PLATAFORMA EN EL WEB PARA COMPUTACIÓN PARALELA

EDWIN IGNACIO MUÑOZ

LUZ AIDA ROZO SANCHEZ

Director: Mag. HECTOR FABIO JARAMILLO ORDOÑEZ

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE TELEMÁTICA

POPAYÁN

2006

PLATAFORMA EN EL WEB PARA COMPUTACIÓN PARALELA

EDWIN IGNACIO MUÑOZ

LUZ AIDA ROZO SANCHEZ

Monografía presentada para optar al título de Ingeniero en Electrónica y Telecomunicaciones

Director: Mag. HECTOR FABIO JARAMILLO ORDOÑEZ

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE TELEMÁTICA

POPAYÁN

2006

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

PRESIDENTE DEL JURADO

_____

JURADO

Popayán, 2006

CONTENIDO

Pag.

LISTA DE FIGURAS

LISTA DE TABLAS

Capitulo 1 INTRODUCCIÓN

Cada día se está exigiendo mayor potencial de procesamiento de información no sólo en el ámbito científico, sino en otros ámbitos tales como el industrial y el comercial [ZAMORA2000]. Este incremento se debe a la búsqueda de un alto grado de precisión y velocidad al obtener resultados de tal forma que se puedan utilizar en modelos complejos que puedan ajustarse con mayor exactitud a la realidad (ej: modelos de código genético, simulación de condiciones climáticas, etc); esto acarrea la necesidad de tener gran capacidad para procesar flujos de datos de manera rápida y confiable, y un tipo de programación que se ajuste a esas necesidades. Por otro lado, la capacidad de procesamiento no siempre se encuentra disponible para su uso por parte de los grupos y personas dedicadas a la investigación [MUNDO2003]. Diferentes factores influyen en la poca disponibilidad de estos recursos, entre ellos la directa proporcionalidad entre el costo de los equipos y sus características; pero con el fin de disminuir los costos de adquisición de equipos más potentes nació la idea de utilizar computadores convencionales trabajando en paralelo para obtener capacidades de cómputo mayores [SPECTOR2000].

Utilizar computadores en paralelo implica una forma de programación diferente conocida como computación paralela. Esta forma de programación inicia su trayectoria desde hace varias décadas con los grandes supercomputadores de compañías como IBM, Control Data, Data General y Digital Equipment Corporation, cuyo mercado estaba dirigido a entidades gubernamentales, científicas, petroleras, y de seguridad [SPECTOR2000].

En los últimos años, un conjunto de factores como la mejora en el desempeño de los computadores personales, los desarrollos en comunicaciones entre los equipos en red, la disminución de costos de los equipos de interconexión y el acceso al software libre, llevaron al desarrollo y construcción de sistemas de bajo costo que utilizan las ventajas

del procesamiento en paralelo [BEOW2001]. Estos sistemas, conocidos como clusters, se forman con equipos, como computadores personales, comunicados entre si por medio de una red y empleando software que les adiciona la capacidad de trabajar en forma conjunta y paralela; en algunos casos pueden lograr desempeños comparables a los obtenidos con supercomputadores construidos por las empresas tradicionales [STERLING2005]. La primera de estas implementaciones bajo Linux fue desarrollada por Donald Becker y Thomas Sterling para la NASA. Este cluster se basó principalmente de componentes y periféricos de la arquitectura x86 [CARDEN2005].

A partir de este proyecto han surgido numerosas iniciativas en este sentido, utilizando clusters para tareas que requieran enormes cantidades de cómputo: simulaciones científicas, gráficos, modelado meteorológico.

Algunos centros de investigación y universidades internacionales y nacionales han utilizado sistemas clusters; algunas han comprado el sistema completo a vendedores de equipos, mientras que otros han empleado los equipos que ya han adquirido, de forma que la capacidad de computación que obtienen sea la necesaria para sus investigaciones y desarrollos [PLAZA2002].

En Latinoamérica encontramos cluster en Venezuela, México y Chile, en Colombia las universidades que han trabajado con este tipo de sistemas son la Universidad Nacional de Colombia, la Universidad de Antioquia, la Universidad de los Andes, la Universidad Sergio Arboleda y el Observatorio Astronómico. Existe un interés creciente sobre el tema en el país, afianzado por el desarrollo de las redes de Alta Velocidad como *Renata* que esta promoviendo la colaboración entre los grupos de investigación en Colombia para aprovechar los recursos computacionales formando Mallas (Grids) con los cluster y otros elementos computacionales de las instituciones.

Los vendedores de equipos de cómputo concientes del impacto de estos sistemas ofrecen clusters con su propio hardware y software; pero al igual que lo hicieron los primeros constructores de cluster, es posible configurarlos con pocos recursos empleando equipos que ya no están en uso y software libre disponible en Internet. La desventaja de este software es que es necesario descargarlo y tiene una configuración dispendiosa ya que

por lo general trabaja con Sistemas Operativos de código abierto[1] que muchas personas no están habituadas a emplear, además una vez se configura el cluster, para programarlo los investigadores deben desplazarse al lugar en el que se encuentra. Por eso surge la idea de realizar un proyecto que acerque a la comunidad a estos sistemas sin la necesidad de realizar la configuración completa del cluster, ni de desplazarse al lugar en el que se encuentra. Al realizar este acercamiento también es necesario considerar la programación paralela pues es poco conocida en nuestro entorno, este concepto no se limita únicamente a grandes sistemas de cálculo, aunque es una de sus aplicaciones más conocida; en la práctica todos los computadores que existen en el mercado de hoy explotan de una u otra manera soluciones paralelas transparentes a los usuarios [OPENM2004].

Por todo lo anterior, el objetivo central del proyecto es dar una visión general de los principales aspectos relacionados con la computación en paralelo y del manejo de los cluster de computadores que explotan este concepto, además explicar los principales aspectos a tener en cuenta en la configuración de un cluster de computadores que permita la programación en paralelo y las consideraciones de diseño en el desarrollo de una aplicación Web que actúe como un “puente” que permita la ejecución de código de programas paralelos en un sistema tipo Cluster, despliegue los resultados obtenidos sin que el equipo desde el que se accede sea un nodo del sistema, y que brinde información acerca de la computación en paralelo aportando en el acercamiento de la comunidad Universitaria a los sistemas de computación paralela y a la generación de espacios propios de divulgación de información relacionada a través de la Web.

Para explicar los aspectos referentes a la construcción de un cluster se propone construir un sistema de este tipo que sirva para realizar las pruebas con la aplicación Web. La construcción de esta aplicación está limitada por el hardware que suministra la Universidad y el software de código abierto empleado. El presente documento constituye un referente acerca de la computación paralela específicamente aplicada a sistemas cluster de bajo costo; y se mencionan algunos sistemas paralelos de propósito específico y soluciones comerciales propietarias.

[1] Es el término por el que se conoce al software distribuido y desarrollado en forma libre.

Se utiliza como base metodológica el Modelo de Construcción de Soluciones (MCS), adecuándolo a las características inherentes del proyecto. Para la implantación adecuada de esta metodología se construye una documentación para el seguimiento y la escalabilidad del proyecto y que permite que la aplicación pueda ser ampliada en otros trabajos de grado. Esta documentación se presenta empleando UML y constituye uno de los anexos de la monografía.

La organización del documento es la siguiente:

En el capítulo 2 titulado *Aspectos Generales* contiene conceptos empleados en la computación paralela, como métricas de rendimiento, algoritmos empleados, modelos de programación y comunicación.

En el capítulo 3 titulado *Arquitecturas multiprocesador* se encuentran características de las arquitecturas paralelas más empleadas y configuraciones que utilizan la programación paralela.

En el capítulo 4 titulado *Construcción de un cluster de computadores* se explican los aspectos a tener en cuenta en la construcción de un cluster de computadoras de bajo costo, y se describe el hardware y el software necesario.

En el capítulo 5 titulado *Ambiente Experimental* se describen diferentes pruebas realizadas sobre el cluster construido y sobre la aplicación Web.

En el capitulo 6 titulado *Conclusiones* se encuentran las conclusiones inferidas del trabajo realizado y las recomendaciones para posibles trabajos futuros.

Los *Anexos* contienen la descripción de la instalación de los programas empleados en el desarrollo del proyecto y la documentación de la aplicación Web.

Capitulo 2 ASPECTOS GENERALES

La computación paralela consiste en la ejecución simultánea de una misma tarea particionada sobre múltiples elementos de procesamiento para obtener resultados más rápidos. Para su implementación es necesario que la información sea tratada mediante procesamiento paralelo. Esta forma de procesamiento generalmente se utiliza en la resolución de problemas complejos y/o con grandes cantidades de información. Debido a su diferencia fundamental con la computación secuencial es necesario definir un grupo de conceptos particulares para comprender como se explota este tipo de computación [OPENM2004].

2.1. Conceptos Principales de Computación Paralela

La Computación Paralela es el estudio sistemático de algoritmos y procesos que describen y modifican la información, capaces de ejecutarse de manera simultánea sobre varios elementos de procesamiento. Este estudio cobija los aspectos relacionados con la teoría, análisis, diseño, eficiencia, implementación y aplicación de algoritmos y procesos [WIKI2005].

A continuación se muestran algunas definiciones que se consideran útiles para entender las ideas fundamentales de la computación paralela y algunos conceptos traídos de la computación secuencial requeridos para entender y complementar estas ideas.

Elemento de Procesamiento: Es un elemento con la capacidad de ejecutar de manera secuencial (uno detrás del otro) los pasos que forman un algoritmo. Su implementación básica en sistemas electrónicos es el procesador, aunque también es

posible desarrollar elementos de proceso mediante dispositivos mecánicos, químicos, biológicos, etc. [WIKI2005]

Procesamiento Paralelo: Se define como la acción de modificar (o procesar) información mediante un conjunto de elementos de proceso que ejecutan una tarea (entidades de ejecución) de manera simultánea. [WIKI2005].

Descomposición: proceso de dividir un cálculo en partes más pequeñas que puedan ser ejecutadas en paralelo [GRAMA2003].

2.1.1. Tareas y Granularidad

Las tareas son unidades de cómputo definidas por el programador en las cuales se subdivide la solución de un problema. La ejecución simultánea de múltiples tareas disminuye el tiempo para resolver todo el problema. Algunas tareas pueden usar datos que resultan de la ejecución de otras tareas, para expresar esas interacciones se utiliza un *gráfico de dependencia de tareas*, este consiste en un grafo en el cual los nodos representan las tareas y las aristas dirigidas indican la dependencia entre ellas. Estos gráficos se emplean en el diseño de programas paralelos [GRAMA2003]. En la figura 2.1 se observa un gráfico de dependencia, en este la tarea 5 depende de las tareas 1 y 2, la tarea 6 de 4 y 3; y la 7 de 5 y 6. Este gráfico se obtiene a partir de una consulta sobre la tabla 2.1.

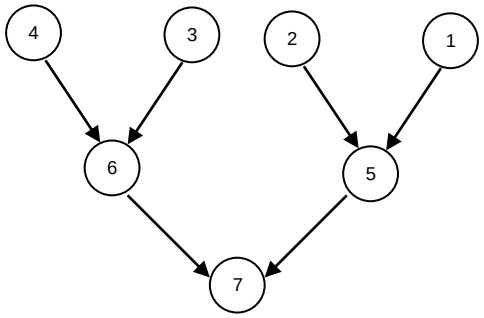


Figura 2.1. Grafico de dependencia de tareas [GRAMA2003]

ID#	Modelo	Año	Color	Comerciante	Precio
4523	Civic	2002	Azul	MN	$18,000
3476	Corolla	1999	Blanco	IL	$15,000
7623	Camry	2001	Verde	NY	$21,000
9834	Prius	2001	Verde	CA	$18,000
6734	Cívic	2001	Blanco	OR	$17,000
5342	Altima	2001	Verde	FL	$19,000
3845	Maxima	2001	Azul	NY	$22,000
8354	Accord	2000	Verde	VT	$18,000
4395	Civic	2001	Rojo	CA	$17,000
7352	Civic	2002	Rojo	WA	$18,000

Tabla 2.1. Información de vehículos [GRAMA2003]

La tabla 2.1 contiene información de vehículos, cada fila es un registro de varios campos que contiene los datos correspondiente a un vehículo en particular, tales datos son ID, modelo, año, color, Comerciante y Precio. Si se desea encontrar todos los vehículos Civic del 2001 cuyo color es verde o blanco una forma de hacerlo es creando tablas intermedias. Una posible opción es crear cuatro tablas: una conteniendo todos los vehículos marca Civic (esto es representado por la tarea 4 en la figura 2.1), otra con todos los automóviles modelo 2001 (tarea 3 figura 2.1), otra que tenga todos los automóviles de color verde (tarea 2 figura 2.1) y una que contenga todos los automóviles de color blanco (tarea 1 figura 2.1). Después se toman parejas de tablas y se obtiene la intersección de la tabla de vehículos Civic con la tabla de los modelo 2001, para construir la tabla de todos los civic modelo 2001 (tarea 6 figura 2.1). Similarmente con la unión de la tablas de vehículos de color verde y la de los de color blanco se forma una tabla que almacena todos los automóviles cuyo color es verde o blanco (tarea 5 figura 2.1). Finalmente, se halla la intersección de la tabla que contiene todos los Civic modelo 2001 con la tabla que almacena a todos los vehículos verdes o blancos, lo que da el resultado deseado (tarea 7 figura 2.1). El proceso se observa en la figura 2.2.

El número y tamaño de las tareas en la cuales se descompone un problema determina la granularidad de la descomposición. Una descomposición en un gran número de tareas pequeñas es llamada de *grano-fino* y una descomposición en un pequeño número de tareas grandes es llamada de *grano-grueso* [GRAMA2003].

