre de "hacker" y es de tipo "servidor" con una dirección IP 192.168.22.151 y una mascara 255.255.255.0

Después de que se define el servidor, se deben definir todos los nodos clientes usando también el comando *"**mklimm**"*.

```
# mklimm –n node1 –t client -i 192.168.22.152 -m 255.255.255.0
```

Una de las ventajas que se debe tener cuando se usa GUI para definir los nodos, es que solamente se necesitan cambiar los atributos diferentes para cada nodo.
LUI tiene una lista de comandos que proporcionan información acerca los nodos, el comando ***"lslimr"*** muestra la lista de recursos con los que cuenta cada nodo, ***"lslimm"*** muestra todos los nodos, ***"unallimr"*** se utiliza cuando el usuario desea hacer alguna

modificación de los recursos en algún nodo, cuando se necesita borrar algún recurso en específico se usa el comando **"*dellimr*"** o si se desea eliminar un nodo por completo se usa el comando **"*dellimm*"**.

3.4.4 Limitaciones de LUI

LUI solo ha sido probado para Servidores Netfinity de IBM, en estaciones de trabajo de IBM, IBM PowerPC, servidores Dell. La distribución de Linux con la que se ha probado es Red Hat 7.0, 7.1 y SuSE 7.1. Se espera adaptar LUI para otras distribuciones de Linux y para plataformas Alpha, entre otras.

3.5 COMPARACIÓN ENTRE MOSIX, CONDOR Y LUI

A continuación se muestra de manera resumida los requerimientos que se necesitan para la instalación de cada una de las herramientas que se han descrito anteriormente, de tal forma que el usuario pueda hacer un balance de acuerdo a los recursos con los que cuenta y en base a esto pueda decidir que herramienta le conviene usar.

Tabla 3.5-1 Tabla comparativa de MOSIX, CONDOR Y LUI.

Herramienta	Sistema Operativo	Arquitectura	Librerías especiales
MOSIX	• Red Hat 5.1, 6.0, 6.2, 7.0, 7.1 • SuSE 6.0, 6.1, 6.2, 6.3, 7.0, 7.1, 7.2, 7.3 • AT&T Unix system V release 2 • BSD/OS	• Alpha • Sparc • X86/Pentium • AMD	• No se necesitan librerías de ambiente paralelo, sólo se tiene que dividir la aplicación en varios procesos.

Tabla 3.5-1 Tabla comparativa de MOSIX, CONDOR Y LUI (continuación).

• CONDOR	• HPUX 10.20, • Solaris 2.5.x, 2.6, 2.7, 2.8 • RedHat Linux 5.2, 6.x, 7.1 • Windows NT 4.0 • OSF/1(Digital Unix) 4.x to 5.0 • Linux 2.2.x	• Hewlett Packard PA-RISC (both PA70000 and PA8000 series) • Sun SPARC Sun4m, Sun UltraSPARC • Silicon Graphics MIPS (R4400, R4600, R8000,R10000) • Intel x86 • Digital ALPHA	• PVM, MPI. • Los lenguajes de programación que se usan son, Fortran y Lenguaje C, C++.
• LUI	• Red Hat 7.0, 7.1 • SuSE 7.1	• Servidores Netfinity de IBM • Estaciones de trabajo RS/6000 SP de IBM • IBM PowerPC • X86, Servidores Dell	• MPI, PVM. MPICH. • Los lenguajes de programación son, Lenguaje C, C++

CAPÍTULO 4

INSTALACIÓN Y CONFIGURACIÓN DE MOSIX

4.1 INTRODUCCIÓN

En este capítulo se describe de manera detallada la instalación de MOSIX así como la tecnología MOSIX para la construcción de un grupo de computadoras (*Cluster Computing-CC*), con el objetivo de que algún usuario que desee construir y configurar un cluster con MOSIX tenga una referencia rápida y segura debido a que se incluyen algunos pasos que en los manuales los omiten, además de que no funcionan tal y como lo menciona la literatura MOSIX.

MOSIX trabaja de forma similar a un sistema de Procesamiento Multisimétrico (*Simetric Multiprocessing-SMP*), el cual es una modalidad del multiprocesamiento paralelo. Las aplicaciones se dividen en subprocesos que pueden ejecutarse de manera concurrente en cualquier procesador disponible. SMP mejora el rendimiento de la aplicación misma y el rendimiento total del sistema. Por lo tanto, el objetivo principal es incrementar el rápido procesamiento de grandes volúmenes de información y una rápida comunicación entre los diferentes procesos compartiendo memoria.

La tecnología *Cluster Computing* se basa en una colección de PC's, estaciones de trabajo o servidores con diferentes velocidades, capacidades de memoria incluso de diferentes generaciones. La parte esencial de la tecnología MOSIX es la capacidad de que varias computadoras puedan trabajar de manera cooperativa, como si fuese un sólo sistema.

MOSIX se considera también como una extensión del núcleo (kernel) de Linux una vez que ha sido agregado a éste. Una de sus características, a diferencia de otros clusters es que no es necesario modificar aplicaciones, ni tampoco utilizar librerías especiales para codificar algún programa. La idea principal es que después de la creación de un nuevo proceso, MOSIX lo asigna al mejor nodo posible en ese momento y estará monitoreando los procesos constantemente, para hacer una nueva asignación cuando sea necesario.

En MOSIX, muchas veces el usuario no necesita saber la manera automática de cómo se lleva a acabo la distribución de los procesos, pero también existe la posibilidad de que pueda manipular los procesos de forma manual de acuerdo a sus necesidades y consideraciones. En un cluster con MOSIX todos los nodos deben tener el mismo sistema operativo.

4.2 LA HERRAMIENTA MOSIX

Es conveniente que antes de continuar con el desarrollo de éste capítulo veamos una definición de MOSIX.

MOSIX es una herramienta diseñada para realizar balanceo de cargas en un cluster de forma totalmente transparente, a tal grado que los nodos del cluster se comporten como una sola máquina y de esta manera incrementar el aprovechamiento de cada uno de ellos [25].

El núcleo de la herramienta MOSIX está formado por un conjunto de algoritmos diseñados para compartir recursos en una tecnología CC, además permiten la migración dinámica y eficiente de procesos entre diferentes computadoras. Estos algoritmos monitorean y responden a la distribución de los procesos entre los diferentes nodos. Obteniendo así un máximo desempeño.

Los algoritmos de MOSIX son descentralizados debido a que asignan y reasignan procesos. Cada nodo puede funcionar como maestro, para los procesos que fueron creados localmente y puede funcionar como servidor para distribuir los procesos que serán enviados a otros nodos.

Los manuales indican que la instalación automática de MOSIX-1.5.7 se puede hacer al mismo tiempo que se agrega la nueva versión de kernel Linux-2.4.17. Este procedimiento no me funcionó, entonces, primero agregué el nuevo kernel y posteriormente agregué MOSIX. Por lo tanto, el procedimiento que se describirá seguirá en este orden.

4.3 INSTALACIÓN DEL NUEVO KERNEL -2.4.17

El kernel es el núcleo del Sistema Operativo y en el caso de Linux es el encargado de administrar todas las tareas que requiere el sistema, el kernel posee todos los controladores de cada uno de los dispositivos del sistema. El usuario puede optimizar el kernel generando una configuración personalizada acorde a sus necesidades.

Cuando Linux inicia, se carga un archivo (kernel comprimido) en memoria que se localiza en el directorio /boot, y se llama /boot/vmlinuz-x.x.x, donde x.x.x es la versión del kernel que se tenga instalada.

La idea de configurar y optimizar el kernel, es la de crear un archivo con los controladores y datos que sean acordes a la configuración técnica de nuestra máquina. Para hacer esto, los controladores (*drivers*) pueden ser configurados de dos formas, ya sea que formen parte del kernel o bien configurarlos como módulos cargables, el propósito de este concepto radica en que hay controladores, como el audio por ejemplo, que no hacen falta todo el tiempo, si no que solo cuando se va a utilizar una aplicación tipo CD Player. Si por el contrario, esta aplicación fuera necesaria para el funcionamiento del kernel, entonces desde un principio convendría que el controlador forme parte de éste. Si se elige cargarlo como

módulo, se puede hacer de dos maneras, habilitar la opción para que el kernel lo cargue automáticamente o bien cargar el módulo manualmente.

4.3.1 Distribución del kernel

El sitio oficial, recomendado para obtener la distribución fuente del kernel linux-2.4.17 es [27]. Una vez que se ha descargado el archivo, se recomienda copiar la distribución fuente de linux-2.4.17.tar.bz2 en el directorio /usr/src/.

4.3.2 Compilar el kernel

Para realizar todas las operaciones que se describen a continuación se debe entrar al sistema como super usuario.

Una recomendación importante antes de agregar un nuevo kernel, es hacer un respaldo de toda la información importante para el usuario.

Posteriormente, cuando se descomprime y se desempaqueta la distribución fuente de linux-2.4.17.tar.bz2 se crea el directorio /usr/src/linux, ahora se cambia el nombre del directorio a /usr/src/linux-2.4.17 que corresponde a la nueva versión del kernel que se instalará.

Antes de iniciar la compilación es necesario tener instalado las versiones correctas de las siguientes utilerías [32]: **make** versión 3.77, **gcc** versión 2.95.3, 2.95.4 o 2.96.74 y **binutils** versión 2.9.1.0.25 o alguna versión posterior.

Dentro del directorio /usr/src/linux-2.4.17, se tienen tres opciones para trabajar en la configuración del nuevo kernel [29]:

- ***make xconfig***. Es un modo gráfico que tiene que ser lanzado desde una terminal X.
- ***make menuconfig***. Se tiene una interfaz gráfica en modo texto con menús.
- ***make config***. Es modo texto, preguntas y respuestas, además no permite ir hacia atrás para corregir alguna opción incorrecta.

En la figura 4.3-1, se muestra la interfaz de configuración de la opción menuconfig.

Posteriormente se habilitan las diferentes opciones que el usuario necesita configurar para optimizar su nuevo kernel, grabando todos los cambios al final. En este punto se pueden hacer dos cosas, una es grabar la configuración en un archivo que el usuario puede definir y la otra es guardar en el mismo archivo *.config*.

Terminado el proceso de selección, se ejecuta el comando "***make dep***" que creará las dependencias entre los distintos módulos del kernel.

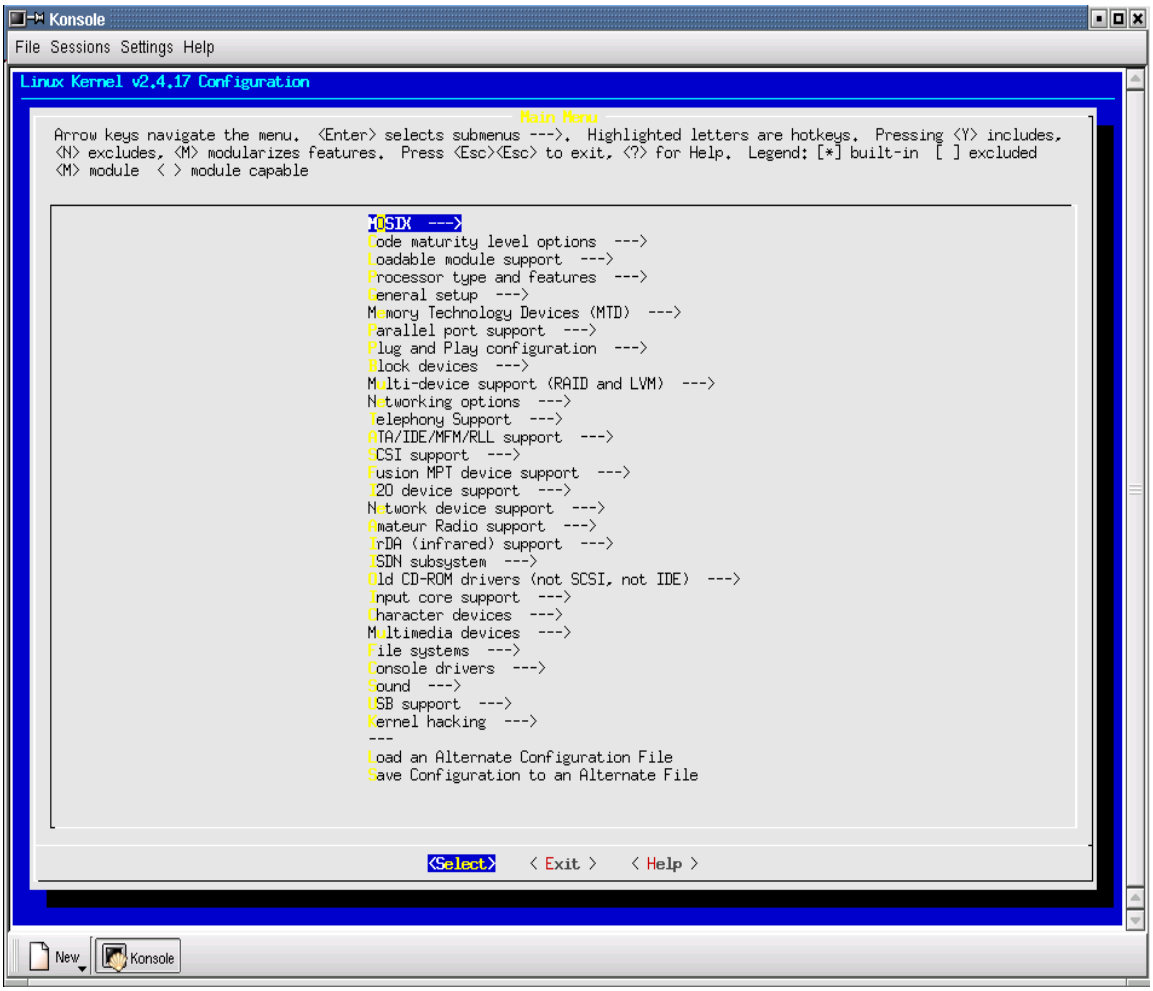


Figura 4.3-1 Interfaz de configuración del kernel usando "menuconfig".

Después se ejecuta el comando "***make bzImage***", este paso es muy importante ya que comprime el nuevo kernel que se ha instalado. Puede ser que aparezca un mensaje indicando que la memoria caché es muy pequeña. Si esto sucede, entonces se ejecuta "***make zImage***". Cabe mencionar, que el sistema operativo Linux hace distinción entre palabras escritas con mayúsculas y minúsculas.

Terminado este proceso se ejecuta el comando "***make modules***" que creará los módulos que el usuario ha definido. Antes de ejecutar el último comando, es necesario crear un directorio en /lib/modules/, con el nombre de la versión del kernel. En dicho directorio se guardarán todos lo controladores, por lo tanto, el directorio queda de la siguiente manera /lib/modules/2.4.17, entonces ahora sí, se procede a ejecutar "***make modules_install***", para instalar todos los módulos creados en el paso anterior.

4.3.3 Agregar el kernel a Linux

Hasta este punto se han creado dos archivos que son muy importantes, uno es *System.map* y se encuentra en el directorio /usr/src/linux/, el otro archivo es *bzImage* y se encuentra en el directorio /usr/src/linux/arch/i386/boot/.

Estos archivos se deben copiar con el nombre que el usuario desee, al directorio /boot/:

```
# cp /usr/src/linux/System.map  /boot/System.map-2.4.17
# cp /usr/src/linux/arch/i386/boot/bzImage  /boot/vmlinux-2.4.17
```

Ahora se debe borrar el archivo /boot/System.map, que suele ser una liga simbólica antigua y se crea nuevamente para la nueva versión de kernel System.map-2.4.17:

```
#  cd /boot
#  rm System.map
#  ln –s  System.map-2.4.17 System.map
```

Posteriormente se agrega el nuevo kernel a LInux LOader (LILO). Se recomienda no quitar el kernel que trae originalmente la distribución de Linux, para lo cual se debe modificar editando el siguiente archivo /etc/lilo.conf, de la siguiente manera:

image = /boot/vmlinux-2.4.17	Aquí se pone el nuevo kernel compilado.
label = Linux-2.4.17	Aquí se usa la etiqueta que el usuario desee.
root = /dev/hdax	Donde “x” es el número de la partición raíz del disco duro, debe ser la misma partición del kernel original.

Después de hacer estos pasos hay que reinstalar lilo para verificar que ha sido agregado correctamente el nuevo kernel. Se escribe en consola “***lilo –v***”.

Si existe algún error, entonces será indicado y si los pasos anteriores se han hecho correctamente, entonces el error probablemente puede ser que aparezca por alguna equivocación de escritura en alguna de las líneas al editar el archivo lilo.conf, se corrige, se reinstala nuevamente y se reinicia la PC eligiendo el nuevo kernel.