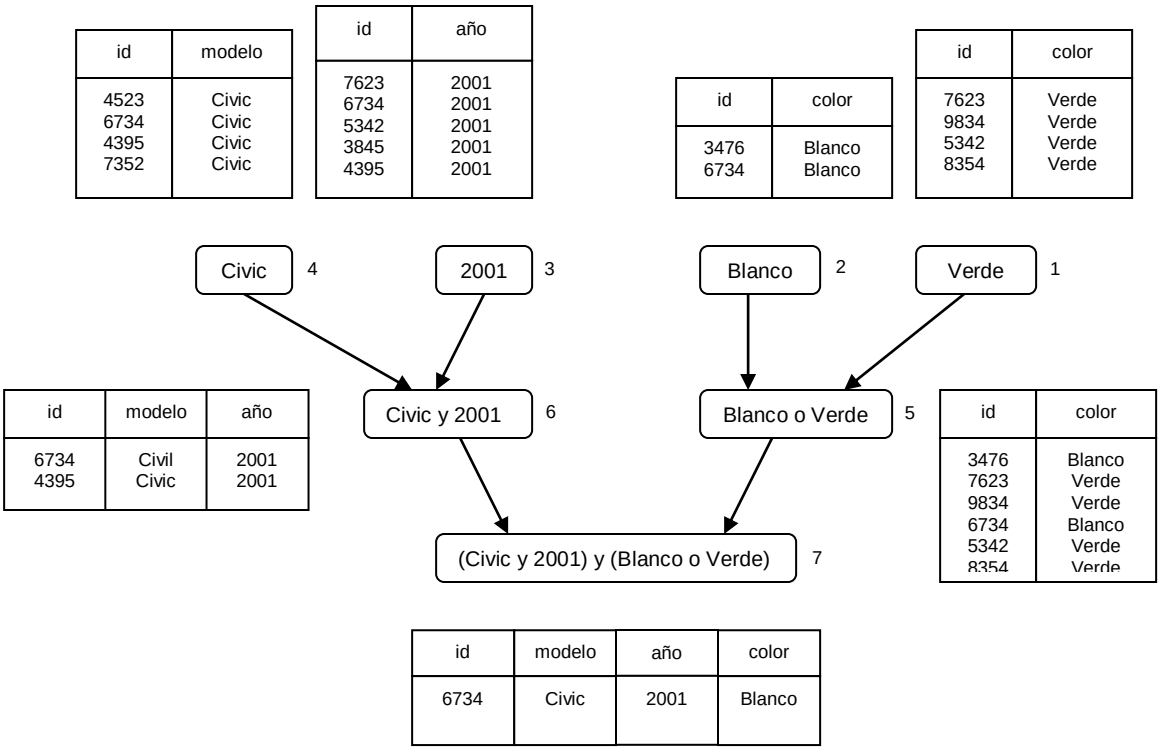
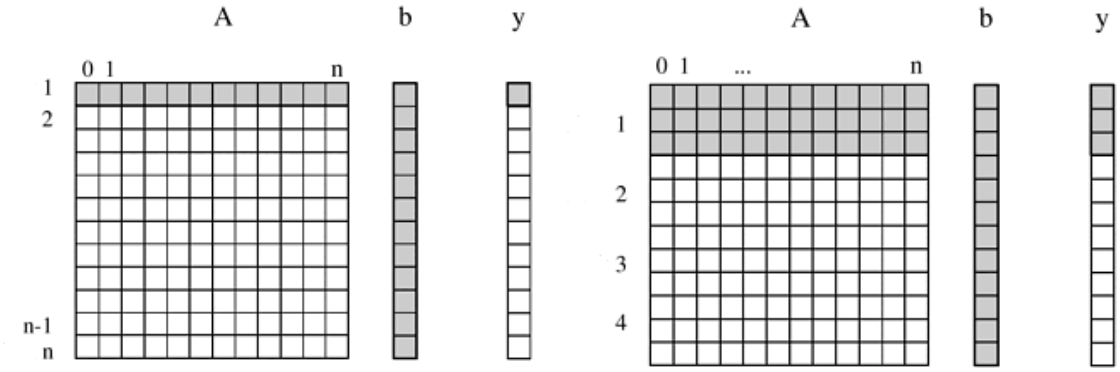


Figura 2.2. Estructura de la consulta [GRAMA2003]

Algunos autores relacionan la granularidad de grano-grueso con la ejecución de programas en paralelo y la de grano-fino con la ejecución de operaciones o de bucles en paralelo; para otros la granularidad depende de la definición que se le de al problema, por ejemplo la multiplicación de una matriz A de n x n con un vector b como se muestra en la figura 2.3, donde las filas sombreadas representan los elementos que se operan en cada tarea. En este problema se considera como granularidad fina que en cada tarea se opera una fila de A con el vector b como se muestra en la figura 2.3a; mientras que la figura 2.3b cada tarea opera 4 filas de A con el vector b y se considera granularidad-gruesa.

Grado de paralelismo: Máximo número de entradas que un sistema puede ejecutar en paralelo [ANGUITA2005].



a. Granularidad Fina. b. Granularidad Gruesa

Figura 2.3. Multiplicación de matriz A con vector b y el resultado es el vector Y [GRAMA2003]

2.1.2. Entidades de Ejecución

Las entidades de ejecución pueden variar de acuerdo a la arquitectura de procesamiento paralelo que se utilice, las que se emplean en sistemas multiprocesadores y multicomputadores son [ANGUITA2005]:

- **Proceso** : Es un conjunto formado por las instrucciones de un programa destinadas a ser ejecutadas por un elemento de procesamiento, su estado de ejecución en un momento dado, su memoria de trabajo y la información de los recursos que tiene en uso y su prioridad de ejecución [PROCESO2006]. Un proceso puede estar conformado por uno o más hilos.

- **Hilo**: Establece un camino de ejecución independiente, que tiene su propia pila y grupo de registros, pero comparte la memoria de programa, las variables globales y otros recursos con los hilos del mismo proceso. Estas características hacen que los hilos se puedan crear y destruir en menos tiempo que los procesos y que la comunicación, sincronización y conmutación entre hilos sea más rápida que entre procesos. Un hilo tiene una granularidad menor que un proceso.

Durante la ejecución de programas paralelos pueden surgir una serie de inconvenientes que se presentan debido a las dependencias, estas son puntos donde el código de un algoritmo depende de los resultados de alguna acción en otra parte del código. Para una aplicación paralela, las dependencias provocan que algunas partes del código deban ser ejecutadas de manera secuencial o que una tarea deba esperar a otra. Las dependencias tienen dos formas: Dependencias de datos y Dependencias de control [SPECTOR2000].

- **Dependencias de datos**: Existen donde alguna operación no puede ejecutarse hasta que el dato se encuentre disponible como un resultado de otra operación, por ejemplo:

```
i = b + 2*4;
j = 24 * i;
```

  para ejecutar j se necesita el resultado de la operación i.

- **Dependencias de control**: Ocurre cuando hay una sentencia anidada dentro de otra condicional, por ejemplo:

```
si ( a==1 )
    para (i=1;i<=max;i++)
          instrucciones....
          fin_para
    fin_si
```

  La ejecución del `bucle para` depende del resultado de la condición *`si`*.

La forma en que se solucionan las dependencias es a través de la sincronización de su ejecución, esto es, coordinar las entidades de ejecución de tal forma que el programa finalice exitosamente [GRAMMA2003].

2.2. Consideraciones generales de Sistemas Paralelos

Al asignar el doble de hardware a la solución de un problema, se espera que un programa se ejecute el doble de rápido, sin embargo, en programas paralelos típicos casi nunca sucede, debido a una variedad de situaciones asociadas con el paralelismo por las cuales la solución depende del algoritmo y la arquitectura paralela empleada, por eso es importante conocer las necesidades que generan los programas paralelos y cómo cuantificar los aspectos generales relacionados con un sistema paralelo en particular.

2.2.1. Necesidades de procesamiento adicionales generadas por los programas paralelos

A continuación se describen las necesidades que se deben satisfacer para implementar soluciones sobre un sistema paralelo.

Interacción entre procesos: Cualquier sistema paralelo requiere que los elementos de proceso interactúen e intercambien datos, para esto se deben destinar recursos y tiempo de ejecución, que representan una pérdida de eficiencia en el procesamiento paralelo.

Tiempo en espera: Los elementos de proceso en un sistema paralelo pueden estar ociosos durante algunos periodos de tiempo debido a muchas razones tales como: sincronización, desproporción de la carga de proceso, partes de un algoritmo que no sean paralelizables, lo que hace que sólo puedan ser ejecutadas en un elemento de proceso [GRAMA2003]. En la figura 2.4 se observa una distribución típica de los requerimientos de tiempo durante la ejecución de un programa paralelo (esencial y adicional), los tiempos de comunicación y de espera.

Computación adicional: La diferencia en la computación ejecutada por el programa paralelo y el mejor programa secuencial[2] es lo que se conoce como computación adicional; es el procesamiento que adiciona el programa paralelo en su ejecución

[2] Programa secuencial más rápido para un problema determinado.

incluyendo tareas como la comunicación entre los nodos y el control de la ejecución.

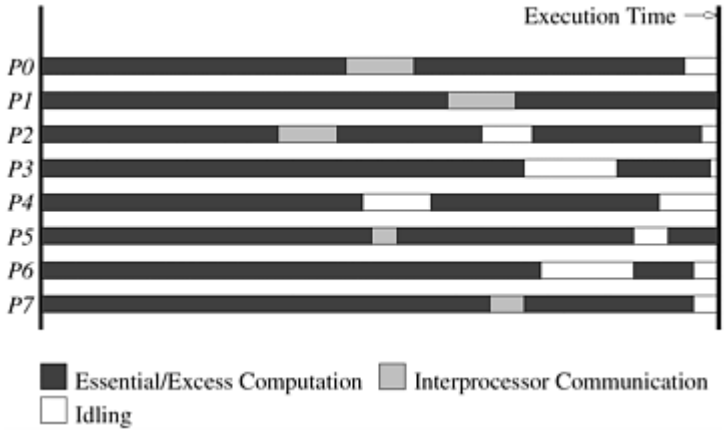


Figura 2.4 Ejecución de un programa paralelo hipotético, ejecutándose en ocho elementos de procesamiento [GRAMA2003].

En un algoritmo paralelo la cuantificación de su rendimiento mediante el tiempo que toma la ejecución en función de los datos que maneja no es posible ya que se debe tener en cuenta su arquitectura de implementación. Esta cuantificación se realiza mediante métricas especialmente definidas para sistemas paralelos, y que se utilizan principalmente para indicar factores como la estimación del beneficio obtenido del paralelismo, las configuraciones y el tamaño de los problemas [GRAMA2003].

2.2.2. Métricas de rendimiento para sistemas paralelos

Entre las métricas de rendimiento para sistemas paralelos se encuentran [ANGUITA2005]:

MFLOPS (millones de operaciones en coma flotante por segundo): Esta medida pretende categorizar el nivel de prestaciones de un sistema de cómputo y facilitar la comparación entre distintos sistemas a través de una única cantidad. Se define por la siguiente expresión:

$$MFLOPS = \frac{Operaciones\_en\_Coma\_Flotante}{T_{CPU} \times 10^6}$$

T_{CPU} = Tiempo de ejecución de la sobre el sistema de computo

Como se puede ver, esta medida puede variar según el programa, por lo que no sirve como medida característica de un sistema. Además, ni el conjunto de instrucciones en coma flotante es el mismo en todas las máquinas, ni tampoco la potencia o el coste de dichas instrucciones (la precisión puede ser diferente, pueden no consumir el mismo número de ciclos la multiplicación y la suma, etc.). Por ello se utilizan a veces los MFLOPS normalizados, que se obtienen dando un peso relativo a cada instrucción.

$$MFLOPS_{normalizados} = \frac{\sum_{i=1}^{n} Operaciones\_en\_Coma\_Flotante_i \times W_i}{T_{CPU} \times 10^6}$$

N = Número de instrucciones en coma flotante en el repertorio

Wi = proporción de costo con relación a la instrucción de menor costo en el repertorio

Tiempo de ejecución: En un programa secuencial es el tiempo transcurrido entre el comienzo y el fin de su ejecución en un computador secuencial. El tiempo de ejecución de un programa paralelo es el tiempo que transcurre a partir del momento en que la computación paralela empieza hasta el momento en que el último elemento de proceso termina la ejecución [MODELOS2005].

Tamaño del problema: Se define como el número de pasos de computación básicos que deben ejecutarse sobre un elemento de procesamiento para resolver un problema implementado con el mejor algoritmo secuencial [GRAMMA2003].

Mejora del rendimiento: Indica el beneficio relativo de resolver un problema en forma paralela. Es definida como la relación del tiempo tomado en resolver un problema sobre un elemento de procesamiento, dividida entre el tiempo requerido para resolver el mismo problema en un sistema paralelo [MODELOS2005].

Costo: Es el producto del tiempo de ejecución paralelo por el número de elementos de procesamiento usados. Los costos reflejan la suma del tiempo que cada elemento de procesamiento gasta resolviendo el problema.

Un sistema paralelo se considera de costo óptimo, si el costo de resolver un problema tiene el mismo crecimiento asintótico en función del tamaño de entrada como el algoritmo secuencial más rápido conocido ejecutándose sobre un único elemento de proceso [GRAMMA2003].

Sobrecarga total de sistemas paralelos: Se define como la computación adicional en que incurre un programa paralelo al ejecutarse sobre los elementos de proceso del sistema paralelo, con respecto al mejor algoritmo secuencial conocido ejecutándose sobre un sólo elemento de procesamiento. Su cálculo se realiza mediante la resta de los costos de ejecutar el programa en el cluster y el tiempo en un sólo nodo. [MODELOS2005].

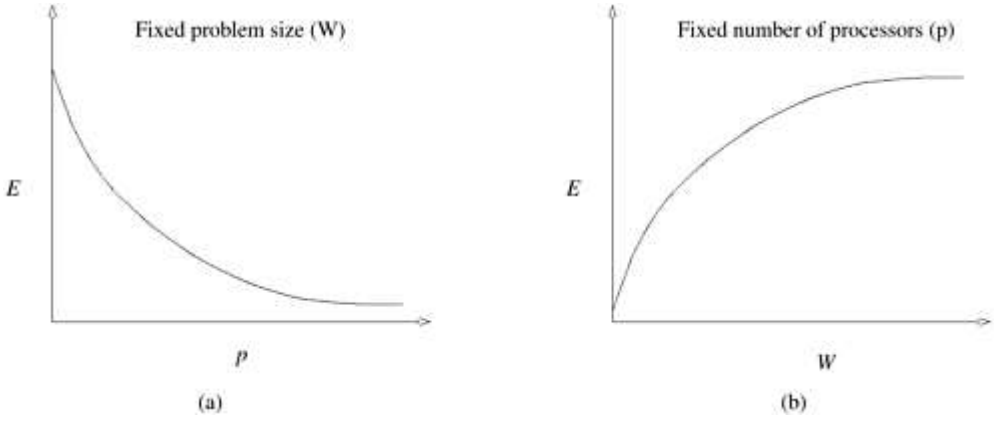
Eficiencia: Indica la cantidad de tiempo que un elemento de proceso es empleado útilmente; es definida como la relación entre la mejora en el rendimiento y el número de elementos de procesamiento. En un sistema paralelo ideal, la mejora en el rendimiento es igual al número de elementos y la eficiencia es igual a uno. En la práctica, la mejora en el rendimiento es menor que el número de elementos y la eficiencia esta entre cero y uno [MODELOS2005].