4.4 INSTALACIÓN DE MOSIX

Existen dos formas de instalar MOSIX, una es la forma automática y la otra es la forma manual [28][32], la forma que se describirá, es la instalación automática.

Tanto la instalación automática como la instalación manual se han probado con la distribución de Red Hat 5.1, 6.0, 6.2, 7.0, 7.1 y la distribución SuSE 6.0, 6.1, 6.2, 6.3, 7.0. De acuerdo a mi criterio, se puede decir que no importa el número de versión de

Linux, siempre y cuando se tenga el kernel correcto que especifique la distribución de MOSIX.

En particular se instaló la versión MOSIX-1.5.7 para la versión de kernel Linux-2.4.17 y se utilizó la distribución SuSE-7.3. Este es un punto muy importante, debido a que si no existe congruencia entre la versión del kernel y la versión de MOSIX, puede ser que los nodos no funcionen correctamente.

4.4.1 Distribución fuente de MOSIX

El sitio oficial, para obtener la distribución fuente de MOSIX-1.5.7 es [25].

Para crear un cluster, es necesario instalar por separado MOSIX en todos los nodos que formarán el cluster. Es preferible que todos los nodos en el cluster tengan la misma versión de kernel y de preferencia la misma versión de MOSIX instalada, aunque se pueden mezclar dos versiones que coincidan en los primeros dos dígitos de la versión de MOSIX. Si estos requerimientos son violados, los nodos pueden rechazar la comunicación entre ellos.

Para la instalación automática, son necesarios los siguientes pasos:

1. Una vez que se ha descargado la distribución de MOSIX en el directorio raíz, se descomprime y desempaqueta:

```
# cd  /root
# tar –xzvf  MOSIX-1.5.7.tar.gz
```

2. Ejecutar:

```
# cd MOSIX-1.5.7
# ./mosix.install
```

Posteriormente "***mosix.install***" nos guiará paso a paso durante la instalación. Una vez que MOSIX se empieza a instalar, pide que se escriba en la línea de comandos la opción de configuración (make xconfig, make menuconfig, make config) con la que el usuario desee trabajar. Durante la instalación, "***mosix.install***" modifica los archivos listados abajo. El contenido original de los archivos es grabado con la extensión ".pre_mosix".

Los archivos son:

/etc/inittab	
/etc/inetd.conf y/o /etc/xiniet.d,	dependiendo de la distribución que se tenga instalada SuSE o Red Hat.
/etc/lilo.conf	
/etc/rc.d/init.d/atd	
/etc/cron.dayly/slocate.cron	

Si alguno de estos archivos es modificado después de haber instalado MOSIX y principalmente si son modificados por alguna instalación de una nueva versión de Linux, puede darse el caso que algunos archivos sean alterados, lo que provocará que no funcione correctamente MOSIX.

Cuando MOSIX es desinstalado ("mosix.install unistall") todos los archivos que fueron modificados, son restaurados con sus valores originales.

4.4.2 Configuración de MOSIX

En la figura 4.4-1 se muestran las opciones que son necesarias habilitar en el kernel para que MOSIX funcione [28].

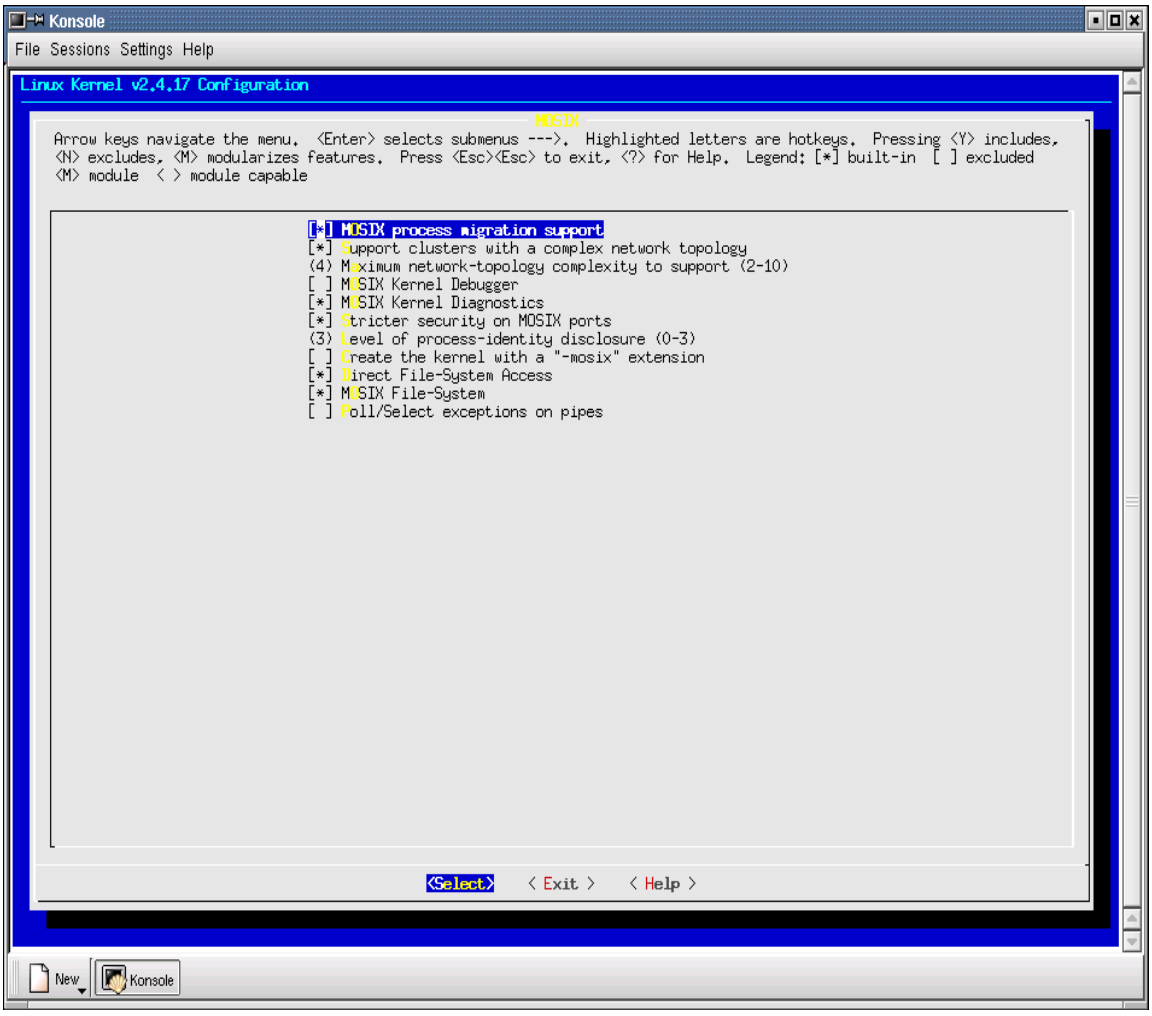


Figura 4.4-1 Opciones de configuración de MOSIX.

[*] MOSIX process migration support.

Esta opción habilita completamente MOSIX.

[*] Support clusters with a complex network topology

Esta opción se habilita cuando los nodos del cluster no forman parte de una simple LAN, es decir, que existan nodos remotos y que estén separados por algún gateway o si las tarjetas de red de los nodos difieren ampliamente en sus características.

[*] Stricter security on MOSIX ports

Esta opción es altamente recomendada debido a que el cluster MOSIX estará protegido ante ataques que intenten dañar el cluster, por ejemplo, en caso de que se tenga acceso a Internet.

[*] MOSIX File-System

Esta opción habilita un conjunto de utilerías que pueden ser usadas para compartir y copiar archivos manualmente entre los nodos que forma el cluster. Mosix File System es altamente recomendado para el manejo de archivos en MOSIX.

[*] Direct File-System Acces

Habilitando esta opción, se incrementa el desempeño en la migración de los procesos del nodo principal hacia los demás nodos en el momento de intercomunicación de estos.

Una vez que las anteriores opciones han sido habilitadas, se guardan los cambios y la instalación de MOSIX continuará de manera automática. Un punto importante, es asegurar que todos los nodos tienen la misma configuración, ya que de lo contrario como se mencionó anteriormente, puede ser que no haya comunicación entre estos.

4.4.3 Configuración del archivo mosix.map

Algo que es necesario para que exista comunicación entre los nodos del cluster, es la configuración de un archivo llamado “mosix.map” [32], que contiene todas las direcciones IP de todos los nodos que se van a instalar. La ubicación del archivo está en /etc/mosix.map. Si al final de la instalación por alguna razón no se encuentra éste archivo, es necesario crearlo y configurarlo manualmente.

En el archivo /etc/mosix.map, en forma de renglones se definen los nodos que formarán el cluster MOSIX, escribiendo el número de nodo, las direcciones IP y cuantos nodos con números IP son consecutivos. Al editar este archivo es importante tener en cuenta que no se deben tener líneas en blanco y en caso de que así sea, cada línea debe ir comentada con el símbolo “#”.

La configuración queda de la siguiente manera, cada renglón contiene 3 campos:

Número de nodo En este campo se coloca el número de nodo, que será identificado por MOSIX (soporta hasta un total de 65535 nodos).

Dirección IP En este campo se agrega la dirección IP del nodo participante en el cluster, la cual debe corresponder a la dirección que se encuentra listada en el archivo"/etc/hosts"

Tamaño de rango En este campo se agrega el número total de nodos que tienen una dirección IP consecutiva.

Sin embargo en el campo tamaño de rango, se puede tener un valor de 1 o la palabra "ALIAS", esto es necesario cuando un nodo tiene más de una dirección IP debido a que puede actuar como puerta de enlace (gateway) de dos o más redes entre los nodos de mismo cluster MOSIX, es decir, cuando se tienen nodos remotos.

A continuación se describen dos puntos muy importantes:

1. El archivo /etc/mosix.map se espera que sea igual en cada uno de los nodos del cluster.

2. Si la tabla está correcta, pero algún nodo en particular no puede ser visto por los demás nodos del cluster, significa que existe algún error y se debe a que la dirección IP del nodo no está incluida en la tabla del archivo "mosix.map". La dirección IP se usa junto con el archivo /etc/mosix.map para determinar el número de nodo local.

4.4.4 Ejemplos de casos más comunes de configuración del archivo mosix.map

Caso 1. En un cluster básico de dos nodos, donde se tienen direcciones IP que no son consecutivas, por ejemplo, el primer nodo tiene una dirección 200.200.120.5 y el segundo nodo tiene una dirección 200.200.120.10, entonces /etc/mosix.map debe quedar de la siguiente manera:

```
1       200.200.120.5       1
2       200.200.120.5       1
```

Caso 2. Ahora en el caso cuando los nodos tienen direcciones IP consecutivas, por ejemplo, si la dirección IP del primer nodo es 200.200.120.5, la dirección del segundo nodo es 200.200.120.6 y la dirección del tercer nodo es 200.200.120.7, entonces /etc/mosix.map quedaría de la siguiente forma:

```
1       200.200.120.5       3
```

Caso 3. En el caso cuando se tienen 6 nodos, por ejemplo, 3 nodos en una red clase-C con dirección 200.200.120.xxx, 1 nodo con dirección 200.200.130.xxx y 2 nodos con dirección 200.200.140.xxx, entonces /etc/mosix.map quedaría de la siguiente manera:

```
1       200.200.120.1       3
4       200.200.130.1       1
5       200.200.140.1       2
```

Caso 4. En un caso más complejo, se tienen 13 nodos, por ejemplo, 6 nodos en una red con dirección IP 192.168.0.xxx, 6 nodos en una red con dirección 192.168.1.xxx y un gateway conectando las dos redes que usan ambas direcciones IP 192.168.0.100 y 192.168.1.100, entonces /etc/mosix.map quedaría así:

```
1       192.168.0.1         6
1       192.168.0.100       1
1       192.168.1.100       ALIAS
1       192.168.1.1         6
```

Cuando la red del cluster MOSIX, es segmentada por gateways, entonces se escribe el número 1, en el archivo /etc/mosgates, en caso de que exista sólo un gateway o el número 2 en caso de que existan dos gateways entre los nodos del cluster.

MOSIX necesita ser activado inmediatamente después del demonio de red en cada nodo y debe ser desactivado antes de que el demonio de red sea desactivado en el nodo, es decir, MOSIX debe ser desactivado antes de que se apague el equipo.

En particular, la instalación del cluster implementado es similar al caso 3, debido a que el cluster tiene dos nodos y las direcciones IP son consecutivas.

4.5 AGREGAR EL RESULTADO FINAL DE MOSIX AL NUEVO KERNEL

Una vez que ya se ha instalado y configurado MOSIX en el nuevo kernel, el siguiente paso es agregar los nuevos archivos generados por la instalación; "/usr/src/System.map" y "/usr/src/linux/arch/i386/boot/bzImage" al directorio /boot/ como sigue. En lo personal agregué la terminación "***mosix***" como identificación:

```
# cp /usr/src/System.map  /boot/System.map_mosix

# cp /usr/src/linux/arch/i386/boot/bzImage  /boot/vmlinux_mosix
```

Es necesario borrar la liga simbólica System.map que apunta a System.map-2.4.17 y crearla nuevamente para el nuevo archivo generado *System.map_mosix*:

```
# cd  /boot

#  rm System.map

# ln –s  System.map_mosix  System.map
```

Los archivos /boot/System-2.4.17 y /boot/vmlinux-2.4.17 pueden ser borrados. Aunque estos archivos inicializan el nuevo kernel linux-2.4.17 pero aún no tienen agregada la herramienta MOSIX.

Nuevamente es necesario editar y modificar el archivo /etc/lilo.conf para agregar los cambios de la nueva compilación, una vez que MOSIX-1.5.7 ha sido agregado y configurado en el nuevo kernel-2.4-17.

Los pasos son similares a los de la sección 4.3.3. Se edita el archivo /etd/lilo.conf y solamente se modificará el primer renglón, eliminando el archivo vmlinux-2.4.17 y sustituyéndolo por el archivo vmlinux_mosix (nuevo kernel), entonces quedaría de la siguiente manera:

image = /boot/vmlinux-mosix

label = Mosix_Linux

root = /dev/hdax Donde "x" es el número de la partición raíz del disco duro.

A criterio o gusto del usuario, puede cambiar el nombre de la etiqueta (*label*), dependiendo de la identificación que se le quiera dar al nuevo kernel, en este caso se agrega la terminación "***mosix***" para identificar que ha sido agregado MOSIX:

label = Linux-2.4.17

se cambió por

label = Mosix_Linux.

Luego de modificar y guardar los cambios hechos a /etc/lilo.conf, se debe nuevamente ejecutar en línea de comando, "***lilo –v***" para que se cargue la nueva configuración.

4.6 EJECUCIÓN DE MOSIX

Ahora lo que queda es reiniciar la computadora y escoger el nuevo kernel "Mosix_Linux". MOSIX puede ser ejecutado en modo texto o en modo gráfico. Entonces una vez que se ha ingresado completamente al sistema, para ejecutar MOSIX, se debe escribir en una terminal el comando "***mon***" (llamado también, Monitor de MOSIX) e inmediatamente debe aparecer una interfaz gráfica donde se muestra el número total de nodos que participan en el cluster, como se muestra en la figura 4.6-1.

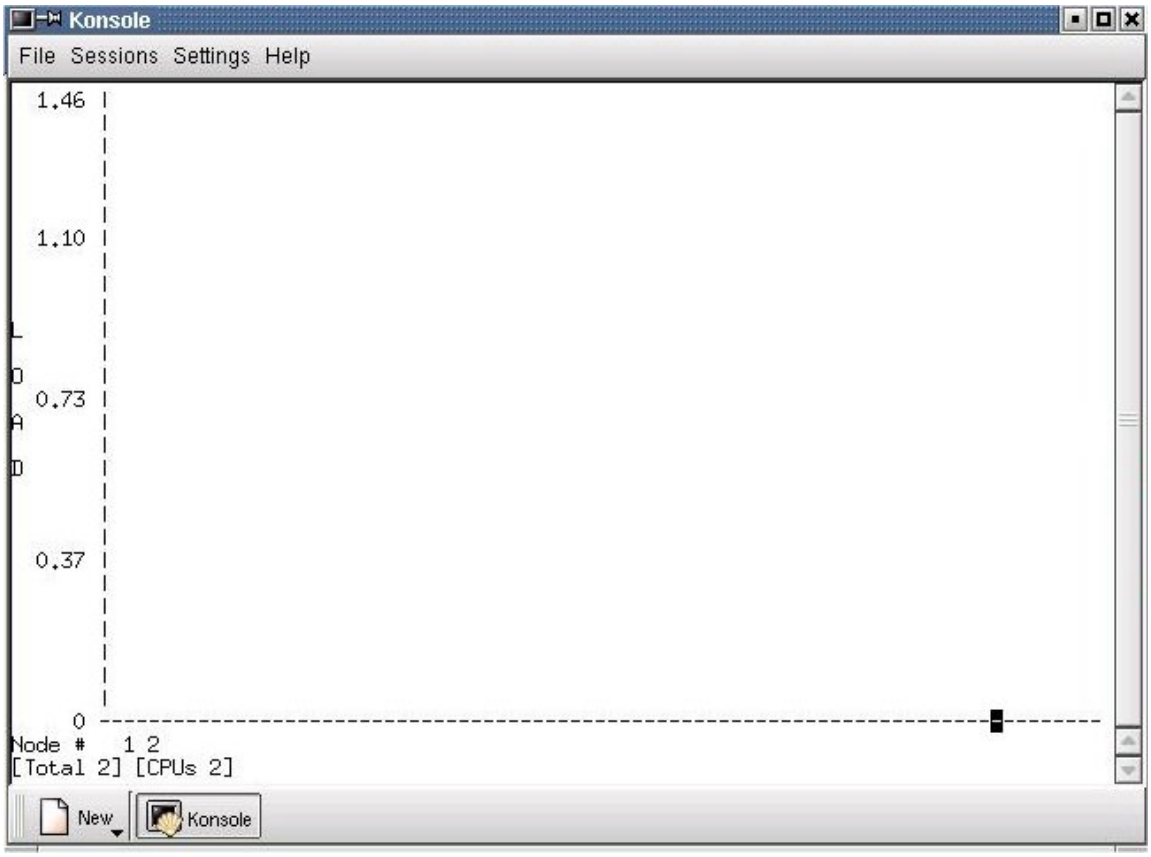


Figura 4.6-1 Pantalla principal de MOSIX.

En la figura 4.6-2 se muestra la forma de representación de los procesos. Cada rectángulo de color negro, representa un proceso en ejecución y en la parte baja de cada rectángulo, se muestra el número de nodo en el que el proceso se está ejecutando.

Para un cluster con un número considerable de nodos (con 20 o más), probablemente no se puedan mostrar todos en una sola pantalla, en este caso MOSIX da la posibilidad de poder mostrar la pantalla en forma horizontal o vertical de tal forma que para el usuario sea más fácil visualizar el número total de nodos, este procedimiento se describe con más detalle en la siguiente sección.

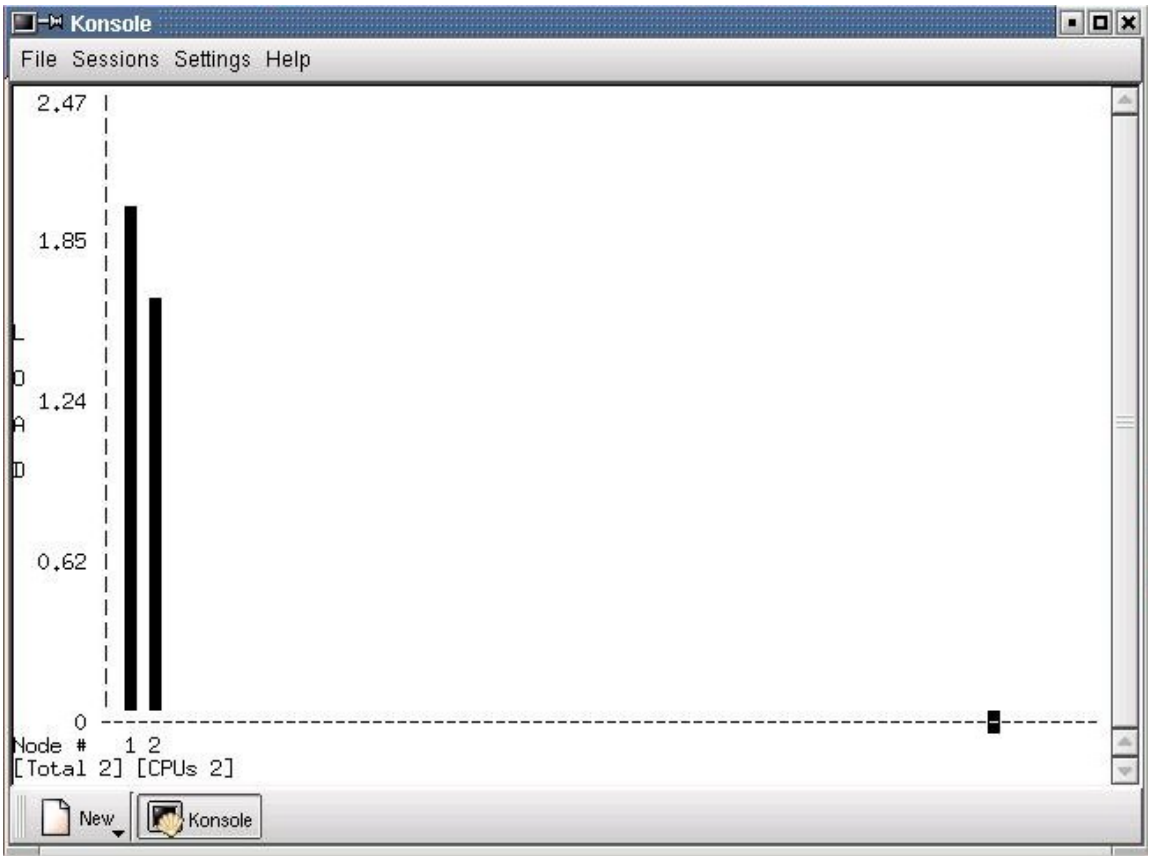


Figura 4.6-2 Representación de los procesos en MOSIX.

4.6.1 Herramientas de monitoreo

MOSIX también trae herramientas para observar lo que esta pasando en cada nodo, es decir, podemos observar el comportamiento de todo el cluster.

Una vez que es ejecutado el monitor de carga de MOSIX llamado "***mon***", alternativamente, puede mostrarse la velocidad de los procesadores, o la memoria disponible, la memoria total y la carga en cada uno de los nodos. De la siguiente manera:

 # mon [opción]

Las opciones disponibles son las siguientes:

-v / -V Selección de numeración vertical, permite que la carga de los nodos sea mostrada en cualquier momento cuando el número de nodos es demasiado grande. La opción –v puede ser mayúscula o minúscula.

-w Selección de numeración horizontal, permite ver los nodos que están activos en cualquier momento.

-t Despliega el número total de nodos que están operando en el cluster.

mon + {numero de nodo} Despliega la carga, en algún nodo especial elegido por el usuario.

-d Despliega los nodos que están configurados pero no responden a las peticiones de comunicación (llamados también nodos muertos).

Mientras se ejecuta "***mon***" se pueden visualizar varias cosas en los nodos, pulsando las siguientes teclas:

h Muestra la ventana de ayuda.

Enter Redibuja la ventana.

v Muestra la numeración de los nodos en forma vertical.

w Muestra la numeración de los nodos en forma horizontal.

a Muestra la numeración de manera automática. La numeración vertical debe ser seleccionada si el usuario desea hacer diferencia entre la capacidad de ver la carga de todos los nodos o no. En caso contrario la numeración horizontal es mostrada por omisión.

s Muestra la velocidad de los procesadores en vez de la carga. Esta opción también muestra el número de nodo.

m Despliega la cantidad de memoria lógicamente usada por un proceso. Esta memoria usada se muestra como una barra sólida mientras que la memoria total es mostrada con signos +.

r Muestra la memoria completa real de cada nodo. La memoria se muestra en megabytes.

u Muestra el porcentaje de utilización del procesador.

l Muestra la carga nuevamente.

d Muestra los nodos configurados pero que no responden (nodos muertos).

D Anula automáticamente los nodos muertos.

t Muestra la cantidad total de los nodos que están operando.

Flecha Derecha (→) Right-Arrow Mueve un nodo a la derecha (cuando no todos los nodos pueden verse en la pantalla).

Flecha Izquierda (←) Left-Arrow Mueve un nodo a la izquierda (cuando no todos los nodos pueden verse en la pantalla).

n Mueve una pantalla a la derecha (cuando no todos los nodos pueden verse en la pantalla).

p Mueve una pantalla a la izquierda (cuando no todos los nodos pueden verse en la pantalla).

q / Q Termina la ejecución de MOSIX.

CAPÍTULO 5

TEORÍA DE HALOS E IMPLEMENTACIÓN EN EL CLUSTER

5.1 INTRODUCCIÓN

En este capítulo, se hace una descripción completa de la aplicación desarrollada para probar el funcionamiento y rendimiento del cluster MOSIX, así como también se muestran de manera gráfica las pruebas de migración de procesos entre los nodos.

La aplicación que se desarrolló, consiste en simular la formación de "halos". Los halos son fenómenos ópticos que se presentan en la atmósfera de manera natural, debido a que en ocasiones bajo ciertas condiciones de temperatura, el agua que existe en la atmósfera se cristaliza, entonces, al pasar la luz del sol a través de estos cristales, se forma un círculo luminoso alrededor del sol o de la luna llamado halo.

Al simular un halo por computadora se toman en cuenta distintos parámetros, como, las longitudes de onda, posición de los cristales, ángulos de incidencia de rayos, número de cristales, entre otros, de tal manera que a veces la información que se tiene que procesar requiere mayor fuerza de cómputo para realizar el trabajo en el menor tiempo posible. Es en este caso cuando se aplica la fuerza que proporciona un cluster distribuyendo la carga de trabajo en los diferentes nodos y de esta manera procesar más rápidamente la información.

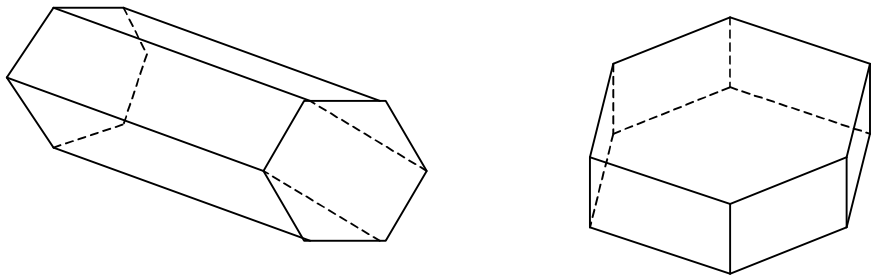
La simulación, consiste de un programa desarrollado en lenguaje C bajo el sistema operativo Linux y se ejecuta de forma distribuida en los nodos que forman el cluster.

5.2 FORMACIÓN DE HALOS

Un halo es un fenómeno óptico que se genera por la descomposición de un rayo de luz blanca al pasar por cristales de hielo (con forma de prismas hexagonales) que se forman en la atmósfera.

Al pasar el rayo de luz a través del cristal, se descompone en sus diferentes frecuencias. Cada frecuencia específica generará un determinado color y una determinada desviación angular al pasar la luz por el cristal.

Los cristales más comunes que se forman en la atmósfera son de forma hexagonal como los que se muestran en la figura 5.2-1, por lo que son éstos los que se tomaron en cuenta para la realizar la simulación [33].



(a) Cristal tipo lápiz. (b) cristal tipo plato.

Figura 5.2-1 Tipos de cristales.

El tamaño de los cristales aproximadamente varía entre 0.3, 0.05, 0.1 milímetros. Con este tipo de cristales ocurren varios efectos que resultan del paso de un rayo de luz a través de una de las caras de los cristales hexagonales. Aunque hay una gran diversidad de efectos, el más común que se presenta es el llamado, "**Halo de 22 grados**", entonces, es por esta razón que se tomó como base para realizar la aplicación.

Para un mejor estudio e ilustración de la descomposición de la luz en sus diferentes colores, los cristales de hielo pueden tratarse como secciones de prismas triangulares, esta semejanza es mostrada en la figura de 5.2-2. Tres de las caras de un cristal tipo lápiz han sido extendidas, para ilustrar como un rayo de luz que pasa a través de una de sus caras es desviado exactamente como si éste rayo pasara por un prisma, en el cual, cada uno de sus ángulos miden 60°.

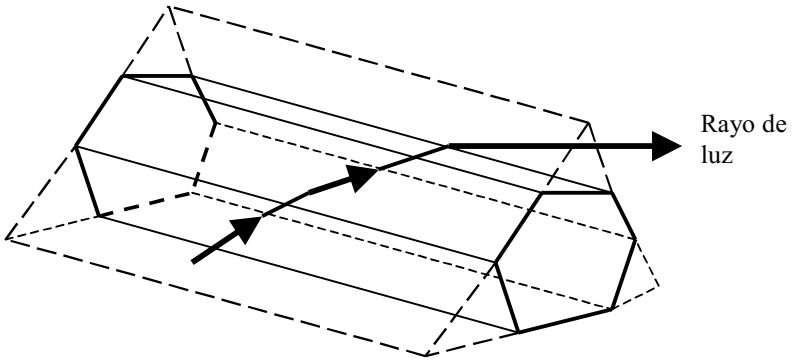


Figura 5-2.2 Prisma con un ángulo de 60°.

En la figura 5.2-3, se muestra de manera más detallada uno de los ángulos de 60° del prisma. También se muestra un rayo de luz que incide en una de las caras del prisma, de manera que el ángulo de desviación total "δ" al salir del cristal es cercano a los 22°. La magnitud del ángulo de desviación del rayo puede incrementar si el cristal es rotado en torno a su eje. Para un prisma de hielo que tiene un ángulo de 60°, la desviación angular mínima de cualquier rayo será de 22°.

En consecuencia, cuando un gran número de rayos de luz inciden en un gran número de cristales tipo lápiz con todas las orientaciones aleatorias posibles que pudieran tener éstos, entonces, hay una concentración de rayos desviados por ángulos cercanos a los de 22°, de ahí el nombre que se le da.

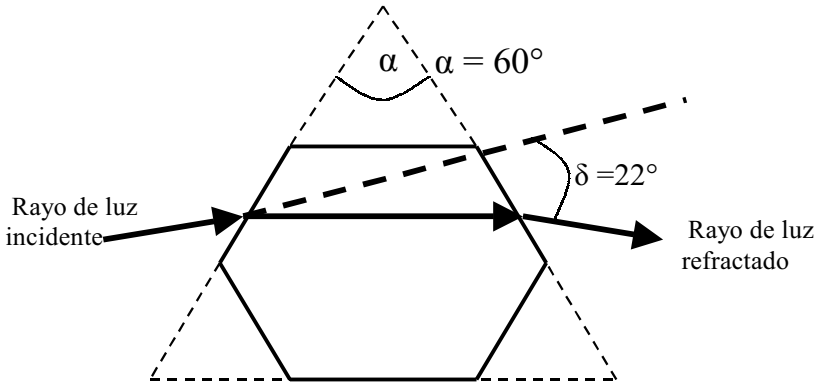


Figura 5.2-3 Rayo refractado con un ángulo mínimo de 22° en un prisma en el que sus ángulos miden 60°.

Para ilustrar cómo un observador percibe un rayo de luz refractado con un ángulo δ, se muestra en la figura 5.2-4. El resultado de tal refracción de un gran número de cristales cayendo en la atmósfera con todas las posibles orientaciones, generan el efecto llamado "**halo**"

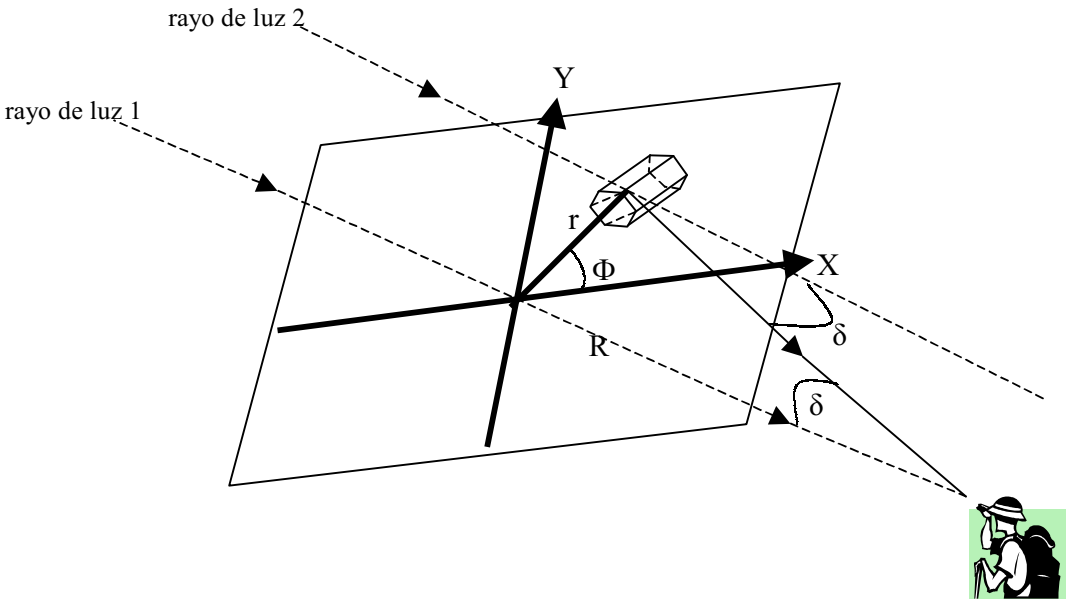


Figura 5.2-4 Manera de cómo un rayo de luz es refractado al ojo de un observador.