Escalabilidad: Es una medida de la capacidad de un sistema paralelo para aumentar la mejora en el rendimiento a medida que se aumenta el número de elementos de procesamiento, indicando qué tan eficiente es el incremento de los recursos de proceso en un sistema paralelo. Esta métrica es muy utilizada ya que inicialmente los programas son diseñados y probados para algunos elementos de procesamiento, y luego se ejecutan sobre máquinas con un número mayor de elementos de procesamiento [GRAMMA2003].

Existen dos fenómenos observados comúnmente en los sistemas paralelos:

- Para un tamaño de problema dado, si se aumenta el número de elementos de proceso, la eficiencia total del sistema paralelo baja. Este fenómeno se muestra en la figura 2.5a.

- En muchos casos, la eficiencia de un sistema paralelo se incrementa si el tamaño de problema aumenta mientras se mantiene el número de elementos de procesamiento constante como se observa en la figura 2.5b.



Con aumento en el número de elementos Con aumento del tamaño de problema

Figura 2.5. Variación de la eficiencia [GRAMA2003].

Otras métricas de la escalabilidad para sistemas paralelos se han especificado con el fin de analizar sistemas diferentes o para considerar limitaciones de arquitecturas físicas, por ejemplo en aplicaciones de tiempo real. En muchas aplicaciones, el tamaño máximo de un problema es limitado no por el tiempo, la eficiencia, o los modelos fundamentales, sino por la memoria disponible en la máquina. En tales casos, las métricas hacen suposiciones en la función de crecimiento de memoria disponible (con el número de elementos de proceso) y la estimación cómo la ejecución del sistema paralelo cambia con tal escalamiento [MODELOS2005].

Isoeficiencia: Determina la proporción de crecimiento del tamaño del problema requerida para mantener la eficiencia constante frente a incrementos en la cantidad de elementos de procesamiento. Una función de isoeficiencia pequeña significa que incrementos pequeños en el tamaño del problema son suficientes para una utilización eficiente de un

número creciente de elementos de proceso, lo que indica que el sistema paralelo es altamente escalable. Por el contrario, una función de isoeficiencia grande indica un sistema paralelo bajamente escalable. La función de isoeficiencia no existe para sistemas paralelos no escalables [MODELOS2005].

La función de isoeficiencia agrupa las principales características de un algoritmo paralelo y de la arquitectura paralela sobre la que se ejecuta. Mediante un análisis de isoeficiencia, es posible probar la ejecución de un programa paralelo sobre unos cuantos elementos de proceso y predecir como será el rendimiento sobre un número más grande de elementos de proceso, la función de isoeficiencia también caracteriza la cantidad de paralelismo inherente en un algoritmo paralelo. Mediante el análisis de isoeficiencia es posible estudiar el comportamiento de un sistema paralelo con respecto a cambios en los parámetros de hardware tales como la velocidad de los elementos de procesamiento y canales de comunicación, además es útil con algoritmos paralelos a los que no es posible calcularles un valor del tiempo de ejecución [GRAMMA2003].

2.3. Programación Paralela

La forma de estructurar una solución empleando programación paralela es diferente a la utilizada en las soluciones secuenciales. Generalmente esta estructuración se realiza desde dos enfoques: la forma en que se comunican las entidades de ejecución y la manera en que son distribuidas e interactúan estas entidades sobre los elementos de procesamiento. Estos dos enfoques se conocen como: modelos de comunicación de programas paralelos y modelos de algoritmos paralelos, respectivamente.

2.3.1. Modelos de Comunicación de programas paralelos

- **Paso de Mensajes:** Consiste en el intercambio de mensajes entre los elementos de procesamiento para intercambiar datos, trabajos e información de sincronización sobre los procesos que se ejecutan [CUMPLIDO2004].

Las interacciones se logran al enviar y recibir mensajes, las operaciones son enviar (SEND) y recibir (RECEIVE), en estas operaciones se especifica el proceso destino y los datos a enviar [CUMPLIDO2004]

Las transmisiones pueden ser síncronas o asíncronas, en la síncrona el proceso que ejecuta un SEND queda bloqueado hasta que el destino ejecute la función RECEIVE correspondiente y viceversa; mientras que en la asíncrona, SEND no deja bloqueado el proceso que lo ejecuta; generalmente se ofrece una implementación de RECEIVE que deja bloqueado el proceso hasta que se realice el correspondiente SEND. Este bloqueo permite sincronizar procesos [ANGUITA2005].

Ejemplos de software que utilizan paso de mensajes son MPI y PVM; las arquitecturas paralelas en las que se emplea son multiprocesadores y multicomputadores [INTRODUC1999].

- **Variables compartidas:** Consiste en la interacción entre procesadores al modificar datos almacenados en un espacio de direcciones compartido. Se emplea en arquitecturas paralelas que tienen un espacio de memoria común a la cual acceden todos los procesadores.

  La comunicación entre procesos se realiza accediendo a variables compartidas, por lo tanto, mediante instrucciones de lectura y escritura en memoria.

  Las interacciones Lectura/Escritura son más difíciles de programar que las interacciones de sólo lectura porque se necesitan mecanismos que controlen los accesos concurrentes a la memoria [GRAMA2003].

  Las herramientas software que soportan esta comunicación ofrecen mecanismos para implementar sincronización en el acceso a memoria como: cerrojos, semáforos, variables condicionales, monitores [ANGUITA2005].

2.3.2. Modelos de Algoritmos Paralelos

- **Paralelismo de Datos:** En este modelo las tareas son mapeadas a procesos de manera estática o semiestática y cada tarea realiza operaciones similares sobre diferentes datos. El trabajo puede ser hecho en fases y los datos operados en

diferentes fases pueden ser diferentes. Generalmente las fases de cálculo de paralelismo de datos están entremezcladas con interacciones para sincronizar las tareas o para que obtengan nuevos datos [ANGUITA2005].

Este paralelismo se encuentra implícito en las operaciones con estructuras de datos (vectores y matrices). Desde el punto de vista de un programa secuencial, las operaciones con vectores y matrices se implementan mediante bucles, es por esto que algunos autores relacionan el paralelismo de datos con los bucles porque el paralelismo de datos se puede extraer de los bucles analizando las operaciones realizadas con la misma estructura de datos en las diferentes iteraciones del bucle [ANGUITA2005]

Una característica de los problemas de paralelismo de datos es que el grado de paralelismo de datos se incrementa con el tamaño del problema haciendo posible usar más procesos para solucionar de manera efectiva grandes problemas [GRAMMA2003].

Un ejemplo de este es la multiplicación de matrices, en la descomposición mostrada en la figura 2.6.

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} 1&: C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ 2&: C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ 3&: C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ 4&: C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

Figura 2.6. Multiplicación de matrices y la descomposición de la operación en paralelismo de datos [GRAMMA2003].

- **Paralelismo de Tareas**: Observando la estructura del gráfico de dependencia de tareas que se defina para una aplicación es posible asignar a los elementos de proceso tareas que se pueden ejecutar en paralelo. Este es el tipo de paralelismo que es expresado por tareas independientes en un gráfico de dependencia de tareas. Generalmente se emplea para solucionar problemas en los cuales la cantidad de datos asociados a las tareas es relativamente mayor a la cantidad cálculos asociados a los datos.

Ejemplos de algoritmos basados en gráficos de tareas son: ordenamiento rápido (quicksort), factorización de matrices y algoritmos paralelos derivados de la descomposición *Divide y Vencerás* (divide-and-conquer), el cual consiste en dividir un problema en problemas independientes más pequeños y combinando los resultados para obtener un resultado final "*Las tareas presentan una estructura de árbol. No habrá interacciones entre las tareas que cuelgan de un mismo padre*" [ANGUITA2005].

- **Modelo Maestro-Esclavo (master-slave) o granja de tareas (task-farming):** Consiste en definir uno o varios procesos maestros y procesos esclavos. El proceso maestro se encarga de generar y distribuir el trabajo entre los esclavos, esta distribución puede ser de forma estática, es decir, el nodo maestro distribuye el trabajo antes de iniciar al ejecución, o puede ser de forma dinámica para que los esclavos que terminen sus tareas puedan realizar otras y no tengan que esperar a los otros esclavos (distribución de carga). El maestro también puede tener funciones de sincronización para los esclavos, esto es empleado cuando se definen varias fases en un programa y el trabajo de cada fase debe terminar antes de iniciar la siguiente fase de trabajo.

  Debe asegurarse que el maestro no se convierta en el cuello de botella del sistema, lo cual sucede si las tareas son muy pequeñas o los esclavos relativamente rápidos [ANGUITA2005].

  Este modelo puede emplearse tanto con Paso de mensajes como con Variables Compartidas por que hay interaccione en dos sentidos: el maestro suministra el trabajo al esclavo y el esclavo espera las órdenes del maestro. La granularidad debe escogerse de forma que el costo de realizar trabajo domine el costo de la transferencia de trabajo y la sincronización [GRAMMA2003].

- **Flujo de Datos o Segmentación:** Es una forma de paralelismo de tareas en la cual distintas funciones procesan un flujo secuencial de datos [CUMPLIDO2004]. La segmentación consiste en una cadena donde los eslabones son: productor-consumidor, cada proceso puede ser visto como un consumidor de una secuencia de datos del proceso anterior y al mismo tiempo como un productor de datos para el siguiente proceso. Generalmente este modelo involucra un mapeo estático de

tareas sobre procesos. La ejecución simultánea de diferentes programas sobre un flujo[3] de datos es llamado paralelismo de flujo [GRAMMA2003].

2.3.3. Alternativas de comunicación en programas paralelos

Existen patrones de comunicación que se repiten en algoritmos paralelos especialmente en los orientados al paralelismo de datos, y que son útiles como referencia cuando se desea realizar este tipo de aplicaciones, entre ellos se encuentran [ANGUITA2005]:

- Reordenamiento de datos entre procesos con patrones como: permutaciones, rotaciones.
- Difusión de datos, por ejemplo difundir la media en el calculo de la desviación típica.
- Reducción de un conjunto de datos como resultados de sumas, restas, cálculos de máximo y mínimo.
- Reducciones en paralelo con el mismo conjunto de datos, pero cada uno con diferente número de elementos (evaluación de polinomios o la aproximación de una función mediante polinomio utilizando la fórmula de Newton).
- Sincronización de múltiples procesos en un punto (barreras[4]).

Diferentes herramientas software encapsulan estos patrones mediante funciones que se conocen como funciones colectivas, entre dichas funciones se encuentran [ANGUITA2005]:

- Comunicación múltiple uno-a-uno: consisten en un elemento que envía un único dato o una estructura de datos) y un único elemento recibe el mensaje. Si todos los elementos envían y reciben se implementa una *permutación*.

[3] Grupo contiguo de datos
[4] es un punto de sincronización en el flujo de control de un programa que todos los procesos de un grupo deben alcanzar para que se pueda continuar con la ejecución que hay después de la barrera

- Comunicación uno a todos: Un elemento envía y los otros reciben puede ser:
  Difusión: todos los procesos reciben el mismo mensaje
  Dispersión (scatter): cada proceso receptor recibe un mensaje diferente

- Comunicación todos a uno: En este caso todos los elementos envían un mensaje a un único elemento

- Reducción: Los mensajes enviados se combinan en un sólo mensaje mediante operador (suma, multiplicación, máximo, mínimo, AND, OR). La operación de combinación es usualmente conmutativa y asociativa.

- Acumulación (gather): Los mensajes se reciben de forma concatenada en el receptor (en una estructura vectorial)

- Comunicación todos a todos: todos los elementos ejecutan una comunicación uno a todos:

- All-broadcast (todos difunden): todos los elementos realizan una difusión, normalmente las transferencias recibidas se concatenan en función del identificador del elemento que envía, de forma que todos los procesos reciben lo mismo. Cuando cada elemento concatena diferentes transferencias se conoce como *All-scater*:

También es posible tener la combinación de algunas de las anteriores.

Finalizando este capítulo cabe resaltar que no existe una única tendencia al abordar la computación paralela; hay otras formas de estudiar los sistemas paralelos dependiendo de la aplicación en particular. En este capitulo se han incluido conceptos aplicables a los sistemas paralelos más comunes, y las clasificaciones conceptuales hechas por diferentes autores se han agrupado en tal forma que a partir de conceptos generales se obtiene la estructura expuesta.

Capitulo 3 ARQUITECTURAS PARALELAS

Las arquitecturas paralelas son un conjunto de recursos, métodos y técnicas que permiten algún nivel de paralelismo sobre arquitecturas computacionales. Además incluyen la forma en que se estructuran dichos recursos para la ejecución simultánea de procesos, funciones y operaciones. No existe una única clasificación que permita integrar todas las arquitecturas que se han desarrollado para este tipo de procesamiento, sin embargo existe una terminología que es necesario conocer cuando se intenta abordar el tema [ANGUITA2005].

Se han propuesto diversas taxonomías para arquitecturas paralelas, pero no se ha definido una general debido principalmente a dos causas:

- Incluir en la taxonomía implementaciones de propósito específico que no se ajustan fácilmente a ninguna de las clasificaciones existentes.
- Excluir arquitecturas que incorporan mecanismos de paralelismo de bajo nivel (Múltiples unidades funcionales en la CPU, procesadores independientes para entrada y salida, periféricos con acceso directo a memoria (DMA), etc) [BEVILACQUA2004].

A continuación se expone una de las primeras taxonomías para sistemas computaciones, la *taxonomía de Flynn*[5], que es ampliamente utilizada:

[5] Propuesta por Michael J. Flynn in 1972 [FLYNN2006]

3.1. Taxonomía de Flynn

La taxonomía de Flynn clasifica a las arquitecturas de computadores en cuatro tipos: SISD, SIMD, MIMD y MISD de acuerdo al número de secuencias o flujos de instrucciones y secuencias o flujos de datos que pueden procesarse simultáneamente en el computador [VRENIOS2002].