5.3 MODELO PROPUESTO PARA LA SIMULACIÓN

Antes de explicar detalladamente el modelo de simulación, es necesario definir y deducir ciertos términos que son importantes para realizar la simulación del halo.

5.3.1 Prisma dispersor

Cuando se tiene un rayo de luz que incide sobre una superficie plana que separa dos medios transparentes, existe una ecuación que describe el comportamiento del rayo al pasar a través de esta superficie (refracción), a esta ecuación se le conoce como *ley de la refracción o ley de Snell* [34].

$$n_i \text{sen}\theta_i = n_t \text{sen}\theta_t \tag{5.1}$$

La refracción se toma como base para explicar el efecto de la refracción de la luz al pasar por un cristal y como consecuencia para explicar el modelo de simulación de la formación de halos.

Para simular la aplicación del halo, en particular se usó un prisma triangular que funciona como prisma dispersor y es capaz de separar las frecuencias que constituyen un rayo de luz policromático, como se muestra en la figura 5.3-1.

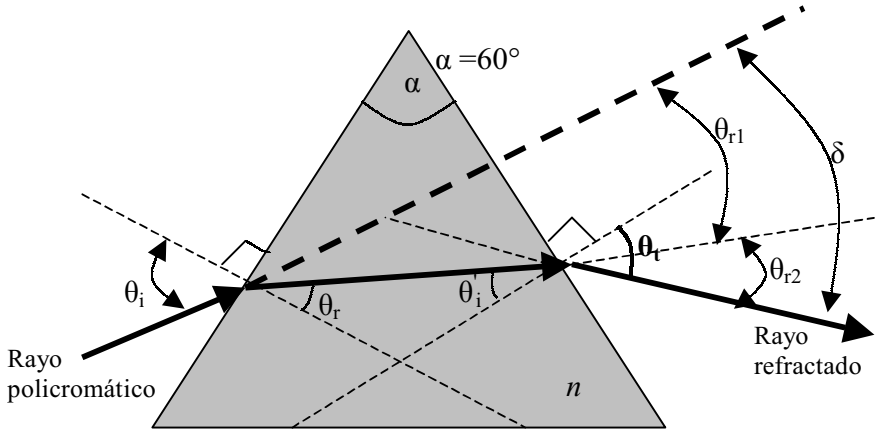


Figura 5.3-1 Trayectoria de un rayo en un prisma dispersor.

Al incidir un rayo de luz en un prisma dispersor que tiene un índice de refacción "n" diferente para cada frecuencia, lo hace con un cierto ángulo "θ_i" con respecto a la normal y la trayectoria que sigue el rayo al pasar por el cristal es la siguiente, al incidir el rayo con el cristal existe una primera refracción "θ_{r1}", posteriormente, cuando el rayo sale es refractado nuevamente "θ_{r2}", por lo que, al sumar de las dos refracciones se obtiene la desviación angular total denotada por "δ". Dado que un prisma dispersor es capaz de dividir un rayo policromático en sus diferentes frecuencias, la desviación angular total será para cada una de las frecuencias que constituyen el rayo de luz, es decir, se tiene una "δ" para cada una de las frecuencias que constituyen los colores de la luz.

En conclusión, típicamente un rayo de luz que entra a un prisma dispersor, emergerá después de haber sido deflectado de su dirección original por un ángulo conocido como *desviación angular total "δ".*

Los parámetros que describen el paso de un rayo de luz a través de un prisma dispersor están relacionados mediante la siguiente ecuación.

$$\delta = \theta_i + \text{sen}^{-1}[(\text{sen}\,\alpha)(n^2 - \text{sen}^2\theta_i)^{1/2} - \text{sen}\,\theta_i \cos\alpha] - \alpha \qquad (5.2)$$

Entonces, particularizando y tomando como base la figura 5.3-1 y la ecuación 5.2 se tiene que para la llevar a cabo la simulación del halo, son necesarios los siguientes elementos: ángulo de incidencia del rayo de luz (θ_i), índice de refracción del cristal (n), un ángulo interior del prisma (α) y la desviación angular total para cada rayo emergente del prisma (δ).

Ahora bien, para mostrar que la desviación mínima de un rayo es cercana a los 22° se tiene la ecuación 5.3, con la cual se obtiene la desviación mínima de todos los rayos incidentes en los cristales [34].

$$\delta_m = 2\,\text{arcsen}\left(n\,\text{sen}\frac{\alpha}{2}\right) - \alpha \qquad (5.3)$$

Donde α = 60° debido a que es uno de los ángulos que se toman como referencia del prisma triangular.

A continuación, usando la ecuación 5.3, en la tabla 5.3-1 se mencionan sólo algunos resultados que demuestran que para cualquier índice de refracción, el ángulo de desviación mínima que se puede obtener es aproximadamente 22°.

Tabla 5.3-1 Angulo de desviación mínima 'δ_m" para diferentes índices de refracción

Longitud de Onda (λ) μicras	Índice de refracción (n)	Angulo de desviación mínima (δ_m)
λ = 0.680	n = 1.3073	δ_m = 21.63
λ = 0.690	n = 1.3071	δ_m = 21.61
λ = 0.700	n = 1.3069	δ_m = 21.53
λ = 0.710	n = 1.3067	δ_m = 21.58
λ = 0.720	n = 1.3065	δ_m = 21.57
λ = 0.530	n = 1.3118	δ_m = 21.97
λ = 0.540	n = 1.3164	δ_m = 21.94
λ = 0.550	n = 1.3110	δ_m = 21.91
λ = 0.560	n = 1.3106	δ_m = 21.88
λ = 0.570	n = 1.3103	δ_m = 21.86
λ = 0.420	n = 1.3177	δ_m = 22.42
λ = 0.430	n = 1.3170	δ_m = 22.37

Tabla 5.3-1 Angulo de desviación mínima "δ_m" para diferentes índices de refracción (continuación)

$\lambda = 0.440$	n = 1.3163	$\delta_m = 22.31$
$\lambda = 0.450$	n = 1.3157	$\delta_m = 22.27$
$\lambda = 0.460$	n = 1.3151	$\delta_m = 22.22$

5.3.2 Modelo para la observación del halo

Para simular la imagen del halo en la computadora es necesario definir un modelo para representar tal efecto, en consecuencia, necesitamos conocer el tamaño que debe tener la imagen "N", la distancia a la que se encuentra el observador "R", la posición del cristal en la atmósfera que es denotada por "Φ" y el ángulo de desviación total "δ" de cada rayo que emerge del cristal.

El primer elemento a definir es el tamaño de la imagen "N", que en este caso particular es igual a 512 (N=512), entonces, tomando como base el tamaño de la imagen y el ángulo "δ", se puede definir la distancia a la que se encuentra el observador, como se muestran en la figura 5.3-2.

Por lo tanto, se tiene que

$$\tan\delta = \frac{\frac{N}{2}}{R} = \frac{N}{2R}$$

y despejando R queda

$$R = \frac{N}{2\tan\delta} \tag{5.4}$$

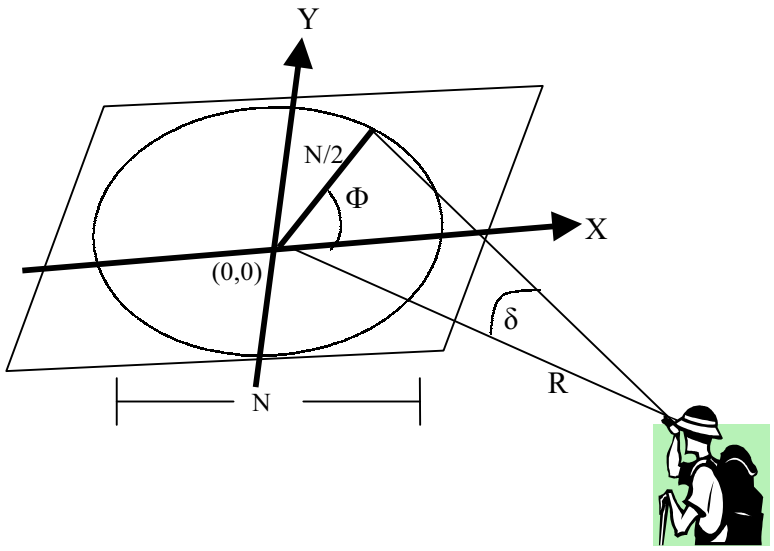


Figura 5.3-2 N es el tamaño de la imagen, R es la distancia a la que se encuentra el observador y δ el ángulo de desviación total de cada rayo.

Posteriormente, se tiene que la distancia (radio) del centro de la imagen (0,0) a donde se dibuja un punto está dada por "r". El valor de "r" varía de acuerdo al valor del ángulo "δ" de cada rayo, para ilustrar esta relación se toma como base la figura 5.3-3 y la ecuación se deduce de la siguiente forma

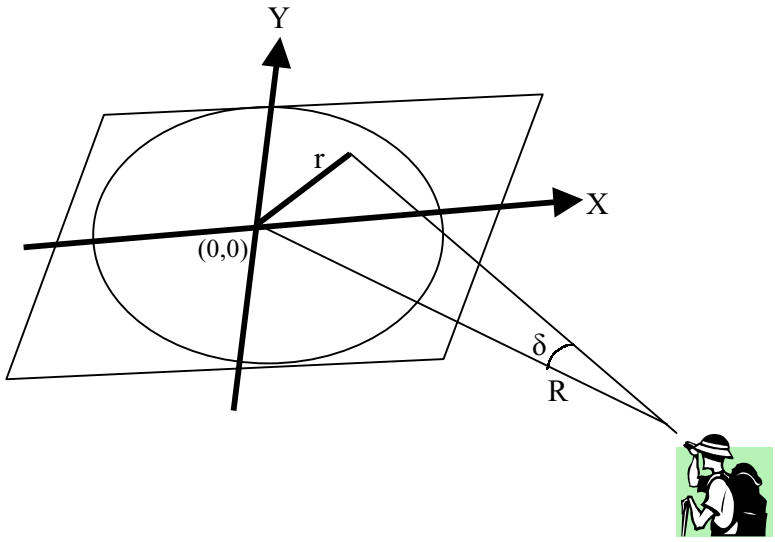


Figura 5.3-3 El radio "r" es la distancia a la que se dibuja un punto, tomando como referencia el centro de la imagen (0,0).

$$\tan\delta = \frac{r}{R}$$

despejando "r" queda

$$r = R\tan\delta \tag{5.5}$$

5.3.3 Modelo para la detección de rayos

Ahora bien, para ubicar exactamente las coordenadas x,y donde se dibujará cada punto correspondiente a un rayo, se toma en cuenta el ángulo Φ que corresponde a la ubicación de cada cristal y por lo tanto, se tiene la siguiente relación que se muestra en la figura 5.3-4

para ubicar la coordenada en "X" se tiene,

$$\cos\Phi = \frac{x}{r}$$

donde

$$x = r\cos\Phi \tag{5.6}$$

similarmente para la coordenada en "Y" tenemos que

$$\text{sen}\,\Phi = \frac{y}{r}$$

donde

$$y = r \operatorname{sen} \Phi \tag{5.7}$$

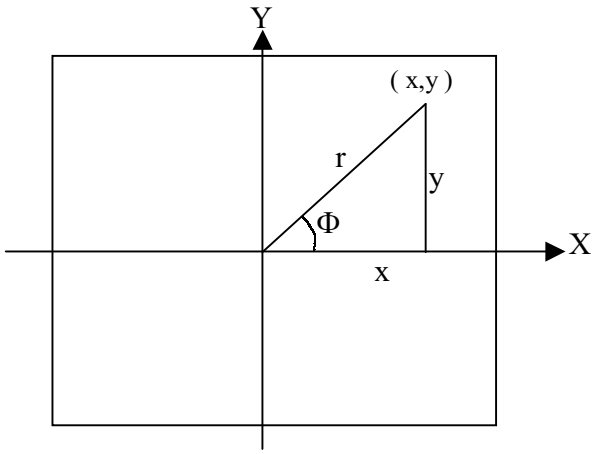


Figura 5.3-4 Relaciones de las coordenadas x,y de un punto en la imagen.

Con la obtención de estas coordenadas exactas se podrán dibujar todos los puntos que simulan los rayos de luz al ser refractados por los cristales y que formarán la imagen final del halo.

5.3.4 Representación digital del plano de observación

Para representar digitalmente un rayo del halo en la pantalla del monitor, se hace a través de un pixel que tiene cierta intensidad de color que va de acuerdo a la longitud de onda y a la cantidad de luz que haya incidido en la posición exacta del pixel. En conclusión se tiene que para dibujar un pixel son necesarios los siguientes elementos:

1. La desviación angular total de cada rayo de luz (δ).
2. El ángulo que indica la posición del cristal en la atmósfera (Φ).
3. La longitud de onda del color (λ).
4. Las coordenadas exactas (*x,y*).

Una vez que se tienen los elementos necesarios para ubicar un punto, las coordenadas exactas en el plano X,Y, (δ, Φ, λ) y el tamaño de la imagen (N) es necesario saber de que intensidad debe ser dicho punto.

Dado que la imagen tiene las dimensiones de 512 por 512 pixeles, entonces, esta dimensión es representada por una matriz de 512 renglones por 512 columnas, donde cada coordenada *x,y* de cada punto será representado por un pixel en la posición *i,j* de la matriz, es decir, para cada coordenada *x,y* se tiene una correspondencia *i,j* en la matriz de representación digital. Cada posición *i,j* en donde será pintado un pixel se considera como un caja pequeña en la cual se encuentran almacenados los colores que determinan la intensidad final de cada punto, por ejemplo, si las coordenadas *x,y* son (20,20), entonces, el pixel será pintado en la posición (20,20) de la matriz digital y así sucesivamente hasta obtener la imagen final.

En la figura 5.3.5 se muestra de manera general la forma de como se obtienen los valores de *i,j* que se usan para conocer la posición en donde se pintará cada pixel. El eje "X" está asociado o relacionado con las columnas y el eje de las "Y" está asociado o relacionado con los renglones. De esta manera con la letra "i" se obtiene el valor del renglón y con la letra "j" se obtiene el valor de la columna.

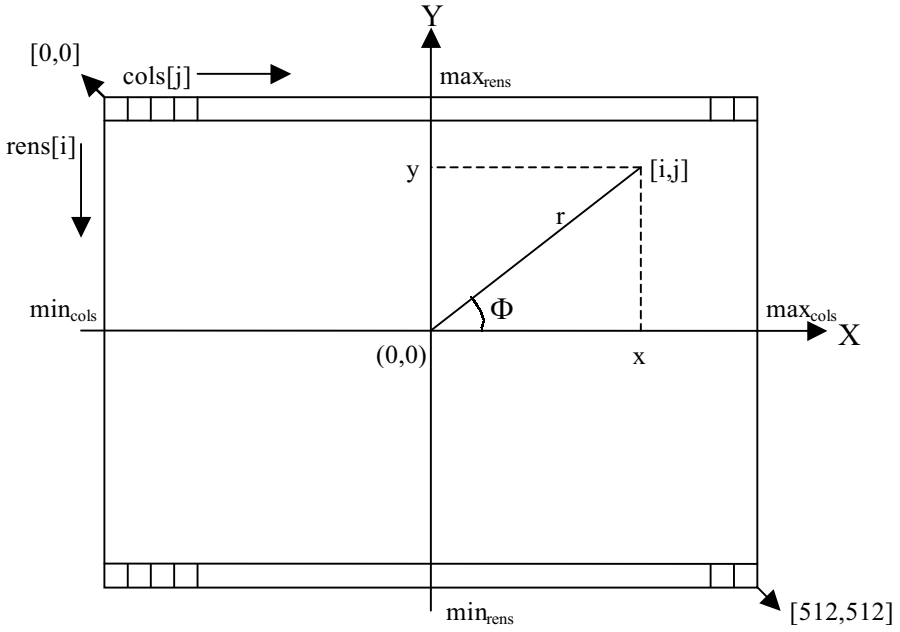


Figura 5.3-5 Ubicación exacta de un píxel en un punto i,j.

A continuación en la figura 5.3-6 (a), se muestra de manera detallada la forma en como se obtiene el valor de la columna "j"a partir de una semejanza de triángulos.

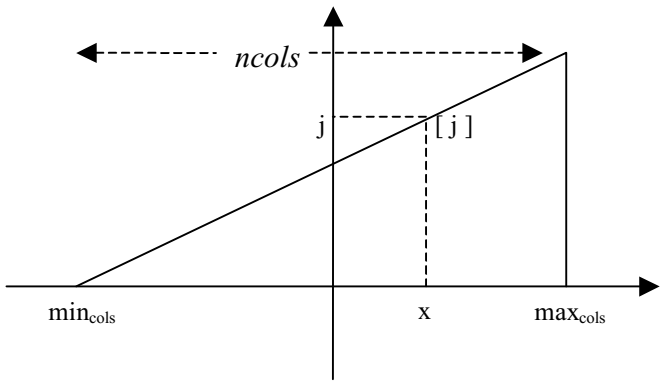


Figura 5.3-6 (a) Obtención del valor de la columna "j".

Se tiene que

$$\frac{ncols}{\max_{cols}-\min_{cols}}=\frac{j}{x-\min_{cols}}$$

ahora, despejando "**j"**

$$j = \left(\frac{x - \min_{cols}}{\max_{cols} - \min_{cols}} \right)(ncols) \tag{5.8}$$

esta ecuación 5.8, es un caso general para cualquier posición en una columna "j".

De igual manera, en la figura 5.3-6 (b), se muestra de manera detallada la forma en como se obtienen el valor del renglón "i".

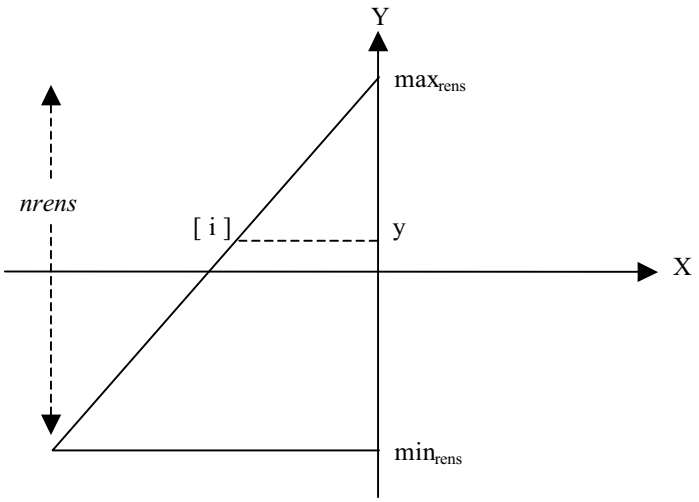


Figura 5.3-6 (b) Obtención del valor del renglón "i".

Se tiene que

$$\left(\frac{nrens}{\max_{rens} - \min_{rens}} \right) = \frac{i}{\max_{rens} - y}$$

ahora bien, si se despeja "**i"** queda

$$\mathrm{i} = \left(\frac{\max_{\mathrm{rens}} - \mathrm{y}}{\max_{\mathrm{rens}} - \min_{\mathrm{rens}}} \right)(\mathrm{nrens}) \tag{5.9}$$

esta ecuación 5.9 es el caso general para cualquier posición en un renglón "i".

Las ecuaciones 5.8 y 5.9 son un caso general para obtener cualquier punto en el eje de las "X" y en el eje de las "Y", ahora, particularizando los valores de estas ecuaciones al tamaño de la imagen real (halo de 22°) que se está manejando queda de la siguiente forma.

Los valores, max_{cols}, min_{cols}, se sustituyen por "N/2" y *ncols* se sustituye por "N", de la ecuación 5.8 queda de la siguiente forma

$$j = \left(\frac{x - \min_{cols}}{\max_{cols} - \min_{cols}} \right) (ncols)$$

$$j = \left[\frac{x - \left(-\frac{N}{2} \right)}{\frac{N}{2} - \left(-\frac{N}{2} \right)} \right] (N)$$

$$j = \left[\frac{x + \frac{N}{2}}{\frac{N}{2} + \frac{N}{2}} \right] (N)$$

$$j = \left(\frac{x + \frac{N}{2}}{N} \right) (N)$$

finalmente al eliminar la N del numerador con la N del denominador se obtiene

$$j = x + \frac{N}{2} \tag{5.10}$$

De igual forma para el caso de la ecuación 5.9 al sustituir los valores de max_{rens}, min_{rens}, por "N/2" y *nrens* por "N" por los valores originales de la imagen, se tiene

$$i = \left(\frac{\max_{rens} - y}{\max_{rens} - \min_{rens}} \right) (nrens)$$

$$i = \left[\frac{\frac{N}{2} - y}{\frac{N}{2} - \left(-\frac{N}{2} \right)} \right] (N)$$

$$i = \left[\frac{\frac{N}{2} - y}{\frac{N}{2} + \frac{N}{2}} \right] (N)$$

$$i = \left(\frac{\frac{N}{2} - y}{N} \right) (N)$$

eliminando la N del numerador con la N del denominador la ecuación final queda

$$i = \frac{N}{2} - y \tag{5.11}$$

Con las ecuaciones 5.10 y 5.11 se podrán pintar todos los pixeles en donde se ubiquen los cristales (puntos de coordenadas X,Y).

5.3.5 Simulación

Tomando como base las secciones 5.3.1, 5.3.2, 5.3.3 y 5.3.4 se explica la forma de cómo se realiza la iluminación de la imagen.

Los colores básicos del modelo RGB (rojo, verde y azul), se usan para construir la imagen del halo de 22°, es por eso, que la imagen resultante será una combinación de estos tres colores.

Es necesario mencionar que los cristales que se encuentran en la atmósfera y que forman el efecto real del halo, al caer a la tierra pueden tener cierta rotación, es decir, al descender pueden girar sobre su propio eje, como se muestra en la figura 5.3-7. y el mayor giro que puede tener un cristal al rotar sobre su propio eje es un ángulo de 360° (2π).

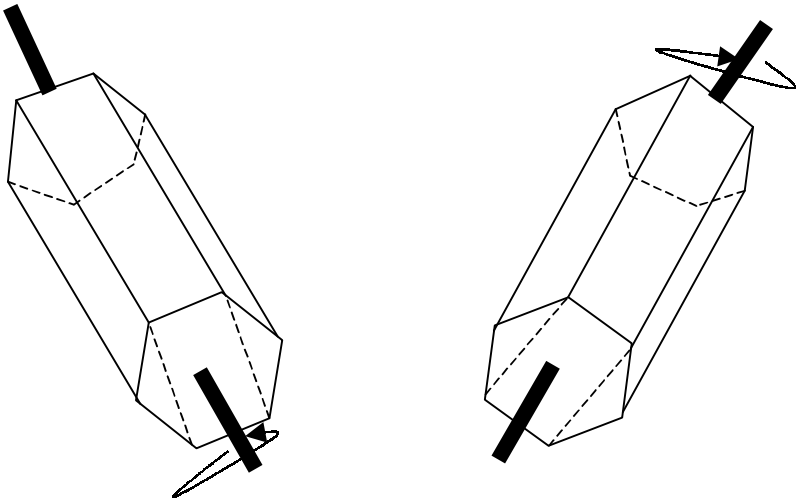


Figura 5.3-7 Primas que al caer giran sobre su propio eje.

Dado que los cristales de hielo que existen en la atmósfera, al caer a la tierra descienden en una posición al azar. En la simulación por computadora, la posición al azar de los cristales es representada mediante números aleatorios denotados por "Φ", pero a estos cristales les llegan rayos de luz que inciden en cada uno de ellos con un cierto ángulo llamado *ángulo*

de incidencia y son representados también por números aleatorios[37], denotados "θ_i", ahora bien, al incidir un rayo de luz con un cristal (prisma hexagonal tipo lápiz), el rayo emerge un tiempo después con un cierto ángulo de desviación denotado por "δ" y a su vez el rayo es separado en las diferentes frecuencias de colores que constituyen la luz.

Cada vez que un rayo de luz incide sobre un cristal con un ángulo "θ_i"se realizan operaciones con cada frecuencia de cada color, para tener así un ángulo de desviación total "δ_r"para el color rojo, un ángulo de desviación total "δ_v" para el color verde y un ángulo de desviación total "δ_a" para el color azul. El ángulo de desviación para el color rojo será menor que el ángulo de color verde y éste a su vez será menor que el ángulo de color azul, debido a que la luz roja se refracta menos que la luz verde y menos que la luz azul. Para fundamentar éste fenómeno se tiene la relación que dice que, en la mayoría de los dieléctricos transparentes (en este caso particular el hielo), cuando el índice de refracción (n) disminuye la longitud de onda (λ) aumenta y viceversa, es por eso que $\delta(\lambda)$ es menor para la luz roja que para la luz verde y ésta a su vez, es menor para la luz azul [34]. Para ilustrar mejor esta relación se muestra la tabla 5.3-2.

Tabla 5.3-2 Relación entre longitud de onda e índice de refacción.

Color	Longitud de onda (λ)	Índice de refracción (n)
Rojo	0.700	1.3069
Verde	0.550	1.3110
Azul	0.440	1.3163

Es por esta razón que al momento de ver el halo de 22°, el color rojo se visualiza más al centro de la imagen, posteriormente se visualiza el color verde y finalmente el color azul se visualiza más en la orilla de la imagen.

En resumen, una vez que se tiene δ_r, δ_v, δ_a para cada color, las operaciones que se realizan son las siguientes: se calcula la distancia "R" con la ecuación 5.4, se calcula el valor del radio "r" con la ecuación 5.5, después con las ecuaciones 5.6 y 5.7 se obtienen las coordenadas "*x,y*" para ubicar la posición exacta donde se localiza un punto, posteriormente se utilizan los valores de las coordenadas para calcular los valores digitales de "i,j" y así poder ubicar y pintar el pixel. El color del pixel está dado por la cantidad de luz de los colores que coincidan en una cierta posición. A continuación se explica más detalladamente la forma en como se hace la asignación correcta del color.

Para la representación digital de la imagen se tienen tres arreglos dinámicos para cada color: ***rojo***, ***verde** y ***azul***, en los cuales cada posición de cada arreglo se considera como una caja pequeña donde están almacenados la cantidad de rayos que coincidieron en una determinada posición, así por ejemplo, los rayos de color rojo se guardan en el arreglo *rojo, los rayos de color verde se guardan en el arreglo *verde, etc.

Cuando se obtiene la intensidad de cada color se realiza la siguiente operación: se recorren todas las posiciones de cada uno de los arreglos en las cuales se tiene guardada la cantidad

de rayos que coincidieron en una determinada posición y la operación que se realiza para cada posición (caja pequeña) de cada uno de los arreglos es la siguiente:

$$\frac{(\textit{número de rayos en la posción "}k\textit{"}) \quad * \quad 255}{(\textit{número máximo de rayos que incideron en una posición específica "}q\textit{"})} \tag{5.12}$$

donde, el valor de 255 se toma como la máxima intensidad que puede tener un pixel y a partir de este valor se hace una interpolación lineal para la asignación de la intensidad de cada pixel, por ejemplo, si en el arreglo ***rojo*** en la primera posición (primera caja) el valor de "k" es "1" y el valor de "q" es igual "1", el resultado será igual a 255, por lo tanto, el pixel tendrá la mayor intensidad, en cambio, si "k" es "5" y el valor de "q" es igual a "10" el resultado es 127.5 (se toma solamente la parte entera), entonces, la intensidad del pixel será menor y así sucesivamente para todos los pixeles. La misma operación se realiza para cada arreglo.

Una vez terminada la operación de asignación de intensidad de cada pixel en los tres arreglos, el resultado se guarda en uno solo, en el que finalmente se almacenará la información de la imagen resultante. El arreglo final se llama ***"colores"***, y a su vez esta información es enviada a un archivo, llamado ***"fimagen"***, el cual se podrá visualizar con algún programa especial para visualización de imágenes.

5.4 FORMATO DE LA IMAGEN

Al guardar los resultados finales de la aplicación que corresponden al halo de 22° deben guardarse con cierto formato, el formato debe tener como encabezado lo siguiente:

- La extensión de la imagen, que en este caso particular es "P6".
  fprintf(fimagen,"P6\n");
- Comentario.
  fprintf(fimagen,"#comentario en el encabezado de la imagen\n");
- Tamaño de la imagen.
  fprintf(fimagen,"%d %d \n",nrens,ncols);
- Intensidad máxima de un punto.
  fprintf(fimagen,"%d\n",255);

donde "fimagen", es el nombre del archivo en el cual se guardan los datos al disco duro. En este parámetro de sintaxis, el usuario puede darle el nombre que él desee.

Finalmente con la instrucción ***fwrite( )***, se manda a escribir al archivo "fimagen".

```
fwrite(colores,1,3*nrens*ncols,fimagen);
```

5.4.1 Visualización de la imagen

Como la aplicación se desarrolló en lenguaje C bajo sistema operativo Linux y distribución SuSE 7.3, en esta distribución se tiene un programa que se llama "XV" el cual se usa para visualizar manipular y convertir imágenes.

Para visualizar la imagen, se usa la siguiente sintaxis:

```
# xv [ruta del archivo[nombre del archivo]]
```

en el caso particular de la aplicación que se desarrolló, la sintaxis es como sigue:

```
# xv /root/programas_halos/halo22.dat
```

al ejecutar ésta sintaxis aparece la imagen.

Las imágenes que resultaron de la simulación para la aplicación desarrollada se muestran detalladamente en el Apéndice A.

5.5 RESULTADOS

Como se ha mencionado, la característica más importante de la herramienta MOSIX es la distribución de procesos entre los diferentes nodos que forman un cluster.

Para el caso particular del cluster MOSIX que se construyó y la aplicación que se desarrolló, se hicieron pruebas de distribución de cargas entre los dos nodos con que cuenta el cluster. La migración de procesos se puede realizar de dos formas, la primera es la distribución automática y la segunda es la distribución manual.

La distribución automática, consiste en que a partir del nodo principal o nodo maestro (definido por el usuario) se distribuyen todos los procesos a los demás nodos, la distribución se realiza al ejecutar varios procesos en el nodo principal, el primer proceso se ejecuta en éste, posteriormente los demás procesos al ser ejecutados son distribuidos automáticamente a los demás nodos que se encuentran ociosos. Ahora en el caso cuando se tiene un nodo "x" con mejores recursos que otro nodo "y", entonces, si el nodo "x" termina más pronto su tarea, el proceso que se ejecuta en el nodo "y" migra automáticamente al nodo "x" que tiene mejores recursos y de esta manera se ejecuta más rápido el proceso.

En la distribución manual, el usuario de acuerdo a criterios propios, hace la distribución de los procesos entre los diferentes nodos a partir del nodo principal. La manera detallada de cómo se puede hacer esta distribución se explica en los manuales [32].

La forma de observar y medir el comportamiento del cluster en la distribución, se realizó de la siguiente manera, la aplicación desarrollada se ejecutó dos veces en el nodo maestro (nodo numero 1, computadora Pentium III), con diferentes parámetros entrada, entonces

MOSIX automáticamente distribuye o migra un proceso al otro nodo ocioso (nodo número 2, computadora AMDK6-2).

La aplicación que se desarrolló para evaluar el funcionamiento del cluster es la simulación de la formación de "halos" y dado que son dos nodos los que forman el cluster MOSIX, entonces, los resultados que se presentan en las tablas 5.5-1 y 5.5-2, son los tiempos obtenidos durante la operación de distribución de procesos.

Tabla 5.5-1 Resultados en el nodo 1.

NODO 1		
Eje X. Número de cristales	Eje Y. Tiempo en minutos	Tiempo en horas
150,000	0.23	0.00
1'500,000	1.8	0.03
15,000,000	12.83	0.30
150'000,000	144.63	2.41
200'000,000	190.72	3.18

Tabla 5.5-2 Resultados en el nodo 2.

NODO 2		
Eje X. Número de cristales	Eje Y. Tiempo en minutos	Tiempo en horas
150,000	0.58	0.01
1'500,000	3.18	0.05
15,000,000	29.50	0.49
150'000,000	556.2	9.27
200'000,000	681.52	11.36

El comportamiento de los tiempos del cluster MOSIX en la distribución de procesos también se muestran gráficamente en la figura 5.5-1.