SISD: (Single Instruction, Single Data) un flujo de instrucciones único trabaja sobre un flujo de datos único [HILERA1998]

SIMD (Single Instruction, Multiple Data) un único flujo de instrucciones procesa varios flujos de datos, cada instrucción codifica varias operaciones iguales pero cada una actúa sobre datos distintos. [ANGUITA2005]

MIMD (Multiple Instruction, Multiple Data) múltiples flujos de instrucciones trabajan sobre un flujo múltiple de datos, ejemplos de esta categoría son los multiprocesadores, los multicomputadores y los cluster Linux [HILERA1998] en las siguientes secciones se encuentran las definiciones de estos sistemas.

MISD (Multiple instrucción, single data) Existe controversia sobre esta clasificación, para algunos autores es un resultado teórico de la taxonomía, otros relacionan el término con las instrucciones de un procesador segmentado [MORGADO2003], mientras que otros lo relacionan con sistemas de respaldo en caso de fallas, por ejemplo en sistemas de control de fallas para aviones [FLYNN2006]. *"Los computadores MISD constituyen una clase de computadoras cuyo comportamiento se puede implementar con iguales prestaciones en un computador MIMD en el que sus procesadores se sincronizan para que los datos vayan pasando desde un procesador a otro*" [ANGUITA2005].

Dentro de cada categoría de esta taxonomía es posible desarrollar sistemas que explotan diferentes tipos de paralelismo. Los desarrollos que han tenido mayor difusión serán abordados desde dos perspectivas: sistema con un procesador y sistemas con múltiples procesadores.

3.2. Sistemas con un Procesador

En este apartado se describe la principal técnica que para aumentar el rendimiento en un procesador mediante el uso de paralelismo, la segmentación de cauce. Posteriormente se describen los principales procesadores desarrollados que implementan paralelismo de dos tipos: *paralelismo entre instrucciones* (ILP) y paralelismo a nivel de datos [ANGUITA2005].

3.2.1. Segmentación de cauce

Para lograr paralelismo a nivel de instrucción en un procesador se ha recurrido a una técnica conocida como segmentación de cauce o *pipelining* la cual permite superponer la ejecución de varias instrucciones [VRENIOS2002]. En un sistema, una operación que se ejecuta en un tiempo T se divide en una serie de fases o etapas de manera que cada una de las fases se ejecuta independientemente de las otras. Con etapas que consumen un tiempo t, la ganancia de velocidad que se consigue en la ejecución de n operaciones consecutivas, vendría dada por:

$$S(n) = \frac{T\,\text{sin}\_segmentar}{Tsegmentado} = \frac{n \times T}{TLI + (n-1) \times t}$$

TLI (Tiempo de Latencia de Inicio), es el tiempo que tarda en procesarse completamente la primera instrucción del cauce.
n, número de operaciones.
t, tiempo de cada etapa.
T, tiempo de ejecución de una operación completa.

Al procesar n operaciones del cauce, la primera instrucción termina de ejecutarse pasado un tiempo igual a TLI desde el comienzo del procesamiento, pero a partir de ese momento, cada tiempo t (el tiempo de procesamiento de una etapa) se terminará cada una de las (n-1) instrucciones restantes. A medida que el número de operaciones a procesar es mayor la ganancia tiende a S_{max}=T/t. Como t es el tiempo de procesamiento correspondiente a una etapa, cuanto menor sea mayor será la ganancia máxima alcanzable. Mediante una mayor relación entre el tiempo de ejecución de la operación sin

segmentar y el tiempo de la etapa, mayor será la ganancia máxima que se puede obtener [ANGUITA2005].

Típicamente una operación tiene cuatro etapas[6] [ANGUITA2005]:

- IF (Instruction Fetch): captación de la instrucción.
- ID/OF (Instruction Decode/Operand Fetch): decodificación y lectura de datos.
- EXEC(Execute): ejecución de la operación en la unidad funcional especifica
- OS (Operand Store): escritura en el banco de registros.

3.2.2. Procesador Segmentado

Un procesador segmentado ejecuta varias instrucciones simultáneamente en diferentes etapas de su procesamiento mediante la implementación de segmentación de cauce. Existen formas de aumentar las prestaciones de este tipo de procesadores, una de ellas es diseñar cauces con más etapas donde cada etapa tiene una duración menor, esto es característico de los procesadores supersegmentados. Las figura 3.1, 3.2 y 3.3 muestran los diagramas del comportamiento de un procesador no segmentado, uno segmentado y uno supersegmentado; respectivamente [ANGUITA2005]:

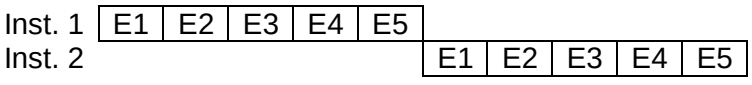


Figura 3.1 Esquema de ejecución de instrucciones en un procesador no segmentado [ANGUITA2005]

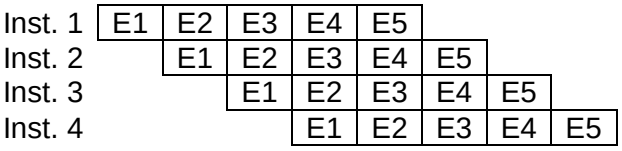


Figura 3.2 Esquema de ejecución de instrucciones en un procesador segmentado [ANGUITA2005]

[6] Estas etapas pueden variar según las características particulares del procesador (RISC, CISC)

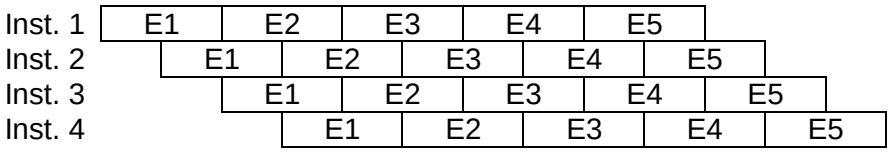


Figura 3.3 Esquema de ejecución de instrucciones en un procesador supersegmentado [ANGUITA2005]

En la ejecución simultánea de instrucciones aparecen problemas como: *dependencia de datos*, la cual sucede cuando se ejecutan dos instrucciones en el mismo intervalo de tiempo pero la segunda necesita los resultados de la primera; y el problema de *dependencia de control* causado por las bifurcaciones que las instrucciones de salto condicional originan en las secuencias de instrucciones [ANGUITA2005]. La gestión de estas dependencias permite incrementar el número de operaciones a realizar en el mismo intervalo de tiempo. De las diferentes soluciones a estos y otros problemas surgieron los procesadores superescalares y VLIW.

3.2.3. Procesador Superescalar

Es básicamente un procesador segmentado con hardware encargado de distribuir instrucciones que se pueden ejecutar en paralelo, aunque tiene limitaciones en el número máximo de instrucciones que se pueden ejecutar en paralelo. A diferencia de un procesador segmentado en el que hay una etapa de decodificación de la instrucción y de búsqueda de los operandos (ID/OF) en un procesador superescalar existen unidades de decodificación y de emisión de instrucciones separadas. La unidad de decodificación toma las instrucciones de la cola y después de decodificarlas, las pone en una estructura de buffers desde donde la unidad de emisión de instrucciones determina cuáles instrucciones pueden pasar a ejecutarse y en qué unidad funcional, permitiéndole ejecutar instrucciones en paralelo como se muestra en la figura 3.4 [ANGUITA2005].

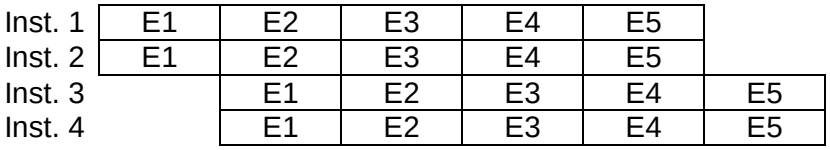


Figura 3.4 Procesador superescalar [ANGUITA2005].

3.2.4. Procesador VLIW (Very Long Instruction Word), Palabra de Instrucción muy Larga

Son procesadores segmentados con varias unidades de ejecución. La solución de la dependencia entre instrucciones se realiza mediante software, un compilador se encarga de planificar la distribución de las operaciones del programa, codificando en cada instrucción las operaciones que el VLWI puede ejecutar en paralelo, es decir, cada instrucción VLIW codifica varias operaciones que se pueden ejecutar simultáneamente en el procesador [ANGUITA2005].

Algunos problemas como el desempeño de los compiladores, la falta de compatibilidad con instrucciones de otros procesadores, el aumento de tamaño en los programas, han causado que las áreas de desarrollo de estos procesadores tienda a propósitos específicos donde las aplicaciones usan operaciones que se repiten con volúmenes de datos considerables, como los DSP para el tratamiento de señales [ANGUITA2005].

Los procesadores segmentados, superescalares y VLIW son catalogados dentro de la clasificación SISD de la taxonomía de Flynn.

3.2.5. Procesador Vectorial

Como su nombre lo indica está orientado al procesamiento de vectores (sumas, productos escalares, etc), su set de instrucciones implementa operaciones donde tanto los operadores como los resultados son vectores [NARAYAN2005]. Contiene elementos para ejecutar códigos que no pueden implementarse mediante operaciones vectoriales y unidades funcionales segmentadas donde se ejecutan las operaciones vectoriales. Debido a que los vectores son estructuras de datos es un procesador con un funcionamiento orientado al *paralelismo de datos* [ANGUITA2005].

Debido a su forma de operación estos procesadores se encuentran relacionados en la categoría SIMD de la taxonomía de Flynn.

3.3. Sistemas con Varios Procesadores

Para realizar la clasificación de estos sistemas se tienen en cuenta diversos factores relacionados con sus características de memoria como la organización de la memoria y la uniformidad en el acceso a memoria.

3.3.1. Organización de la memoria

- **Multiprocesadores o SM (Shared Memory, Memoria Compartida)** todos los procesadores comparten un mismo espacio de direcciones, los módulos de memoria se encuentran separados de los procesadores por medio de una red de interconexión que permite el acceso de todos los procesadores a la memoria. Este sistema tiene una variante en la cual los módulos de memoria se encuentran físicamente distribuidos entre los procesadores y el acceso a los datos de la memoria es realizado a través de un único espacio lógico de direcciones, se conoce como **DSM** (Distributed Shared Memory, Multiprocesadores con Memoria Compartida Distribuida) [ANGUITA2005].

- **Multicomputadores o DM (Distributed Memory, Memoria Distribuida)** cada procesador tiene su propio espacio de direcciones particular, así cuando un procesador requiere acceder a un dato ubicado en un espacio diferente al local, debe hacerlo a través de un sistema de comunicación adicional. El programador necesita tener en cuenta dónde se almacenan los datos para acceder a ellos [ANGUITA2005].

3.3.2. Uniformidad en el acceso a memoria

- **UMA (Uniform Memory Acces, Acceso Uniforme a Memoria)**: El tiempo de acceso de todos los procesadores a una determinada posición de memoria es el mismo, por esta característica también se conocen como SMPs (Symmetric

Shared-Memory Multiprocessors, Multiprocesadores Simétricos de Memoria Compartida) [HENNESY2002]. La figura 3.5 muestra la arquitectura UMA.

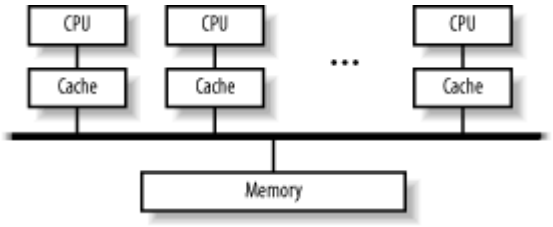


Figura 3.5 Arquitectura UMA [SLOAN2004]

- **NUMA (Non-Uniform Memory Acces, Acceso no Uniforme a Memoria)** El tiempo de acceso a una posición de memoria local es menor que a una memoria local de otro procesador. Debido al múltiple acceso de todos los procesadores a las memorias, aparece el término *coherencia* para indicar la integridad de los datos almacenados en sus memorias cache locales. De acuerdo a la coherencia, esta clasificación se ha dividido en tres grupos:

    - NCC-NUMA (Non Cache Coherent - Non Uniform Memory Access - Acceso a Memoria no Uniforme sin Coherencia de Cache) Estas arquitecturas no incorporan hardware encargado de evitar incoherencias entre caches de distintos nodos [ANGUITA2005].
    - CC-NUMA: (Cache Coherent – Non Uniform Memory Access, Acceso a Memoria no Uniforme y con Coherencia de Caché) Utilizan hardware dedicado para mantener la coherencia entre caches de distintos nodos [ANGUITA2005].
    - COMA (Cache Only Memory Access, Acceso Solo A Memoria Cache). La memoria local de los procesadores se gestiona como si fuese una memoria cache. El sistema de mantenimiento de coherencia se encarga de llevar dinámicamente (en tiempo de ejecución) el código y los datos a los nodos donde se necesiten. No existe actualmente ningún sistema comercial que implemente esta técnica [ANGUITA2005]. La figura 3.6 muestra la arquitectura NUMA.

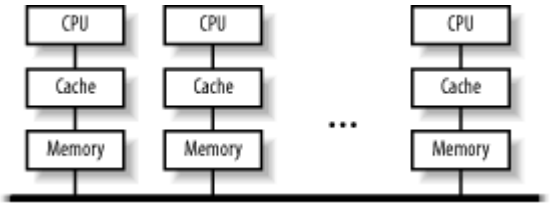


Figura 3.6 Arquitectura NUMA [SLOAN2004]

Los multiprocesadores y multicomputadores se consideran pertenecientes a la categoría MIMD de la arquitectura de Flynn.

Aunque la diferencia de programación entre multiprocesadores y multicomputadores parece ser clara, se esta dando una convergencia que esta disminuyendo la diferencia entre multiprocesadores y multicomputadores, por ejemplo en multicomputadores se puede implementar un espacio de memoria virtual compartido (virtual shared memory) los mensajes en este caso transfieren paginas, también hay interfaces de red que incluyen capacidades de RMA (Remote Memory Access) que permiten escribir y leer (en y de) zonas de memoria remota incluidas en la interfaz de red o incluso sobre regiones de memoria del propio procesador como QsNet de Quadrics. [ANGUITA2005]

3.3.3. Otras clasificaciones

Hay otras denominaciones de multicomputadores paralelos resaltando diferentes aspectos de su arquitectura: utilización de componentes disponibles en el mercado, la relación prestaciones/precio, disponibilidad y el nivel o niveles de paralelismo que aprovechan [ANGUITA2005], una de las más comunes es de acuerdo al nivel de acoplamiento entre los nodos, como se muestra en la Figura 3.7.