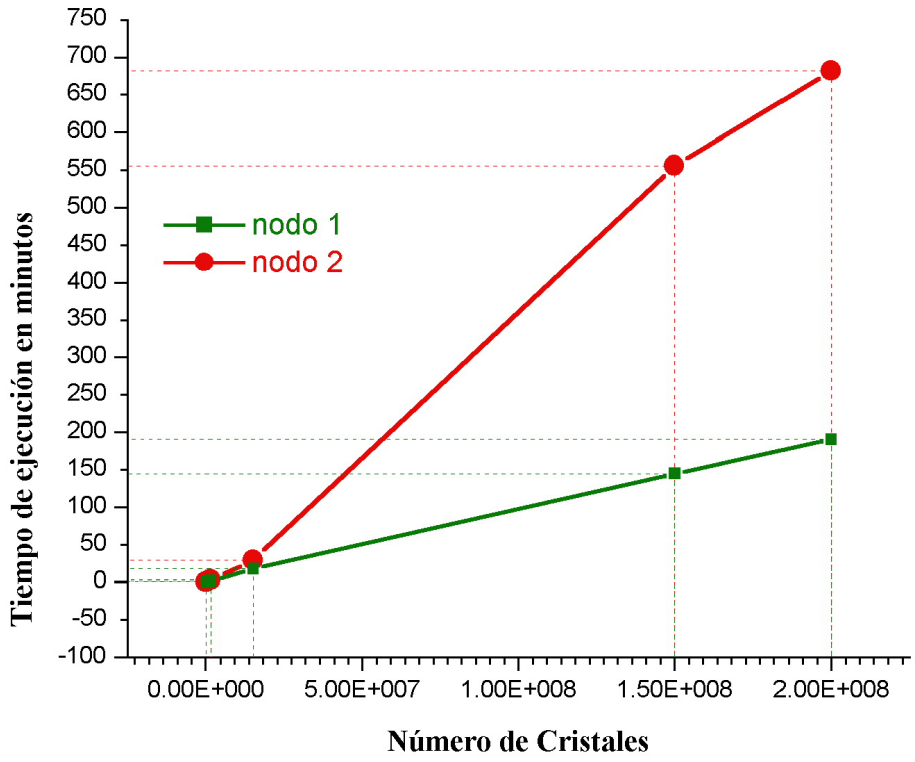


Figura 5.5-1 Comportamiento del cluster.

Como se puede notar las ventajas de utilizar un cluster MOSIX es distribuir varios procesos de manera simultánea, en este caso particular, es la distribución de la misma aplicación dos veces pero con diferentes parámetros, obteniendo así dos imágenes de halos al mismo tiempo. Cabe recalcar que lo que se desea mostrar es el rendimiento que se obtiene en MOSIX al distribuir los procesos de manera automática o manual, ejecutando simultáneamente tantos procesos como número de nodos tenga el cluster.

En el capítulo 4 sección 4.6.1, se mencionó que al ejecutar el monitor de carga de MOSIX **"mon -t"** aparecen enumerados los nodos que forman el cluster, además se pueden observar varios aspectos del comportamiento de los procesos en el momento de la ejecución y distribución.

A continuación se describe el comportamiento del cluster para la aplicación desarrollada (simulación de halos) tomando como base los datos mostrados en las tablas 5.5-1 y 5.5-2. Las figuras que muestran el comportamiento corresponden a interfaces propias de MOSIX.

La figura 5.5-2 muestra la carga en cada uno de los nodos.

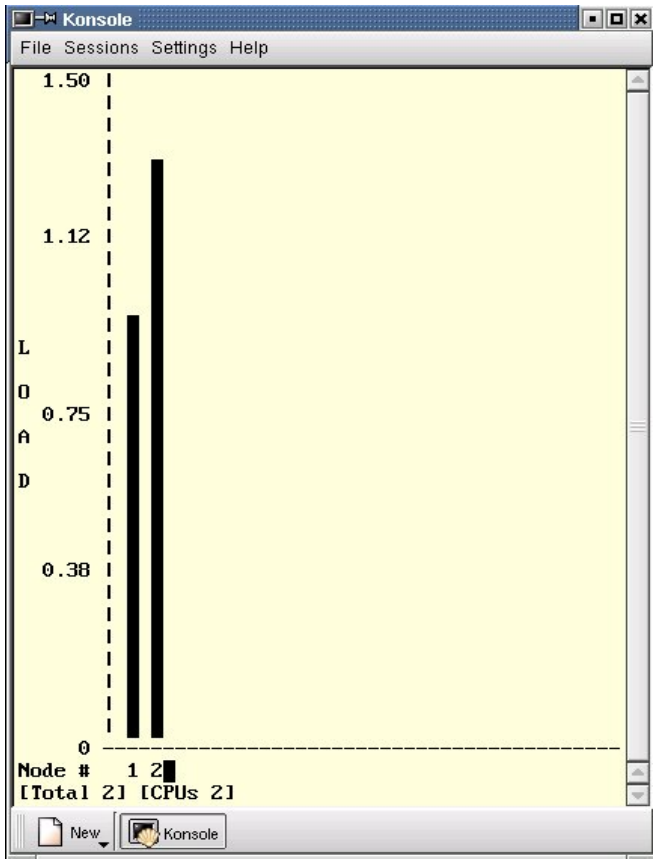


Figura 5.5-2 Carga de los procesos.

Ahora bien, en la figura 5.5-3 se muestra en Megabytes la cantidad de memoria que está usando cada proceso en cada nodo.

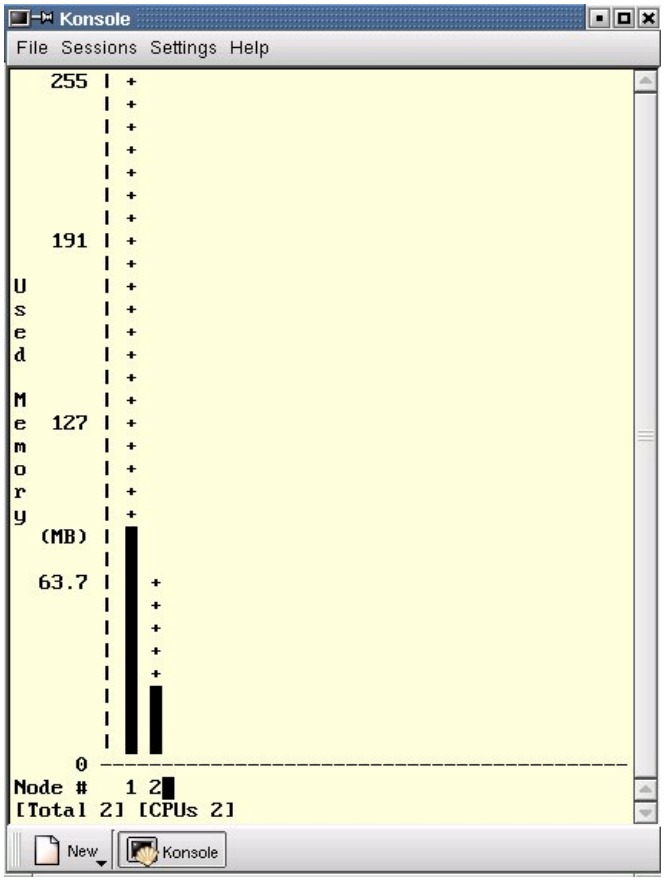


Figura 5.5-3 Memoria utilizada.

En la figura 5.5-4, se muestra en Megabytes también la memoria real que tiene cada nodo. Una forma de corroborar que en verdad MOSIX muestra la cantidad de memoria exacta que tiene cada nodo, se puede observar en la descripción de los recursos de cada máquina.

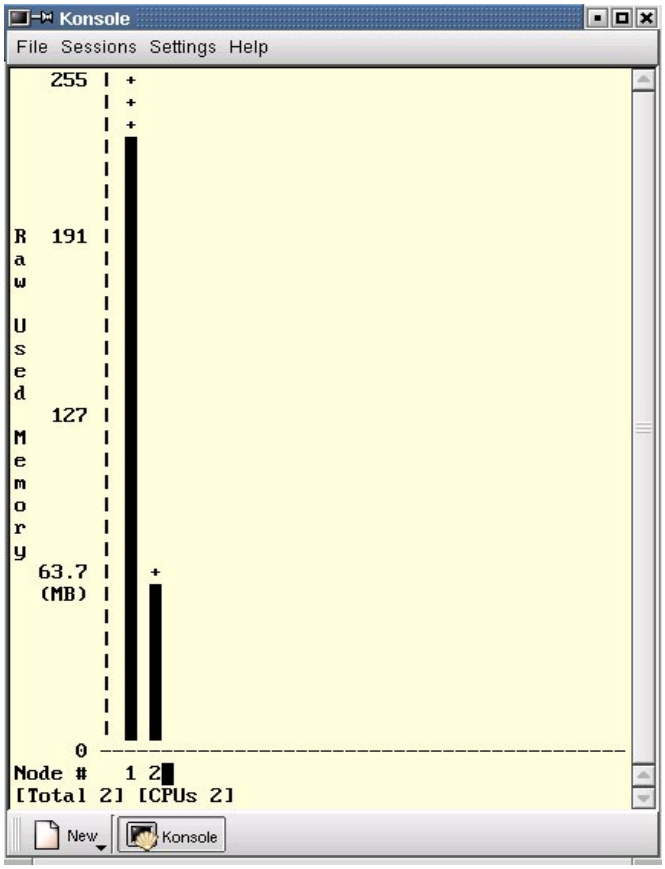


Figura 5.5-4 Memoria real.

En la figura 5.5-5, MOSIX muestra la velocidad de cada procesador en cada nodo.

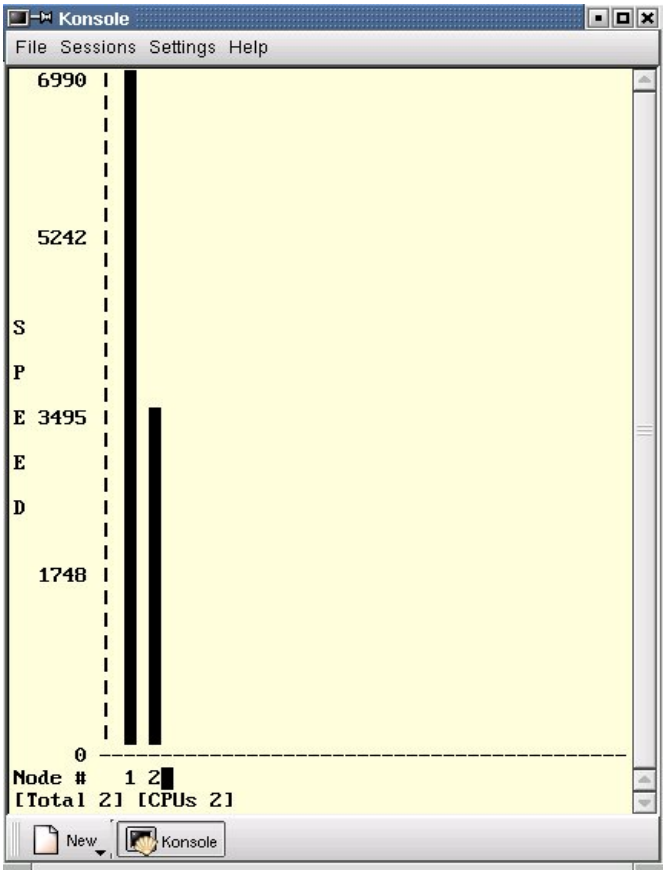


Figura 5.5-5 Velocidad de los procesadores.

Finalmente en la figura 5.5-6 se muestra el porcentaje de utilización de los procesadores de cada nodo.

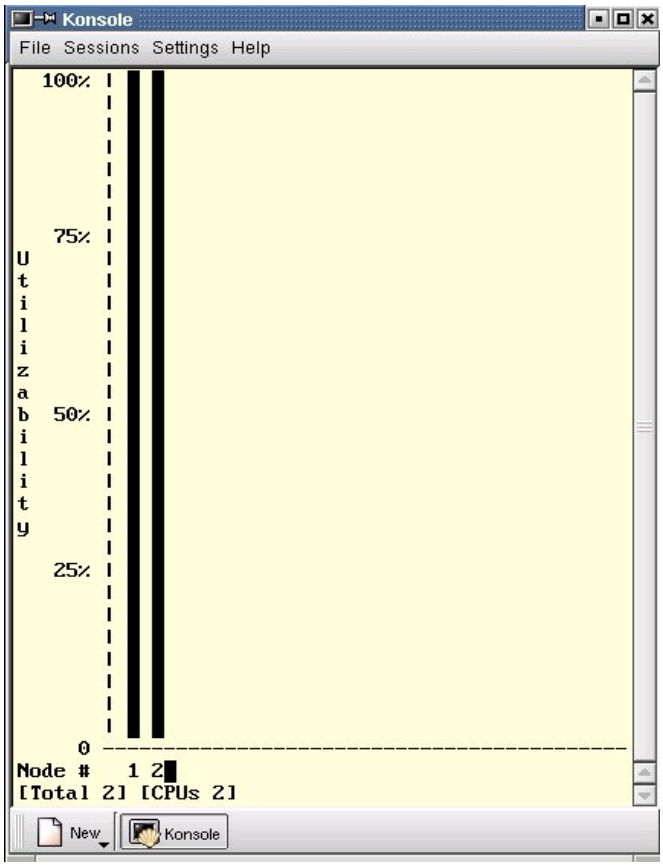


Figura 5.5-6 Utilización de cada procesador.

5.5.1 Descripción de los recursos de los nodos y material de interconexión del cluster

A continuación se describen los recursos de las computadoras (nodos) que se usaron en el cluster.

Tabla 5.5-3 Características de los nodos.

Nodo 1	Nodo 2
• Procesador Pentium III MMX a 700MHZ	• Procesador AMDK6-2 a 350 MHZ
• 256 MB de memoria RAM	• 64 MB en memoria RAM
• Disco duro de 40 Gb.	• Disco duro de 30 Gb.
• Sistema Operativo SuSE 7.3, kernel-2.4.17 y MOSIX-1.5.7	• Sistema Operativo SuSE 7.3, Kernel-2.4.17 y MOSIX-1.5.7
• Monitor ViewSonic 17”	• Monitor KDS 14”

El Material usado para la interconexión de cluster es el siguiente.

Tabla 5.5-4 Material.

Material de interconexión
• 3Com, OfficeConect Ethernet hub 4
• Cable par trenzado BELDEN categoría 5
• Conectores RJ45

5.5.2 Imágenes resultantes

Las imágenes, que se presentan a continuación (de la 5.5-7 a la 5.5-12), es una comparación entre una fotografía real de un halo de 22° y las imágenes que resultaron de la simulación de la aplicación en forma distribuida, para cada imagen se usó un número diferente de cristales. Como se puede observar, conforme se incrementa el número de cristales, la imagen que se forma es más perfecta, es decir, se asemejan más a la realidad.

La siguiente figura muestra una Fotografía real de un halo de 22°, tomada en el sur de Florida el 17 Mayo de 2002

Figura 5.5-7 Halo de 22°

Imágenes que resultaron de la simulación:

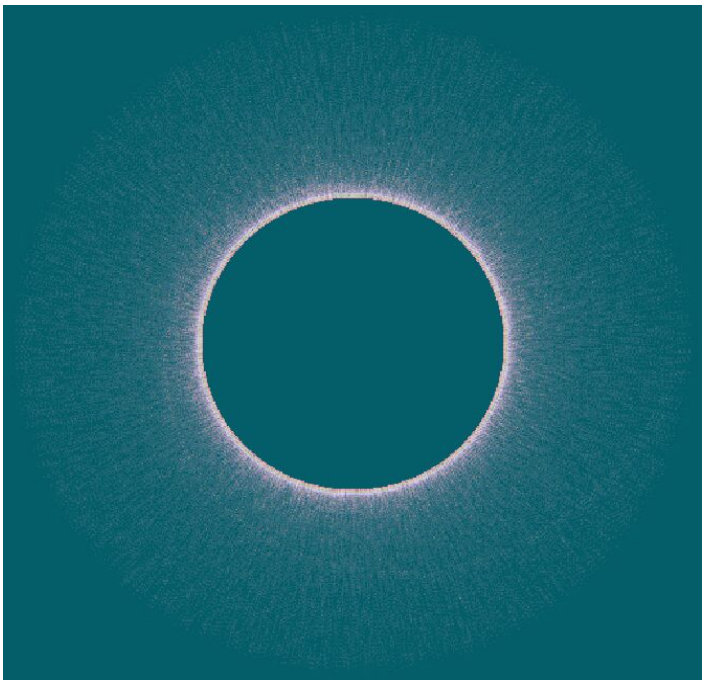


Figura 5.5-8 Figura con 150,000 cristales.