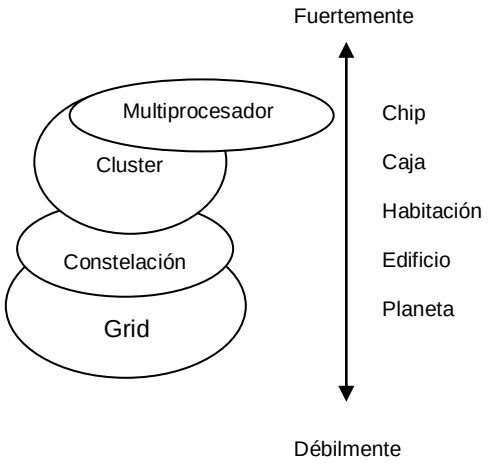


Figura 3.7 Arquitecturas según acoplamiento entre nodos [PLATA2003].

A continuación se presenta una descripción de estos sistemas:

Grid: La computación en grid es una forma de computación distribuida que implica coordinación y división de los cálculos, aplicaciones, datos y/o recursos de red a través de organizaciones distribuidas geográfica, funcional o administrativamente [GRID2004].

Un grid es básicamente un conjunto de software y hardware físicamente distribuido, formado por recursos computacionales que pueden ser entidades lógicas, tales como un sistema de archivos distribuidos o un cluster de computadores; o entidades software y hardware como sistemas de almacenamiento, recursos de red, equipos de telemetría, computadoras de escritorio entre otros [TARRICONE2004].

Este conjunto de recursos es estructurado en un modelo de arquitectura virtual que permite distribuir la ejecución de procesos sobre una infraestructura paralela heterogénea (diferentes arquitecturas, sistemas operativos, etc), que aprovecha los recursos que no están siendo utilizados localmente (ciclos de CPU, capacidad de almacenamiento en disco) de manera que todos los recursos disponibles participan en el procesamiento total logrando alta disponibilidad [WIKI2006].

Se observa en la Figura 3.8 un grid local interconectado por una LAN, un grid más amplio conectando grids locales por medio de una WAN y ampliado por un sistema distribuido soportado por software Condor, y un grid global interconectado por Internet

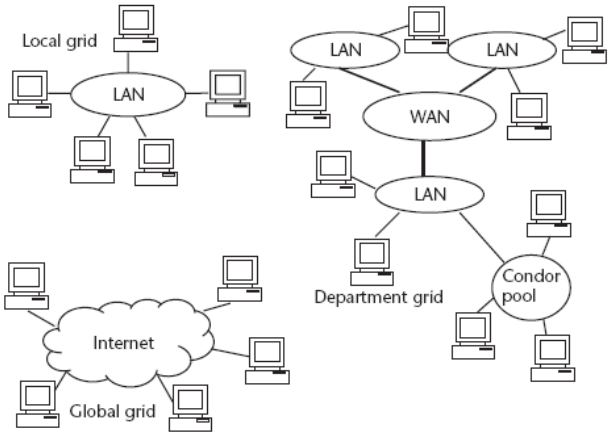


Figura 3.8 Configuraciones típicas de Grid [TARRICONE2004]

Un grid es concebido como un depósito gigante de recursos donde los usuarios pueden añadir o quitar elementos en cualquier momento. Esta flexibilidad se consigue debido a que se basa en estándares universalmente reconocidos, protocolos y tecnologías compatibles con la WEB; además de ser compatible con la mayoría de estructuras y metodologías de implementación de sistemas distribuidos [TARRICONE2004].

Un grid posee una arquitectura formada por tres niveles como se aprecia en el Figura 3.9.

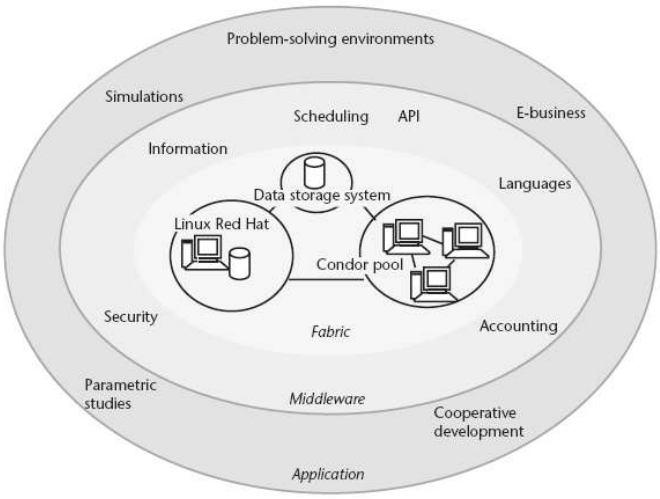


Figura 3.9 Estructura en capas de un entorno de grid [TARRICONE2004]

- Fabric Level, Nivel Estructural: Incluye todos los elementos que une el grid. Esta conformado por los recursos software, hardware y lógicos pertenecientes al grid e interconectados a través de redes o de Internet.

- Midleware Level, Nivel Intermedio: Conformado por el software encargado de enlazar los recursos con los manejadores de niveles más altos con el fin hacer transparentes los detalles específicos de la implementación a los usuarios del grid.

- Aplication Level, Nivel de Aplicación: Es el que interactúa directamente con los usuarios finales, incluye servicios de alto nivel para que los desarrolladores implementen soluciones en el grid y herramientas Web que permiten enviar trabajos, recoger y analizar los resultados, etc [TARRICONE2004].

Constelaciones: Son sistemas en los que el número de nodos es menor que el número de procesadores, es decir, el sistema esta conformado por nodos con varios procesadores.

Multiprocesadores: Un ejemplo de este nivel de acoplamiento es el MPP (Massively Parallel Processors- Procesadores Masivamente Paralelos), formados por un número de procesadores superior a 100. Sus implementaciones son muy variadas, algunas se catalogan dentro de la clasificación DSM y otras dentro de los multicomputadores. Son sistemas fuertemente acoplados generalmente diseñados para solucionar problemas paralelos concretos. Ejemplos de estos sistemas son la línea SX de NEC o la X1 de Cray, procesadores de propósito general como la línea Origin y Altix de SGI y T3E de Cray.

Cluster: esta constituido por un conjunto de computadores completos (PCs, estaciones de trabajo, servidores) interconectados a través una red de comunicación, que se utiliza como un recurso de cómputo único, la red esta dedicada únicamente al tráfico que genera la aplicación o aplicaciones paralelas. Según la aplicabilidad de los clusters, se han desarrollado las siguientes líneas tecnológicas [YAÑEZ2004]:

- Servidores virtuales: Este tipo de tecnología de clusters es el destinado al balanceo de carga, permite que un conjunto de servidores de red compartan la carga de trabajo y de tráfico de sus clientes, aunque aparezcan para estos clientes como un único servidor. Al balancear la carga de trabajo en un conjunto de servidores, se mejora el tiempo de acceso y la confiabilidad en la Web de un portal de cualquier tipo (empresarial, educativo, recreativo).

- Alta disponibilidad: Implementa agrupaciones de servidores que actúan entre ellos como respaldos de información. Este tipo de clusters también se conoce como "clusters de redundancia". La flexibilidad y robustez que proporcionan este tipo de clusters, los hacen necesarios en ambientes de intercambio masivo de información, almacenamiento de datos sensibles y donde sea necesaria una disponibilidad continua del servicio ofrecido.

- Alto rendimiento: Surgida para aliviar la necesidad de computación en determinadas aplicaciones, lo que persigue es conseguir que un gran número de máquinas individuales actúen como una sola máquina muy potente. Su mayor uso se da en problemas grandes y complejos que requieren una gran cantidad de potencia computacional.

Se ha utilizado la denominación de *commodity cluster* para distinguir a los cluster que utilizan redes disponibles comercialmente de aquellos que utilizan redes propietarias y que vienen incorporadas en soluciones clustering completas. El termino *Cluster Beowulf* identifica clusters con sistema operativo libre (generalmente Linux) y con un número limitado de hardware (sólo un nodo tiene teclado, ratón, monitor). Un cluster *Beowulf* emplea redes disponibles comercialmente [SLOAN2004].

Si los nodos están dedicados exclusivamente a trabajar en el sistema cluster este se considera *Dedicado*, pero si pueden trabajar independientemente del sistema cluster, y sólo al ser requerido se enlazan con el sistema, es *no Dedicado*: [SPECTOR2000].

Un cluster tiene tres elementos básicos un conjunto de computadoras individuales, una red que une esas computadoras y software que les permite compartir trabajo [SLOAN2004].

Según la configuración hardware (procesador, memoria, disco, interfaz de red) y el tipo de sistema operativo que tiene instalado cada nodo, es posible determinar si un cluster es Homogéneo o Heterogéneo, en el primer caso todos los nodos tienen arquitecturas idénticas y utilizan el mismo tipo de sistema operativo, en el segundo las arquitecturas son diferentes y/o utilizan diferentes sistemas operativos. [SPECTOR2000].

De acuerdo a la configuración de la red, se tienen diferentes arquitecturas de conexión para implementar un cluster, entre ellas se encuentra la configuración en Bus en la que los nodos comparten el medio de conexión y se encuentran unidos a través de un Hub o un Switch, como se muestra en la figura 3.10. Es una de las configuraciones más sencillas, y por ser un medio compartido se debe utilizar el hardware que reduzca al mínimo las colisiones y el tiempo de tránsito de la información en la red [SPECTOR2000].

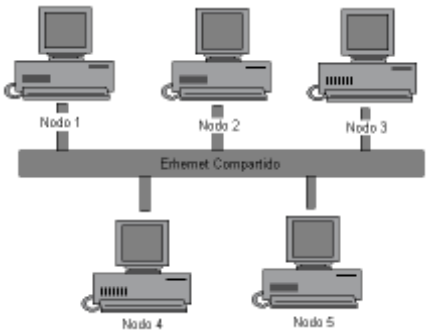


Figura 3.10 Topología en Bus [SPECTOR2000]

Otra configuración empleada es en anillo (RING) con tecnologías como ATM, FDDI, SONET, esta topología se muestra en la figura 3.11 [SPECTOR2000].

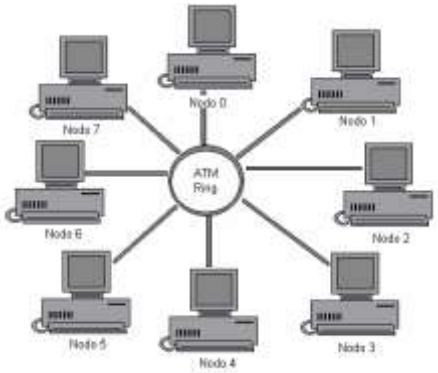


Figura 3.11 Cluster en configuración anillo [SPECTOR2000]

Mallas e hipercubos

La configuración en Malla (MESH) se asemeja a un plano cartesiano con nodos a lo largo del eje vertical *Y*, y a lo largo del eje horizontal *X* [GRAMA2003].
Las mallas de dos dimensiones son usadas comúnmente para interconectar sistemas paralelos, las mallas pueden extenderse en otras dimensiones o unir los nodos de sus extremos formando toroides, estas configuraciones se muestran en la figura 3.12.

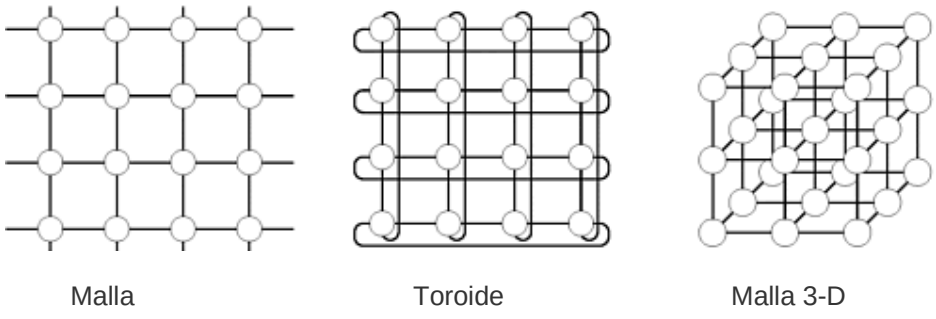


Figura 3.12 Configuraciones en Malla [GRAMA2003]

Un cubo tridimensional es una generalización de una malla 2-D a tres dimensiones. Una variedad de simulaciones físicas pueden ser mapeadas directamente a esta topología como modelos climáticos y modelos estructurales [GRAMA2003].

Existen otras estructuras específicas derivadas de los cubos que son muy empleadas en los cluster más complejos llamadas hipercubos. Se componen por una estructura

geométrica de uno o más cubos de tres dimensiones donde todos los nodos tienen conexión con nodos adyacentes, como se muestra en la figura 3.13 para un hipercubo de dos dimensiones:

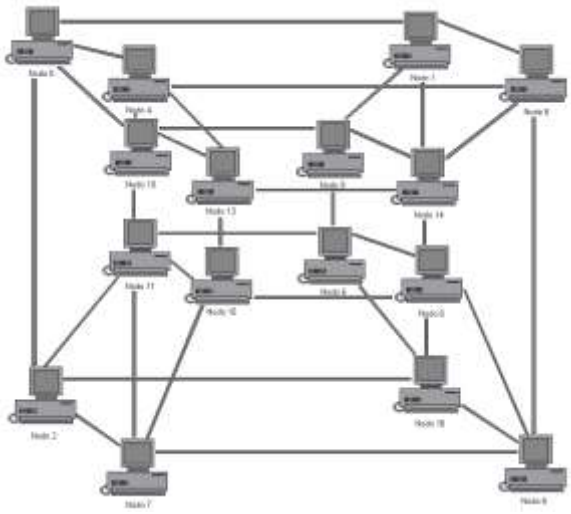


Figura 3.13. Hipercubo de dos dimensiones [SPECTOR2000]

3.3.4. Relación de las implementaciones más comunes con la taxonomía de Flynn

Los sistemas formados por sólo un procesador y que utilizan técnicas especiales para implementar paralelismo se pueden ubicar dentro de la taxonomía de Flynn, debido a que al estar conformados por un único procesador se tiene una sola unidad de procesamiento ejecutándose en un instante de tiempo especifico y por ende un único flujo de instrucciones, sólo resta identificar si estas instrucciones procesan un único flujo de datos o varios simultáneamente.