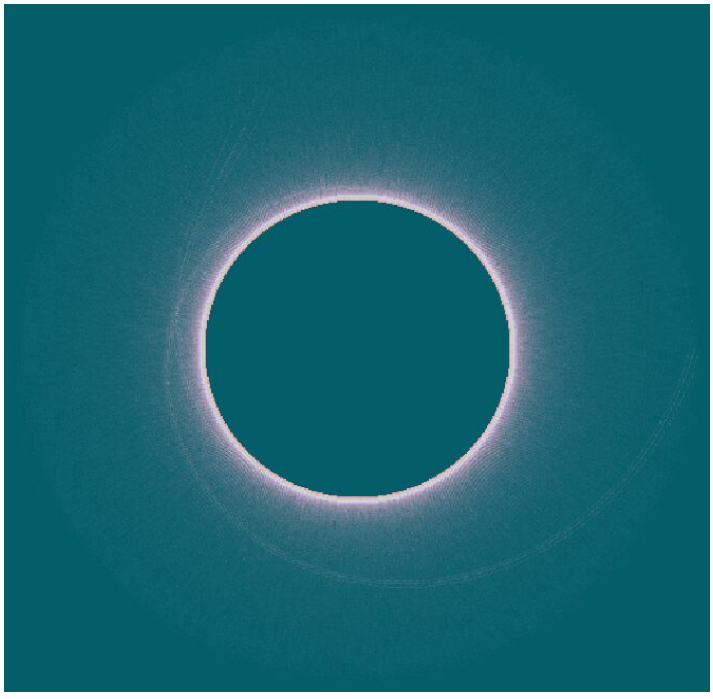


Figura 5.5-9 Figura con 1’500,000 cristales.

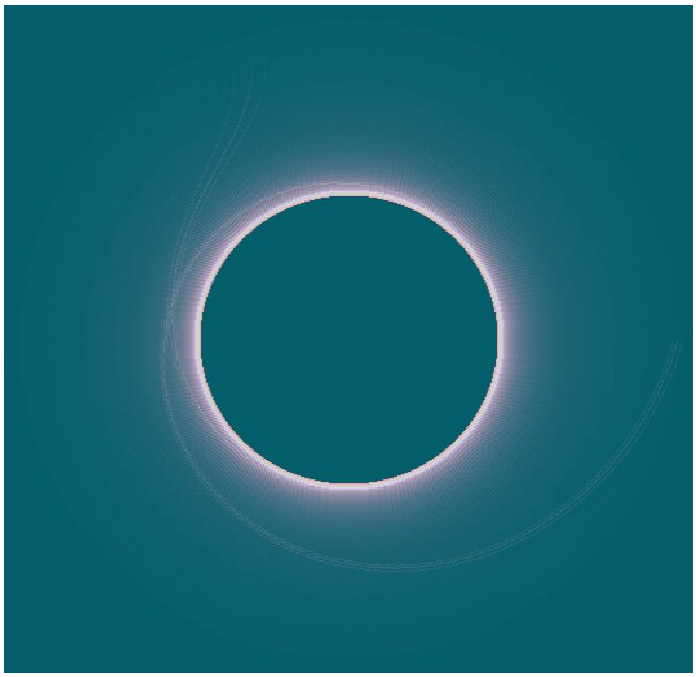


Figura 5.5-10 Figura con 15’000,000 cristales.

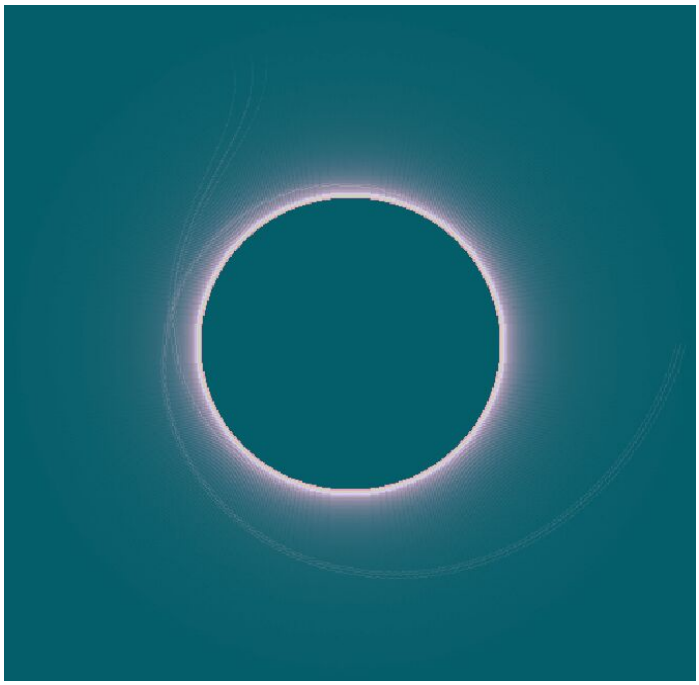


Figura 5.5-11 Figura con 150’000,000 cristales.

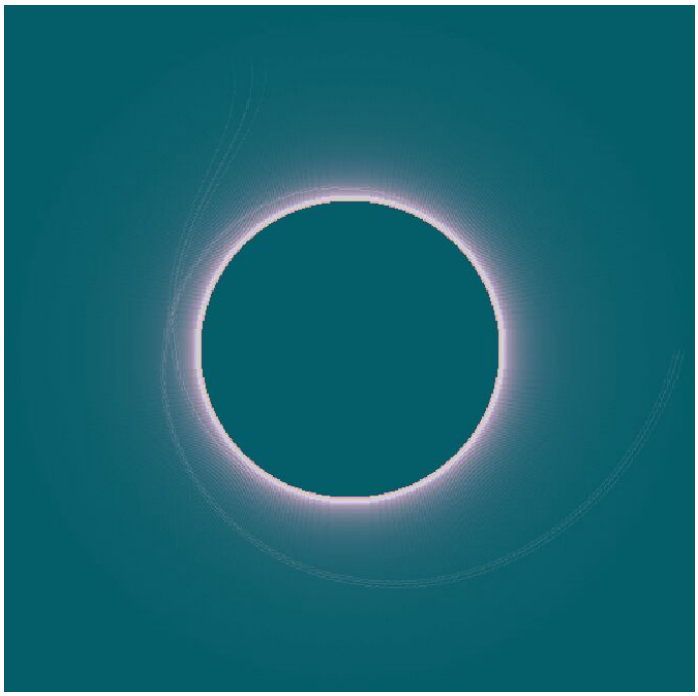


Figura 5.5-12 Figura con 200’000,000 cristales.

CONCLUSIONES

Los clusters en la actualidad han tenido un gran auge debido a que cada vez más se tiene la necesidad de resolver problemas que necesitan mucho tiempo de cómputo para llegar a un resultado.

Después de haber concluido el presente trabajo de tesis y haber alcanzado el objetivo de la misma, se concluye lo siguiente:

- Un cluster es una forma económica tanto en hardware como en software para hacer cómputo distribuido y paralelo. En hardware, porque se pueden adquirir PC's de alto rendimiento a un bajo costo, además el material para redes de alta velocidad que también es relativamente económico. En software porque se tienen muchas herramientas gratis, tales como el sistema operativo Linux, y herramientas tanto de programación como de monitoreo de procesos.

- Al usar la herramienta MOSIX no es necesario indicarle por código la forma de cómo debe hacerse la distribución debido a que MOSIX realiza la migración de manera automática y en caso de que el usuario quiera hacer la distribución de forma manual, no es el código fuente del programa lo que se tiene que modificar, si no que son ciertos archivos propios de la distribución de MOSIX los que se configuran para tal propósito, por lo que esto hace a MOSIX más amigable.

- La distribución de procesos en el cluster MOSIX que se construyó se realizó éxitosamente tal y como lo indica su propia literatura.

- Los clusters pueden tener una amplia gama de aplicaciones, por ejemplo, en cómputo científico, para la aplicación de métodos numéricos y técnicas de optimización para la solución de problemas, principalmente de la ciencia y la ingeniería. También en muchas aplicaciones que tienen gran impacto en la vida cotidiana tales como: diseños de fármacos, predicción de clima y dispersión de contaminantes.

- Los programas se pueden codificar en lenguaje Fortran en todas sus versiones, lenguaje C y C++.

TRABAJO FUTURO DE LA TESIS

Algunos de los trabajos posteriores a esta tesis son:

- Dado que sólo son dos nodos los que conforman el cluster, entonces un trabajo es aumentar el número de nodos para tener así mayor fuerza de cómputo y resolver el mismo problema en un tiempo más pequeño, no solamente éste problema de simulación de los halos, si no que también el cluster puede usarse para resolver otro tipo de aplicaciones tales como, formación de atmósferas, simulación de vuelos de aviones, simulaciones matemáticas, procesamiento digital de imágenes, entre otras.

- Otro trabajo futuro es implementar un cluster MOSIX de tal manera que se tengan máquinas remotas y que formen parte del cluster, además configurarlas para que en horas en que las computadoras están ociosas se agreguen al cluster aumentando así el poder de cómputo en las aplicaciones.

- En el caso de la aplicación que se desarrolló, si se quisiera continuar, otro trabajo futuro sería abarcar la simulación de los demás tipos de halos, utilizando diferentes modelos de representación digital, mayor número de longitudes de onda, considerar la posición de los cristales en tres dimensiones, utilizar todos los tipos de cristales que existen y si se usa el mismo cluster, aumentar el número de nodos.

GLOSARIO

RAID. Conjunto de dos o más discos que funcionan de forma conjunta, para poder aumentar el rendimiento y el nivel de protección de los datos.

Latencia. Lapso de tiempo necesario para que un paquete de información viaje desde la fuente hasta su destino. La latencia y el ancho de banda, juntos, definen la capacidad y la velocidad de una red.

Half duplex. Característica de un medio de comunicación por el cual no se pueden enviar y recibir datos simultáneamente. Por ejemplo, hablar por teléfono es un proceso de comunicación half-duplex, donde por momentos se habla y por momentos se escucha, pero donde se hace difícil establecer una comunicación si los dos participantes hablan a la vez

Full duplex. Característica de un medio de comunicación por el que se pueden enviar y recibir datos simultáneamente. A diferencia del half duplex que no se puede enviar y recibir datos al mismo tiempo.

Broadcast. Realiza una llamada general a todos los PC que puedan estar conectados a una red. Podemos tener nuestro programa escuchando y atender la llamada de distinta manera. Puede servir para tener solo una ejecución de nuestro programa en toda la red

GPL. (GNU General Public License), es un tipo de licencia sobre la propiedad intelectual en la cual únicamente se exige que aquellos desarrollos hechos con material licenciado bajo GPL sean a su vez GPL. Se trata de proteger la no ocultación de código.

Firmware. Son pequeños programas que por lo general vienen en un chip en el hardware, como es el caso de la ROM BIOS.
A estos programas permanentemente almacenados en la memoria de sólo lectura se les denomina firmware.

Broadcast. Es el envío de una señal desde un punto hacia todos los lugares que puedan recibir esa señal. El nombre de broadcast se origina en la radio y la televisión, y es característico de ellas.

DHCP. Dynamic Host Control Protocol, es un método de asignación de direcciones IP a computadoras.

LUI. The Linux Utility for cluster Install

MIPS. Millones de Instrucciones Por Segundo

Supercomputadoras. Término con el que se denota a las computadoras más rápidas o más poderosas. Cuentan con grandes recursos de memoria, disco, procesadores, redes de alta velocidad e interconexiones optimizadas.

TFTP. Trivial File Transfer Protocol, es un método de transferir archivos de un lado a otro.

NFS. Network File System, es un método para montar sistemas de archivos de un nodo a otro a través de la red.

BIBLIOGRAFIA

[1] A. S. Tanenbaum, **Sistemas Operativos Distribuidos**. Prentice Hall Hispanoamericana, S.A., México, 1996.

[2] George Coulouris, Jean Dollimore y Tim Kindberg, **Distributed Systems Concepts and Design,** Second Edition, Addison Wesley,

[3] H. M. Deitel, **Introducción a los Sistemas Operativos,** Addison-Wesley Iberoamericana, México, 1987

[4] Sterling Thomas, Salmon John, Becker Donald. **How to build a beowulf: a guide to the implementation and application of PC Clusters**. Cambridge: The MIT Press, 1998, 1999.

[5] **A Comparison of Queueing, Cluster and Distributed Computing Systems,** http://techreports.larc.nasa.gov/ltrs/94/tm109025.tex.refer.html

[6] **Metodologías de Paralelización en la Supercomputadora CICESE2000,** http://telematica.cicese.mx/computo/super/cicese2000/paralelo/

[7] **Sun Cluster Grid Architecture,** http://wwws.sun.com/software/grid/SunClusterGridArchitecture.pdf

[8] **Campus Clusters Based on Sun,** http://wwws.sun.com/software/cluster/wp-campuscluster/wp-campuscluster.pdf

[9] **TruCluster Server, Cluster Technical Overview,** http://www.tru64unix.compaq.com/docs/cluster_doc/cluster_51A/ACRO_DUX/ARHGVDTE.PDF

[10] **Meta Clusters,** http://www.byte.com/documents/s=1141/byt20010820s0002/

[11] **Linux Parallel Processing Using Clusters**, http://parallel.rz.uni-mannheim.de/Linux/parallel/ppcluster.html

[12] **The UNIX Operating System: A Robust, Standardized Foundation for Cluster Architectures,** http://www.fsmlabs.com/developers/docs/html/susv2/whitepapers/cluster.html

[13] **Clusters en México**, http://clusters.unam.mx/Historia

[14] **Installation from diskette boot**, http://sourceforge.etherboot.com

[15] L. Amar, A. Eizenberg and A. Shiloh. **The MOSIX Scalable Cluster File Systems for Linux,** July 2000, htt://www.mosix.org

[16] Y.Amir, B. Averbuch, A. Barak, R.S. Borstrom and A. Keren. **An Opportunity Cost Approach for Assignment and Reassignment en Scalable Computing Cluster**, proc PDCS'98, Oct. 1998, htt://www.mosix.org

[17] A. Barak, O. La'adan and Shiloh. **Scalable Cluster Computing with MOSIX for LINUX,** july 2000, Atlanta, Ga, 1999, htt://www.mosix.org

[18] A. Barak and O. La'adan. **The MOSIX Multicomputer Operating System for High Performance Cluster Computing,** Journal of Future Generation Computer Systems, 13(4-5):361-372, March 1998, htt://www.mosix.org

[19] **About the LUI project and download the source**, http://oss.software.ibm.com

[20] **Sys Admin, Remote Installation of Heterogeneous Linux Clusters Using LUI,** April 2001, volme10, number 4.

[21] **Condor Version 6.4.0 Manual,** http://www.cs.wisc.edu/condor/manual/v6.4/

[22] **Download Condor Binaries,** http://www.cs.wisc.edu/condor/downloads

[23] Armando Fox; Steven Gribble; Yatin Chawathe; Eric Brewer; Paul Gauthier. **Cluster-based scalable network services**. 1997.