Para el caso de los sistemas MIMD, existe una gran variedad de implementaciones por lo que se hace necesario incluir una división interna que permita caracterizar de manera específica cada sistema particular. Tradicionalmente se recurre a la estructura básica de su memoria y a la forma en que acceden a esta, esta clasificación da una idea muy adecuada de la forma en que funcionan estos sistemas; pero existen implementaciones que a pesar de usar la misma estructura hardware pueden ubicarse en clasificaciones

diferentes debido a su arquitectura lógica, lo que dificulta diferenciarlos claramente. También se clasifican según su grado de acoplamiento y su distribución física, lo que a pesar de ser muy intuitivo no incluye todas las implementaciones especificas.

Debido a esto se expone una clasificación según la forma en que los elementos de proceso se relacionan con los sistemas de entrada y salida: se consideran sistemas *multiprocesador* como aquellos que comparten un único sistema (o conjunto de sistemas) completo de entrada y salida, y sistemas *multicomputador* a los que en cada uno de los elementos de procesamiento se observa su propio sistema de entrada; para esta clasificación se han considerado las definiciones de cada sistema presentadas a lo largo de este capítulo.

Existen sistemas como los MPP que son implementaciones paralelas con un número de procesadores superior a 100 que se pueden encontrar dentro de los multiprocesadores con memoria compartida distribuida y dentro de los multicomputadores, por esto se ubican en una intersección entre los sistemas multicomputador y multiprocesador como se muestra en la figura 3.14

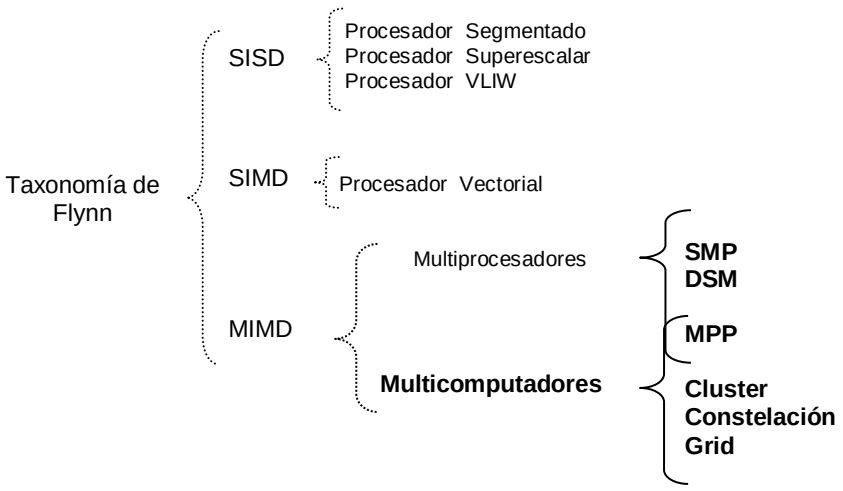


Figura 3.14. Relación de las implementaciones más comunes con la taxonomía de Flynn

Capitulo 4 CONSTRUCCIÓN DE UN CLUSTER DE COMPUTADORES

Para la construcción de un cluster de computadores, los autores [SLOAN2004] y [SPECTOR2000] recomiendan considerar las siguientes decisiones:

1. Determinar cuál es el objetivo del cluster, es decir, para qué va a ser empleado.
2. Escoger una arquitectura general para el cluster.
3. Escoger el hardware y el software del cluster.
4. Adecuar el espacio en el cual se ubicará el cluster.

Todas estas tareas están relacionadas entre si, pero la definición del uso que se le va a dar al cluster influye directamente en las otras actividades; aunque el hardware no siempre se puede elegir, en muchas organizaciones debe trabajarse con los recursos disponibles [SLOAN2004].

Al determinar cuál es el objetivo del cluster, hay que preguntarse si el cluster será dedicado o no, qué tipo de aplicaciones se ejecutarán y cuántos serán los usuarios, para decidir que programas se deben instalar y si es necesario un software de gestión de usuarios que permita que el sistema pueda atender un número de peticiones definido [SLOAN2004].

Después de definir la funcionalidad del cluster se busca una arquitectura conveniente y acorde con lo que se va a trabajar, una visión de las arquitecturas de cluster se encuentran en el capitulo 2 de este documento.

4.1. Selección del Hardware del Cluster de Computadores

Para la construcción de un cluster generalmente son empleados los siguientes componentes:

- Procesadores y placas base
- Memorias
- Discos para almacenamiento
- Dispositivos de entrada y salida
- Redes de interconexión

Un cluster se pueden construir comprando e integrando cada uno de los componentes o utilizando las torres de los equipos que se tienen a disposición (con sus propios elementos incorporados) conectándolos por medio de una red [GOVEA2002].

La decisión de la compra del sistema completo o de las partes del cluster esta influenciada por una serie de factores entre los que se encuentran la funcionalidad del cluster, el presupuesto y la disposición que se tenga para armar los equipos [SLOAN2004].

Cuando se utiliza el mismo tipo de elementos para todos los nodos, es posible configurar y evaluar un sólo sistema y luego clonarlo al resto. Hay que revisar por dentro todos los equipos y conocer qué dispositivos tienen y en qué condiciones se encuentran elementos como discos duros, unidades de CD-ROM, memorias, puertos y cables de potencia.

Al determinar el rendimiento de un nodo, los factores con mayor ingerencia son: la frecuencia de reloj del procesador, los tamaños de memoria cache, velocidad de los buses, capacidad de memoria, velocidad de acceso a disco y la latencia de la red. Los cuatro primeros están determinados por las características de la placa base y el procesador [SLOAN2004].

4.1.1. Procesadores y Placas Base

La selección del procesador esta directamente relacionada con la selección de la placa base, estas dos partes son el corazón del sistema, por lo tanto se necesita total compatibilidad entre ellas. Comercialmente ofrecen placa bases con procesadores integrados; pero estas configuraciones tienen bajo rendimiento [SLOAN2004].

Al seleccionar el hardware para un cluster se debe decidir cuántos procesadores va a tener cada uno de los nodos. Si se va a usar más de un procesador en un mismo nodo, se debe tener en cuenta que esto implica una arquitectura SMP y un acceso compartido a memoria, lo cual influye directamente en la programación empleada para el cluster. Si se decide emplear este tipo de hardware es aconsejable optar por una solución comercial de compañías con experiencia en multiprocesadores, como Compaq y Sun, y que la empresa brinde soporte y garantía. Los nodos SMP se recomiendan cuando el uso de cada nodo va a ser bastante alto [SLOAN2004].

No es suficiente considerar el número de procesadores y su velocidad, sino también las características particulares de los procesadores, por ejemplo los Intel Pentium MMX tienen instrucciones adicionales para gráficos y se tiene que escribir código específicamente para esas instrucciones para aprovechar esa característica del procesador, si no se van a realizar aplicaciones que requieran manejo de gráficos no debería manejarse este tipo de procesadores [SPECTOR2000].

Si se cuenta previamente con los equipos pueden ocurrir una variedad de situaciones, por ejemplo, disponer de una colección de máquinas Intel de diferentes generaciones (una mezcla de 486, Pentium y Pentium II) o máquinas de la misma generación de diferentes fabricantes (por ejemplo Compaq y DELL) o una mezcla de nodos multiprocesador y uniprocesador, de hecho existen diferentes configuraciones posibles; en estos casos lo mas importante es conocer que software permite aprovechar esas características particulares de cada sistema [SPECTOR2000].

En lo referente a la placa base es conveniente revisar todas las características que ofrece el fabricante y sopesar tanto el precio como la funcionalidad.

4.1.2. Memorias

Se debe tener en cuenta el tipo de memoria que soporta la placa base, su capacidad de almacenamiento y velocidad de acceso, lo más recomendable es que se use una memoria con la velocidad de acceso más rápida disponible en el mercado. Si el cluster incluye nodos SMP, no será posible modificar las características de memoria ya que estas vienen predefinidas por el fabricante [SLOAN2004].

4.1.3. Discos para el almacenamiento

La selección del disco duro depende de los conectores disponibles en la placa base, la interfaz más difundida en nuestro ambiente es la IDE (Integrated Drive Electronics Interface) también conocida como ATA (Advanced Technology Attachment), aunque actualmente las placas base vienen con S-ATA (serial ATA) que es una interfaz más rápida que la anterior; pero también lleva tiempo en el mercado la SCSI (Small Computer System Interface) que en su versión SCSI UltraWide soporta transferencias de 10Mbits/seg [WIKIIDE2006].

Se puede construir un cluster cuyos nodos obtengan la información de arranque por Red en lugar almacenarla en disco duro, esto es posible si la placa base tiene la opción de arranque por red. En los nodos sin disco puede ser una desventaja el tiempo que tardan en cargar los programas desde la red y si hay demasiado tráfico es posible que los nodos dejen de funcionar [NIETO2002].

Hace varios años el costo de los discos duros era alto y la ventaja de los nodos sin discos era la economía, pero como esto ha cambiado, ahora las principales ventajas de los nodos sin disco es la facilidad de mantenimiento (fallas hardware, reemplazo de discos) y la disminución de consumo de potencia [NIETO2002].

4.1.4. Dispositivos de entrada y salida

Si la placa base de los equipos lo permite es posible construir un cluster con un mínimo de componentes, un sólo monitor, mouse y teclado para todos los nodos. Esto es deseable cuando se utiliza una gran cantidad de nodos por razones como consumo de potencia, espacio, organización, etc.

Para trabajar con clusters de este tipo es necesario tener un control sobre su salida. Para esto hay dos opciones: una consola serial o un Switch KVM, este último es la opción más costosa.

- Consola Serial:
  Es un dispositivo que se utiliza con servidores Linux, permite la conexión de los equipos por medio del puerto serial. [PENGUIN2006]
  En la figura 4.1 se muestra una consola de la empresa Penguincomputing.

Figura 4.1 Consola para acceso a dispositivos entrada/salida
compatible con UNIX y Windows [PENGUIN2006]

- Switch KVM
  Un switch KVM (KeyBoard Video Mouse, Teclado Video Ratón) es un dispositivo de conmutación que permite el control de distintos equipos con tan sólo un monitor, un teclado y un ratón. Funciona a través de diferentes puertos como LPT1, LPT2, COM1, COM2 o una conexión inalámbrica a través de radio o infrarrojos. Este equipo permite la construcción de un cluster mínimo, con placas base que no soportan arranque sin monitor, teclado y mouse porque simula la conexión con los periféricos. [KVM2006]

En la figura 4.2 se muestra un diagrama que ilustra el funcionamiento de este dispositivo.

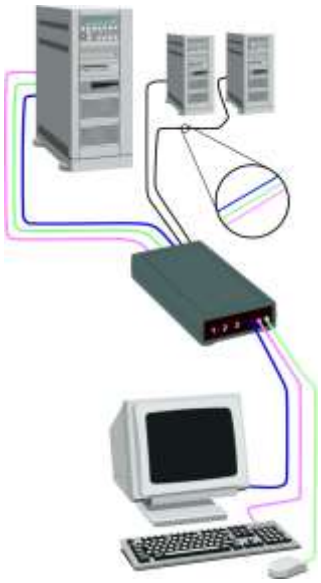


Figura 4.2 Funcionamiento básico de una switch KVM [KVM2006]

4.1.5. Redes de interconexión

Los dos factores claves a considerar al diseñar la red son el ancho de banda y la latencia. El ancho de banda es el principal tópico a tener en cuenta si las aplicaciones necesitan mover grandes bloques de datos, para el caso de aplicaciones de tiempo real o aplicaciones que necesiten mucha interacción entre los nodos es importante minimizar la latencia [VRENIOS2002].

La selección del tipo de red y el hardware de la misma, esta influenciado por la topología que se va a emplear. Se debe asegurar que los nodos tengan dispositivos tecnológicamente compatibles [SPECTOR2000].

Cuando se trabaja con máquinas previamente asignadas hay que verificar cuál es la velocidad que soporta la interfaz de cada nodo antes de comprar un dispositivo de interconexión.

Las redes Ethernet son la opción más común para clusters; también existen otras opciones de mejores características, pero costosas como: Myrinet de Myricom. Myrinet es una solución propietaria que proporciona conectividad bidireccional (cerca de 2Gb en cada sentido), una baja latencia (del orden de nanosegundos) y permite paquetes de tamaño arbitrario [VRENIOS2002]. La figura 4.3 tiene ejemplos de dispositivos Myrinet.

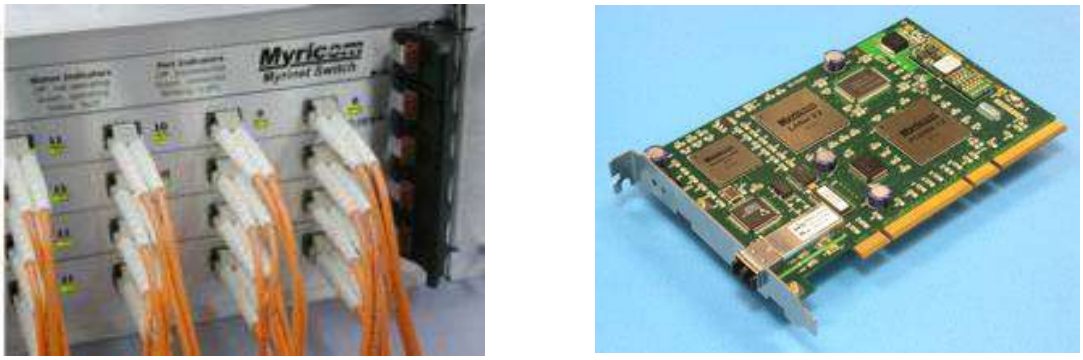


Figura 4.3 Switch de interconexión y tarjeta de red fabricados por Mirycom [CHOY2002]

Si el propósito del sistema es un reemplazo competitivo de un supercomputador existe la necesidad de una red mucho más rápida que Ethernet, esta fue una de las conclusiones obtenidas por el proyecto Beowulf [DIETZ1998].

Otras tecnologías de red que están emergiendo o están disponibles son:

- cLAN de Emulex
- QsNet de Quadrics
- Infiniband de Infiniband Consortium

El problema de estas tecnologías alternativas es su alto precio, en algunos casos los adaptadores de red pueden llegar a tener un costo mayor que todo el hardware en un nodo, por lo tanto se usan en cluster de alto rendimiento diseñados para reemplazar supercomputadores.

También se pueden usar redes con las que se haya trabajado en el lugar de construcción del cluster por ejemplo ATM y HiPPI (High Performance Parallel Interface) [SPECTOR2000].

Muchos cluster no necesitan un nivel tan alto de rendimiento, continuamente mejora la velocidad y disminuye el precio de las Redes Ethernet, lo que la hace una tecnología bastante empleada en Cluster con la ventaja adicional de que es una tecnología bien establecida y difundida por lo cual su configuración y gestión no requiere mayor entrenamiento [CHIRINOV2003].

La cantidad de hardware necesario para construcción de la red depende de la complejidad de la configuración que se desea construir. Si el cluster es pequeño basta con un hub o un switch; si se esta pensando en una arquitectura como una malla (mesh) se necesitan otros dispositivos de interconexión como tarjetas con múltiples puertos de conexión.

Una de las decisiones más difíciles es la compra de una tarjeta de Red externa o una placa base que tenga integrada la tarjeta de Red, algunos autores recomiendan utilizar tarjetas de Red que se puedan conectar en ranuras (slots) de expansión, debido a que es posible reemplazarlas por tarjetas de Red más rápidas que utilicen el mismo puerto a medida que cambia la tecnología.

Al adquirir dispositivos de interconexión hay que tener en cuenta el desempeño, por ejemplo un switch es más costoso que un hub pero mejora la comunicación entre los nodos. Por regla general los dispositivos de interconexión debe tener más puertos que el número de nodos, de esa manera es posible expandir el cluster [DIETZ1998].

4.2. Adecuación del Cluster

Todos los elementos Hardware que componen un cluster deben estar organizados adecuadamente dentro de un contenedor (o contenedores) que le permita su correcto funcionamiento. La selección del contenedor es crítica a medida que el tamaño del cluster aumenta.

Hay una dependencia mutua entre el tipo de hardware que se utiliza y su contenedor. Las dos formas más comunes de organizar los elementos son torres y racks.

4.2.1. Torres

Son las torres empleadas en los PCs, pueden albergar placas base AT y ATX aunque permiten colocar varios componentes hardware, no son muy empleadas en grandes cluster por el desperdicio de espacio.
Si se van a usar torres se tienen opciones limitadas para ubicarlas, necesita considerarse el espacio, el flujo de aire y el cableado, usualmente se utilizan armarios metálicos para colocar los nodos del cluster como se muestra en la figura 4.4:

Figura 4.4 Cluster de torres [CHOY2002]

4.2.2. Racks

Un rack es un gabinete dentro del cual se instalan todos los componentes necesarios para la conformación de uno o varios nodos. Los racks proporcionan un manejo más eficiente del espacio que las torres. Los Racks son medidos en un esquema llamado “rack units” (unidades rack) que es abreviado como RU o simplemente como U, se utiliza un número para indicar las unidades rack de alto. Los tamaños de los Racks se encuentran

estandarizados. Con un sistema rack 2U se pueden colocar 20 nodos en un espacio de 83 pulgadas de alto.
La circulación del aire caliente en estos sistemas es de adelante hacia atrás, mientras que en las torres es de abajo hacia arriba.
Los racks y las cajas racks son más costosas que las torres. La figura 4.5 muestra un sistema que emplea racks:

Figura 4.5 Cluster ubicado en un rack [RACK2006]

Además de la ubicación de las partes de los nodos hay que tener ciertas consideraciones con la adecuación del sistema total. Debe realizarse una planeación cuidadosa teniendo en cuenta el cableado, la refrigeración y el acceso físico a los equipos, si los equipos están demasiado cerca se tienen problemas de refrigeración, problemas de identificación de cableado y/o dificultades para reparar los nodos.

Idealmente se debe tener acceso al frente y la espalda de cada nodo y poder reparar cada equipo sin mover los otros equipos, también se deben tener los cables de datos separados, por al menos un pie, de los cables de potencia por facilidad de manejo y para disminuir interferencias, aunque en la práctica la señal de 60 Hz no afecta señales digitales de alta frecuencia.

Consejos prácticos [SPECTOR2000]:

- Cuando se seleccione el lugar en el cual se ubicarán los equipos, debe quedar suficiente espacio para que una persona pueda realizar tareas de mantenimiento, actualizar hardware y trabajar con el cluster (programarlo).
- Si el número de equipos lo amerita adicionar un sistema de enfriamiento para evitar daños por calor.
- Asegurarse de que los cables de potencia estén bien instalados y no haya ningún peligro de cortos o sobrecargas en el sistema.
- Si es posible ubicar los nodos y equipos de Red en racks o anaqueles (si se utilizan torres) para facilitar el acceso a los componentes.
- Hay que estar seguros de que el piso puede resistir el peso del número de nodos y todos los elementos que componen el cluster. Esto es muy importante al usar Racks porque todo el peso de los equipos se concentra en un pequeño espacio.
- Si se va a instalar una gran cantidad de nodos se puede considerar un piso falso, en el cual se introduzca la mayoría del cableado de potencia y de interconexión del cluster, para una mayor organización y para evitar accidentes con los cables esparcidos.

4.2.3. Aire acondicionado

Cuando se construye un cluster pequeño no se necesita una habitación especial, pero a medida que aumenta el número de nodos se requiere un sistema de refrigeración.
La mayoría de los lugares con computadores se mantiene por debajo de 21 °C [SPECTOR2000].
Para determinar el calor producido por el cluster que es necesario disipar, se debe utilizar una ecuación que convierta el consumo de potencia eléctrica (medida en vatios) en unidades de calor, que son medidas en BTU (British Termal Units). Hay muchas variables termodinámicas implicadas en este tipo de cálculos, pero se pueden simplificar considerando que un Vatio de potencia es aproximadamente igual a 3.412 BTUs, por lo tanto se puede aplicar la siguiente ecuación [SPECTOR2000]:

Calor Generado (BTU) = Potencia Total X 3.412

Donde la *Potencia Total* es la suma de las potencias que consumen los elementos que conforman el Cluster.
Adicional al calor generado por los equipos se deben adicionar 300 BTU/H por cada persona que trabaje en el área [SPECTOR2000].

La humedad también se debe tener en cuenta, con alta humedad la condensación es un problema, si por el contrario la humedad es baja aumenta la electricidad estática. Los rangos recomendados están entre un 40% a 60% de humedad [SPECTOR2000].

4.2.4. Potencia

En grandes cluster hay un significativo consumo de potencia, es importante contar con una infraestructura y un cableado eléctrico organizado, que suministra la energía necesaria al cluster evitando cortos, sobrecargas, picos de voltajes u otras fluctuaciones que puedan dañar los equipos. También es recomendable contar con una batería o UPS (Uninterruptible Power System) para evitar la pérdida de información cuando ocurran fallas eléctricas [SPECTOR2000].

4.3. Selección del Software del Cluster de Computadoras

La primera generación de sistemas de programación paralela sobre estaciones de trabajo en red fueron paquetes comerciales propietarios lentos con soporte a pocas arquitecturas.
En las siguientes generaciones surgieron soluciones de código abierto de diferentes clases, entre ellas librerías para programación paralela y modificaciones a núcleos de sistemas operativos (conocidas como parches para recompilar el núcleo del S.O. [SPECTOR2000].

A continuación se presenta un resumen de los componentes software más empleados en un cluster de computadoras:

4.3.1. Modificaciones al sistema operativo

- **OpenMosix**

Es un paquete software que modifica el núcleo de Linux para que computadores en red se comporten como un cluster; permite distribuir los programas de usuario, archivos y otros recursos de manera transparente y sin cambios adicionales mediante un balanceador de carga automático gracias al cual se pueden agregar o retirar equipos sin detener la ejecución. La carga es distribuida entre los nodos según su conexión y velocidad de procesador, el algoritmo de equilibrio de carga encargado de migrar los procesos puede ser configurado manualmente en cualquier momento.
Proporciona un sistema de gestión de archivos llamado oMFS (openMosix File-System) orientado a aplicaciones HPC (High Performace Computing) que incorpora funciones de consistencia de cache, tiempo y enlace [BUYTAERT2003].

4.3.2. Software sin modificaciones al sistema operativo

Para programar un cluster se puede utilizar un lenguaje de programación secuencial con una librería de programación en paralelo. Las librerías de programación paralela evitan que el programador tenga que preocuparse por la implementación de la comunicación entre los nodos. Se encuentran disponibles para un conjunto limitado de lenguajes de programación, como C, C++ y FORTRAN.

Entre las librerías más usadas encontramos:

- **PVM, Parallel Virtual Machine,** Maquina Virtual Paralela

Fue desarrollado en Oak Ridge National Labs. Es un sistema software con el cual los usuarios pueden configurar una estación de trabajo para que envíe subprocesos a otras máquinas, creando un ambiente virtual encapsulado para correr programas paralelos, sobre diferentes plataformas hardware. Este ambiente brinda acceso a la información de

los procesos que se encuentran corriendo mediante directivas de línea de comando. Cada usuario puede construir su propio ambiente controlado desde un equipo y lanzar subprocesos hacia otras máquinas. Es posible monitorear el comportamiento del entorno con herramientas graficas como XPVM [XPVM1994], además se dispone de entornos de desarrollo de aplicaciones paralelas (basados en PVM) como PADE [DEVANEY1995].

Algunos autores opinan que no ha sido tan seguido como la librería MPI debido a que tiene menos primitivas y es compatible con un menor número de plataformas.

- **MPI, Message Passing Interface,** Interfaz de Paso de Mensajes.

Es uno de los estándares de facto para diferentes arquitecturas hardware que permiten la programación en paralelo con el paradigma de paso de mensajes. Es un estándar abierto, es decir, se encuentra publicada una implementación de referencia. Todos los miembros del consorcio MPI pueden tomar la implementación de referencia y optimizarla para sus plataformas hardware/software particulares teniendo en cuenta que no pueden modificar la API y la estructura de llamadas [MPI2006].
Las dos implementaciones libres más populares de MPI son “MPICH” (MPI Chamelot) y “LAM” (Local Area Multicomputer). Las dos son versiones completas de MPI, están basadas en la implementación de referencia MPI1.1 y permiten que los programas MPI sean ejecutados implementando la comunicación con sockets UDP/TCP. LAM corre en máquinas Unix pero no funciona en Windows; MPICH esta disponible tanto para entornos Unix como para Windows NT.
Existe una implementación de MPI2.0 que es el último estándar disponible en la página oficial, llamada AFMP (Aggregate function MPI), tiene una curva de aprendizaje alta debido a que incorpora otros paradigmas de comunicación que lo hacen diferente al MPI1.1.

Utilizando paso de mensajes con MPI se han creado diferentes librerías Matemáticas muy útiles en el uso de clusters como PLAPACK [PLAPACK] que es una librería de Algebra Lineal y FFTW [FFTW2006] que realiza transformadas rápidas de Fourier.

- **MPE, Multi-Processing Enviroment,** Entorno de Multiprocesamiento

La librería MPE extiende a MPI, proporciona facilidades adicionales como librerías para creación de archivos log, herramientas y librerías de visualización gráfica, rutinas para convertir en secuencial secciones de código paralelo. Puede ser usada con cualquier implementación de MPI [GROPP2003].

- **SPRNG, Scalable Parallel Random Number Generator,** Generador de Números Aleatorios paralelo Escalable

Es una librería que proporciona seis diferentes formas de generar números aleatorios para usar con programas paralelos. [SPRNG2003].

- **HDF5, Hierarchical Data Format 5,** Formato de Jerarquia de Datos 5

Es un paquete software que tiene utilidades específicamente diseñadas para almacenar datos científicos, soporta archivos muy grandes y maneja un almacenamiento eficiente en sistemas de cómputo paralelo [HDF52006].

- **OpenMP Open Multi Processing,** Multiprocesamiento Abierto

Es una especificación que define un conjunto de directivas de compilación, librerías de rutinas y variables de entorno. Es utilizado en implementaciones de paralelismo orientadas al uso de memoria compartida entre procesos teniendo una granularidad gruesa, esto contrario a la tendencia existente en el pasado de que este tipo de paralelismo solo era adecuadamente implementable para paralelismo de tareas (granularidad fina). Se emplea en sistemas SMP y esta disponible para ambientes Unix y Windows [OPENMP2004].

Además de las anteriores librerías que se utilizan con lenguajes secuenciales también se han desarrollado lenguajes para programación en paralelo como: Cilk que es un lenguaje para programación paralela multihilo basado en C, diseñado para utilizarse como un lenguaje de propósito general pero es específicamente efectivo para paralelismo asíncrono, que puede ser difícil de escribir utilizando un estilo de paso de mensajes.

Adiciona funciones especificas para la comunicación y migración, estas son interpretadas por un precompilador (cilck2) que genera código fuente C, este es compilado por gcc (compilador estándar de C) y para su ensamblaje debe ser enlazado mediante las librerías de Cilk Runtime, permitiendo que sea gestionado por el balanceador de carga y el manejador de Tareas [CILK2006].

4.3.3. Compiladores y depuradores:

Las distribuciones de Linux vienen con compiladores y depuradores por lo cual normalmente no es necesaria la instalación de programas adicionales para estos fines; las soluciones no compatibles con estas herramientas incluyen sus propios compiladores y/o depuradores, y en su mayoría son propietarias o demasiado especificas.

La versión de Red Hat 7.3 que se usa en el proyecto incorpora gcc y g++ que son compiladores para trabajar con C/C++ respectivamente y g77 para Fortran, además tiene gdb que es un debugger tradicional. Gdb puede usarse con un programa que utilice MPI especificando el proceso que se quiere depurar. Existen depuradores diseñados específicamente para código paralelo como: TotalView que es un producto comercial [TOTALVIEW2005] y PGDBG [PGDB2005] diseñado para programas paralelos escritos con MPI y OpenMP. Otras herramientas que muestran información importante de los programas son: TAU [SHENDE2005] que es una librería que permite encontrar dónde se esta gastando la mayoría del tiempo al ejecutar una aplicación, y PCL [PCL2003] que muestra información acerca del rendimiento (performance) de un programa.