[24] Eric Anderson; Dave Patterson. **Extensible, scalable monitoring for clusters of computers**. 1997, htt://www.mosix.org

[25] **MOSIX,** http://www.mosix.org/

[26] **Instalación y Uso del Cluster Beowulf en Recalcula,** http://www.cecalc.ula.ve/documentacion/tutoriales/beowulf/node56.html

[27] **The Linux Kernel Archives**, http://www.kernel.org/pub/linux/kernel/v2.4/

[28] **Linux Clustering with MOSIX,** http://www6.software.ibm.com/developerworks/education/l-mosix/l-mosix-ltr.pdf

[29] **Cómo recompilar el Kernel,** http://diskobolo.mat.ucm.es/personales/jdiaz/recompilakernel.html

[30] **Cómo se configura e instala el kernel,** http://www.hispafuentes.com/manuales/ldp/es/Kernel-Como/Kernel-Como-3.html

[31] **Cómo compilar un Kernel,** http://members.tripod.com/javara/linux7.html

[32] **README, MOSIX-1.5.7,** http://www.mosix.org

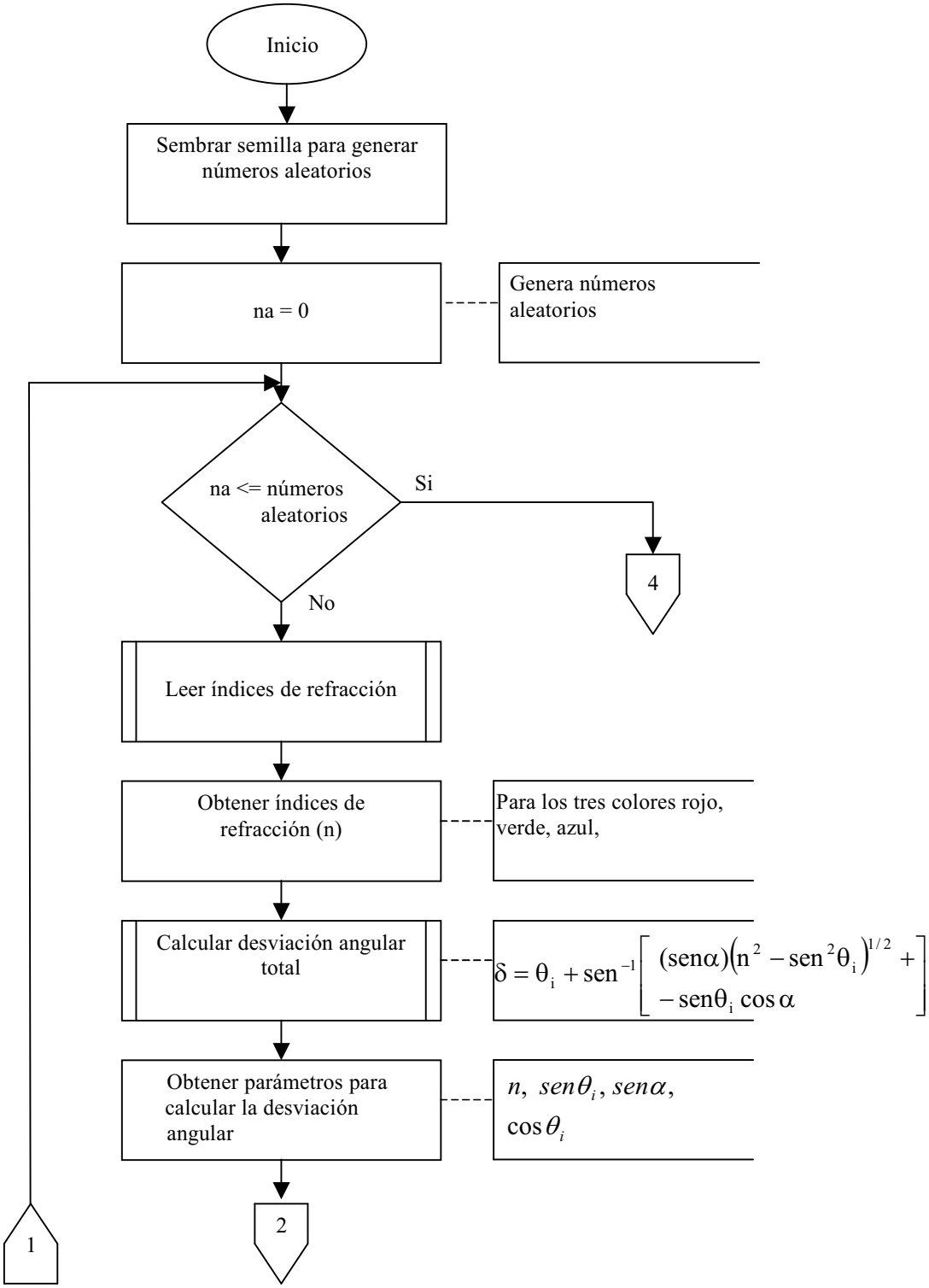
[33] Robert Greenleer, **Raimbows, Halos and Glories**, Cambridge University Press.

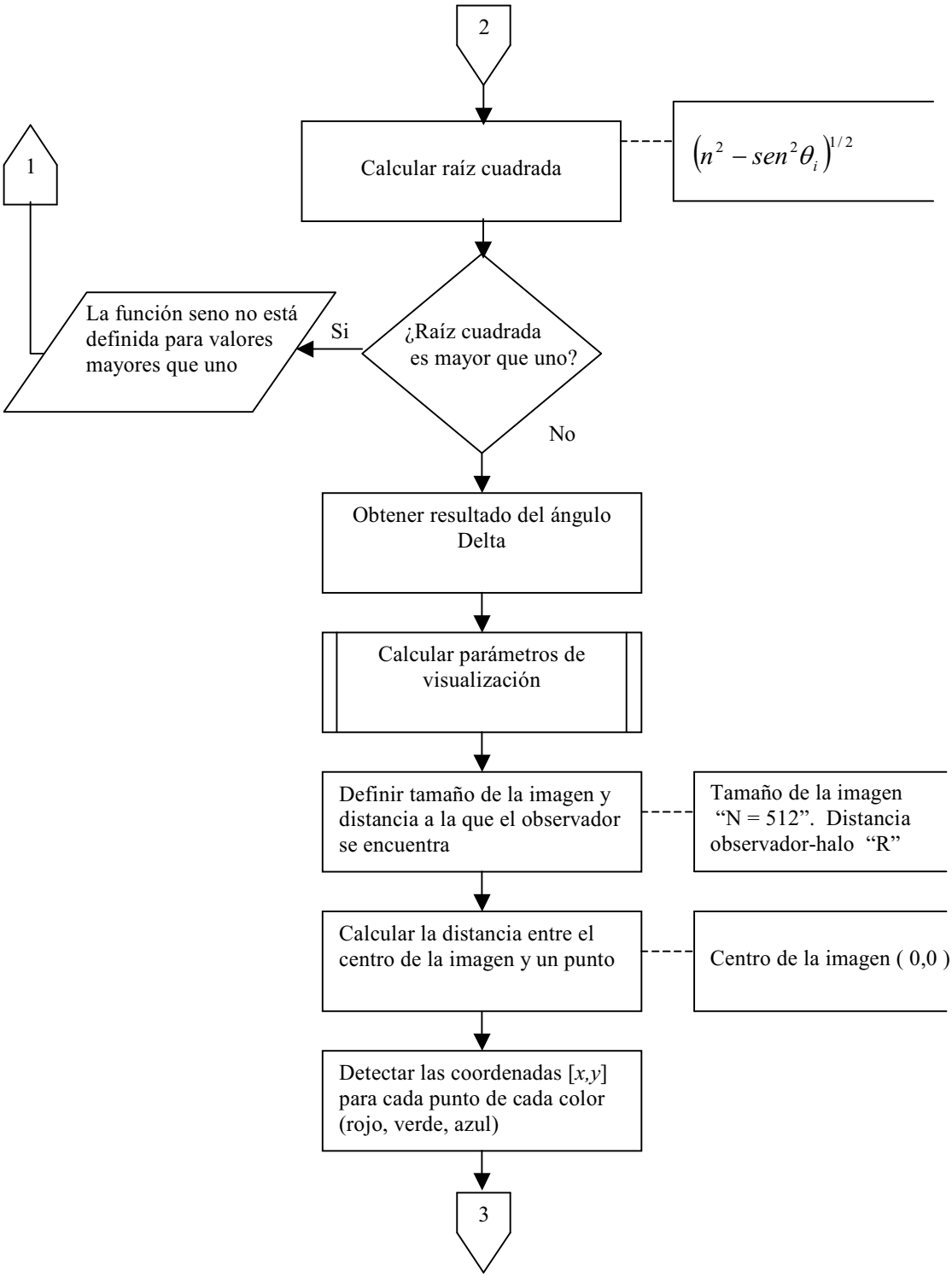
[34] Hecht, Zajac, **Óptica**, Addison Wesley Iberoamericana

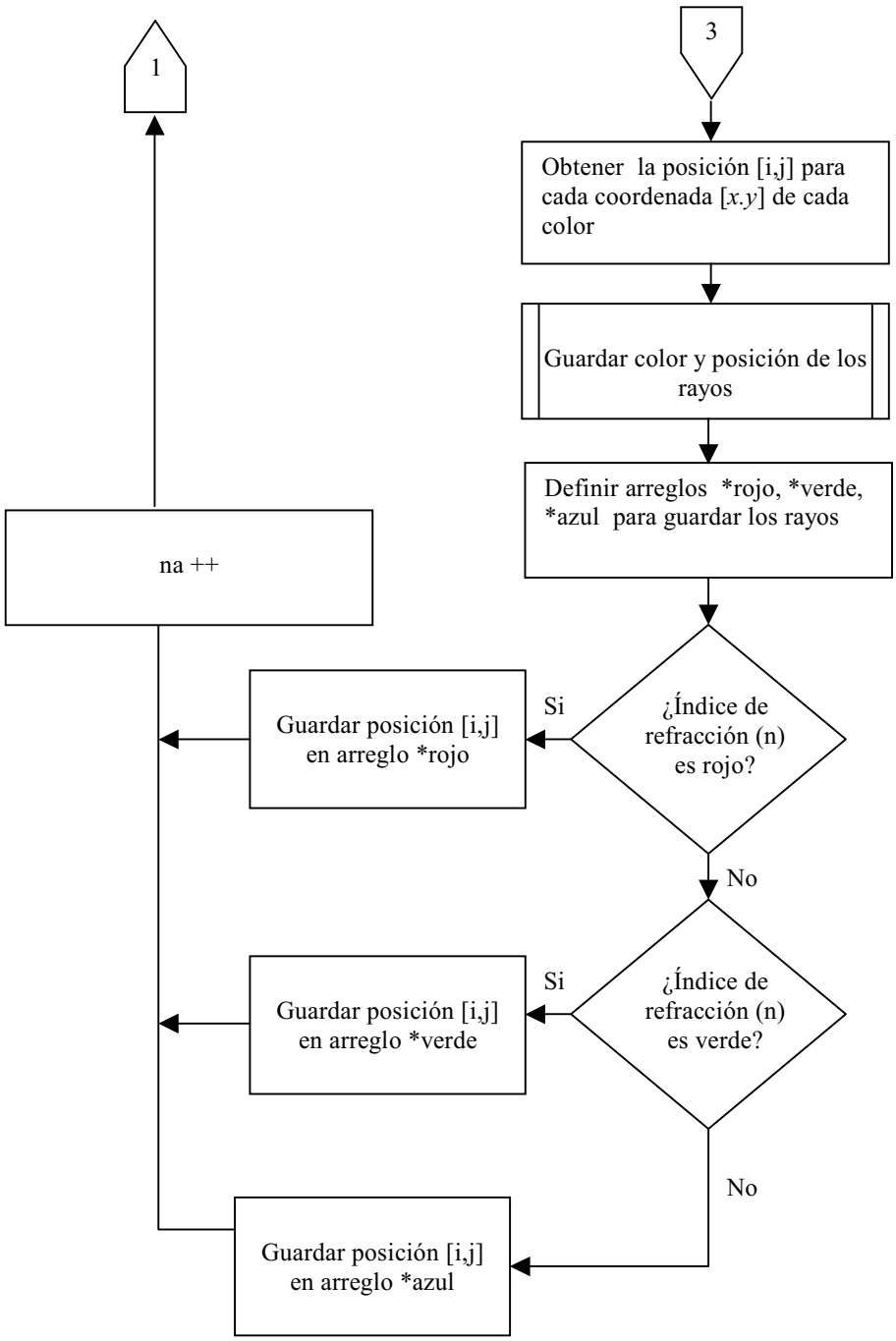
[37] **Numerical Recipes in C**, http://lib-www.lanl.gov/numerical/bookcpdf/c7-1.pdf

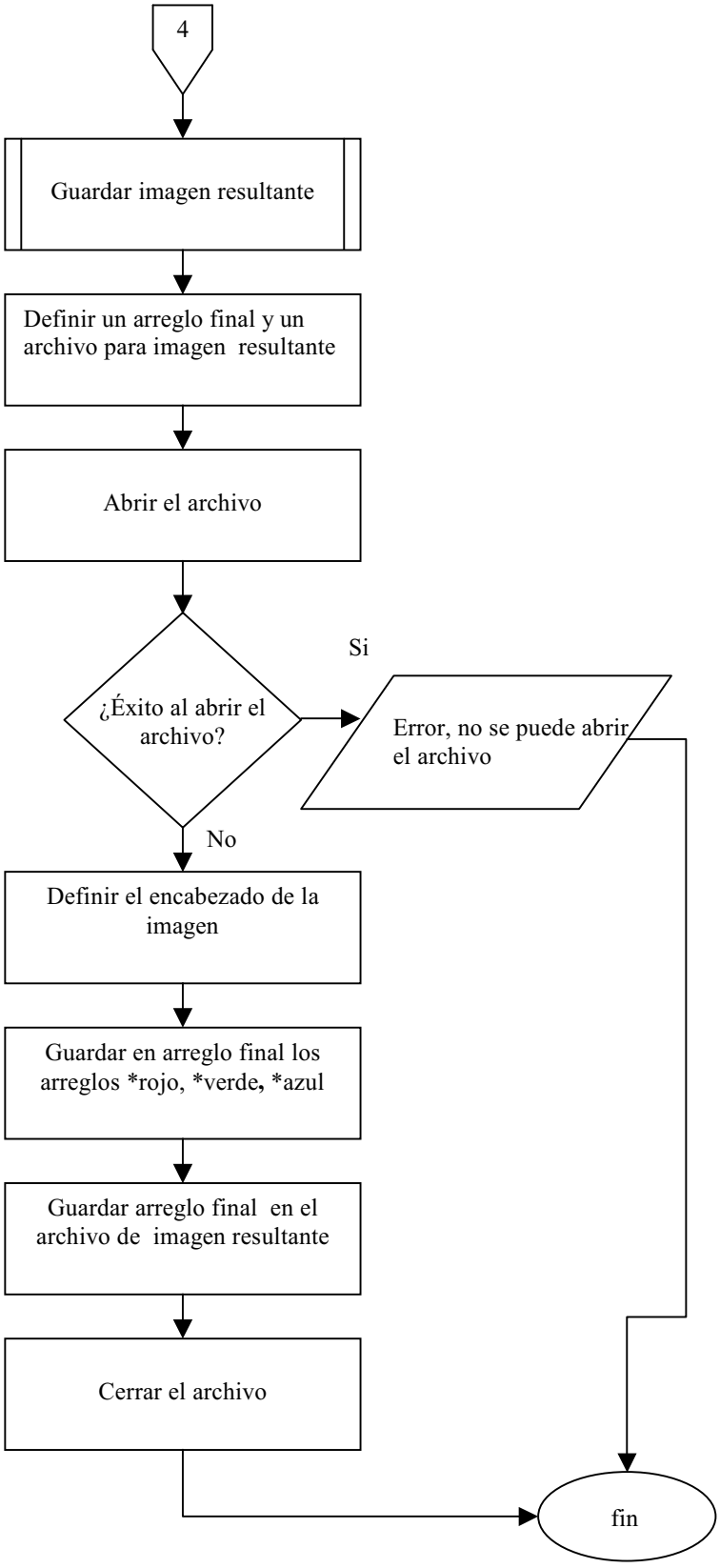
[38] Pradeep K. Sinha, **Distributed Operating System**, **concepts and design,** IEEE COMPUTER SOCIETY PRESS

APÉNDICE A
Diagrama de Flujo del programa de simulación de halos




2
Calcular raíz cuadrada
$\left(n^2 - sen^2\theta_i\right)^{1/2}$
1
La función seno no está definida para valores mayores que uno
Si
¿Raíz cuadrada es mayor que uno?
No
Obtener resultado del ángulo Delta
Calcular parámetros de visualización
Definir tamaño de la imagen y distancia a la que el observador se encuentra
Tamaño de la imagen "N = 512". Distancia observador-halo "R"
Calcular la distancia entre el centro de la imagen y un punto
Centro de la imagen (0,0)
Detectar las coordenadas [x,y] para cada punto de cada color (rojo, verde, azul)
3


1
3
Obtener la posición [i,j] para cada coordenada [x.y] de cada color
Guardar color y posición de los rayos
Definir arreglos *rojo, *verde, *azul para guardar los rayos
¿Índice de refracción (n) es rojo?
Si
Guardar posición [i,j] en arreglo *rojo
No
¿Índice de refracción (n) es verde?
Si
Guardar posición [i,j] en arreglo *verde
No
Guardar posición [i,j] en arreglo *azul
na ++


4
Guardar imagen resultante
Definir un arreglo final y un archivo para imagen resultante
Abrir el archivo
¿Éxito al abrir el archivo?
Si
Error, no se puede abrir el archivo
No
Definir el encabezado de la imagen
Guardar en arreglo final los arreglos *rojo, *verde, *azul
Guardar arreglo final en el archivo de imagen resultante
Cerrar el archivo
fin