4.3.4. Sistemas de archivos

En la ejecución de programas paralelos se necesita una copia del código compilado o del programa “ejecutable” en cada máquina o que todos los nodos accedan a ciertos archivos, para lograr esto se utilizan sistemas de archivos. En la selección del sistema de archivos se tienen en cuenta dos factores: la solución debe ser transparente a los usuarios, con los archivos disponibles para todos los nodos del cluster; y los datos tienen

que encontrarse tan rápidamente como el procesador de cada nodo los requiera [SPECTOR2000].

Si las aplicaciones que se van a ejecutar en el cluster dan como resultado grandes cantidades de datos o interactúan continuamente con el sistema de entrada/salida es necesario utilizar un sistema de archivos específico de alto rendimiento (high-performance).

La gestión de archivos involucra hardware y software. Hay dos tendencias en cuanto al manejo de archivos orientado a Hardware: NAS y SAN.

- **NAS Network Attached Storage,** Almacenamiento Ligado a Red

Consiste en un servidor dedicado encargado de las peticiones de archivos realizadas desde la red. Los servidores NAS tienden a ser servidores de archivos altamente optimizados pero por centralizar la información pueden presentar latencia. [NAS2006]

- **SAN Storage Area Network,** Red de Área de Almacenamiento.

Es una red independiente de almacenamiento de altas prestaciones que utiliza fibra óptica y proporciona acceso directo al hardware físico por ello el tipo de tráfico en una SAN es muy similar al de los discos duros como ATA y SCSI. Para manejar este tipo de redes también es necesario un software, un ejemplo con código abierto de este sistema es OpenGFS. [OPENGFS2004].

Entre las soluciones orientadas a software para la gestión de archivos encontramos los siguientes programas:

- **NFS Network File System,** Sistema de Archivos en Red.

Es un protocolo de sistema de archivos distribuido que permite a un nodo utilizar particiones de disco en equipos remotos como si fueran locales. No es un sistema de archivos de alto rendimiento pero permite compartir los archivos del cluster. Típicamente

se implementa configurando una computadora como servidor NFS desde la cual se distribuyen los archivos necesarios para la ejecución de programas. Generalmente se utiliza el nodo principal del cluster pero en algunas aplicaciones se usan máquinas diferentes para el servidor de NFS [NFS2006].
Como en el servidor NFS es donde residen los archivos de usuario, se debe asegurar que tiene suficiente capacidad de almacenamiento.

- ***ClusterNFS***

Es un conjunto de adiciones software para el servidor NFS. Es comúnmente utilizado en cluster en los que los nodos no tienen disco duro, pues las adiciones permiten que los clientes sin disco monten el mismo sistema de archivos del usuario root [CLUSNFS2000].

- ***Lustre***

Es un sistema de archivos diseñado para trabajar en sistemas formados por más de 10,000 nodos. Fue desarrollado y es mantenido por Cluster File System, Inc y esta disponible bajo licencia GPL, funciona para Kernel 2.4.x [LUSTRE2006].

- ***PVFS Parallel Virtual File System,*** Sistema de Archivos Virtual Paralelo

Proporciona un sistema de archivos paralelo de alto rendimiento, esta diseñado para distribuir datos entre los discos del cluster y trabaja tanto con programas paralelos como secuenciales. Es una solución disponible sin costo, desarrollada por el *Argonne National Laboratory* y la *Universidad Clemson* [PVFS2006].

Existen sistemas de gestión de archivos de proveedores comerciales pero debido a que su información técnica es reservada no es posible clasificarlos en soluciones orientadas a software y/o hardware, en estos casos el factor fundamental es la compatibilidad entre el sistema de gestión de archivos y la solución cluster que se adquiere al vendedor, por ejemplo IBM tiene el sistema de archivos General Parallel File System (*GPFS*) para trabajar con el cluster AIX 5L (también de IBM), el esquema de este sistema se muestra en la figura 4.6[IBMGPFS2006]:

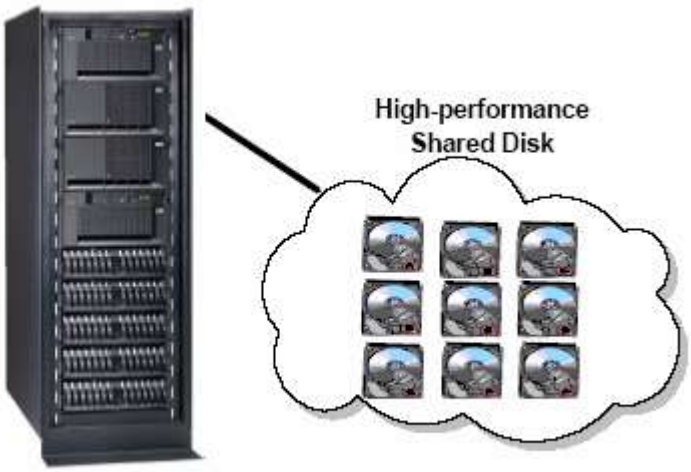


Figura 4.6. Cluster IBM AIX 5L con un sistema GPFS [IBMGPFS2006]

4.3.5. Software de scheduling (Despachamiento de tareas)

Este software permite que se reciban varios trabajos en el cluster y que se realicen a medida que existan recursos disponibles. Los trabajos se ponen en una pila y los usuarios pueden seguir el progreso de los trabajos en la pila o pueden eliminar un trabajo. Un administrador puede establecer prioridades y gestionar la pila. El software de despachamiento ayuda a utilizar de manera eficiente el hardware con que se cuenta y permite compartir recursos de una manera equitativa [SPECTOR2000].

Para Linux cluster existen dos alternativas comunes: Condor y PBS.

- **Condor**

Es un sistema de manejo de carga de trabajo especializado para tareas de cálculo intensivo, proporciona un mecanismo de colas de trabajo, políticas de despachamiento, monitoreo y gestión de recursos. Puede ser usado para trabajos secuenciales o paralelos y es compatible con entornos de Grid. [CONDOR2006].

- **PBS, Portable Batch System,** Sistema Portable

Fue desarrollado por la empresa Veridian, está disponible en dos formas OpenPBS o PBSPro. OpenPBS es la versión original de PBS, que es de código abierto y sin soporte, posteriormente se desarrolló PBSPro como un producto comercial [PBS2006].
OpenPBS usa el modelo cliente-servidor y está organizado como un conjunto de comandos a nivel de usuario que interactúan con tres servicios a nivel de sistema, estos servicios gestionan la ejecución y reciben las peticiones de usuario [OPENPBS2003].

Es posible adicionar características a las soportadas por estos programas mediante herramientas que incorporan funciones avanzadas de despachamiento necesarias en casos particulares como cluster de gran tamaño o con configuraciones especiales de red. Entre estas herramientas se encuentra MAUI que trabaja con una variedad de software de gestión de recursos y es empleado en cluster de gran tamaño que necesitan optimizar su *throughput* [7] [MAUI2006].

4.3.6. Software de gestión

Para el mantenimiento del cluster es necesario realizar tareas administrativas como creación de cuentas de usuario, instalación de software, monitoreo del estado de los nodos, entre otras. Estas tareas se facilitan por medio del uso de herramientas de gestión. Entre las herramientas más comunes se encuentran Ganglia y C3.

- **Ganglia**

Es un monitor de rendimiento para Cluster y Grids. Inicialmente Ganglia fue desarrollado en la Universidad de Berkeley por la división de ciencias computacionales como manera de enlazar cluster entre los campus de manera lógica. Ganglia es de código abierto, funciona sobre diferentes arquitecturas y utiliza un modelo cliente-servidor; debe ser instalado en una estación de trabajo dedicada a la gestión [GANGLIA2003].

[7] Rendimiento Efectivo o específico del procesamiento de un computador, combina las velocidades de entrada y salida de datos.

- **C3, The Cluster Command and Control,** Comandos y Control para Cluster

Es un conjunto de cerca de una docena de utilidades de línea de comando usadas para ejecutar tareas de gestión comunes, fue desarrollado en el Oak Ridge Nacional Laboratory [C32006].

4.3.7. Kits de cluster

Existen paquetes o "kits" que automatizan la instalación de todo el software necesario en un cluster. Algunos de estos kits tienen una distribución Linux incluida, por ejemplo Rocks [ROCKS2006] y otros son instalados sobre una distribución Linux como Oscar [OSCAR2005].

Algunos autores consideran como desventaja que el kit realice configuraciones automáticas debido a que no permite al usuario conocer como está configurado el cluster y qué se ha instalado, además pueden tener software adicional al que se desea trabajar, consumiendo recursos de manera innecesaria. También se debe tener en cuenta que para lograr el funcionamiento conjunto de los programas del kit no siempre los paquetes individuales se instalan como en la versión original y tampoco tienen la última versión de cada paquete, así que es necesario enterarse de qué tiene el kit y aprender los detalles de su manejo. A pesar de estos posibles inconvenientes varios autores consideran que la rapidez con la que se configura un cluster y se tiene listo para ejecutar programas tiene mayor importancia que las limitaciones de software que pueda tener.

Los kits de cluster más usados son Rocks, OSCAR y Scyld Beowulf.

- **Scyld Beowulf**

Fue desarrollado sobre RedHat e incluye un núcleo mejorado, herramientas y utilidades, es un producto comercial de Penguin Computing, una versión sin soporte esta disponible a un costo "nominal" en la página Web de la Compañía [PENGUIN2006].

- **Rocks**

Contiene tanto el sistema Linux como las herramientas de cluster, para su instalación se descarga una imagen ISO de la pagina Web y se crear un disco de instalación; este software esta diseñado para realizar una instalación automática y rápida, si se presenta alguna falla en la instalación es más rápido reinstalar el nodo que tratar de resolver el problema. Rocks es capaz de detectar el hardware empleado en el cluster, por ello es posible usarlo en clusters heterogéneos, requiere que todos los nodos tengan disco duro.
Es desarrollado y mantenido por NPACI (National Partnership for Advanced Computational Infrastructure) [NPACI2006].

- **Oscar**

Para su instalación es necesario tener una versión de Linux previamente instalada, descargar un paquete RPM de la página Web y realizar una serie de pasos de configuración que están documentados en el mismo paquete. Este tipo de instalación da mayor control sobre la configuración del cluster y permite que se utilicen diferentes versiones de Linux.
Oscar se instala en un nodo que será el servidor y en este se realiza una imagen del sistema para distribuirla a los discos de los otros nodos del cluster, por lo tanto es más sencillo de instalar si se emplea el mismo hardware en todos los equipos. Oscar puede funcionar en clusters formados por nodos sin disco duro como en nodos con disco duro; adicionalmente brinda una documentación en la que se describe el software de manera individual y la forma de manejarlo [OSCAR2005].

4.3.8. Cluster basado en CD-ROM

Los cluster basados en CD-ROM son una buena opción cuando sólo se quiere aprender acerca de programación en cluster o cuando se dispone de un cluster ocasionalmente. Con estas aplicaciones basta con reiniciar cada nodo desde el CD y se tiene listo un cluster para usar. Los discos de los nodos no se alteran porque todo lo necesario para el cluster se carga desde el CD, cuando se termina de trabajar con el cluster simplemente

de retira el CD y se reinicia el sistema. Debido a esto existen dificultades de almacenamiento, pero pueden solucionarse utilizando un sistema de almacenamiento dedicado y la realización de los cálculos sólo con el CD.

Ejemplos de estos sistemas son: ClusterKnoppix [KANNOPIX2004] y BCCD [BCD2005].

Capitulo 5 AMBIENTE EXPERIMENTAL

El proyecto planteado es el desarrollo de un “puente” entre una aplicación Web y el software (no comercial) de un cluster de alto rendimiento, permitiendo la programación del cluster a través de un portal Web desde el cual el usuario puede ejecutar programas paralelos y obtener sus resultados, como se muestra en la figura 5.1.

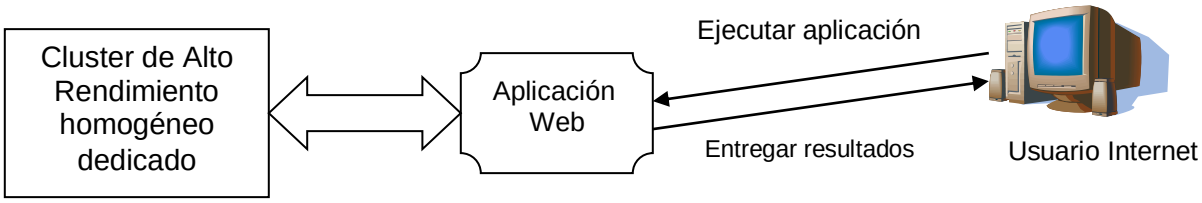


Figura 5.1 Esquema general de la solución planteada

El cluster es homogéneo, dedicado y con software no comercial; debido a que la arquitectura hardware esta preestablecida por los equipos asignados al proyecto, antes de definir los requisitos de la aplicación Web es necesario definir el software del cluster. En este capítulo se describe el cluster configurado, el software seleccionado y posteriormente los requerimientos definidos para la aplicación Web.

5.1. Descripción del Cluster

Inicialmente se describe la configuración hardware de los equipos y después el software seleccionado para la implementación del cluster.

5.1.1. Descripción de los nodos

El cluster configurado consta de 5 equipos como se muestra en la figura 5.2:

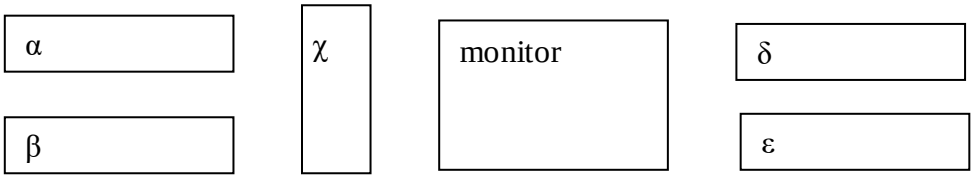


Figura 5.2 Ubicación física de los equipos del Cluster

Para identificar fácilmente cada nodo se le asigna una letra del alfabeto griego, la descripción del hardware de cada equipo se muestra a continuación:

α

Memoria	64 Mega Bytes
Disco Duro	4 Giga Bytes
Procesador	Pentium-MMX 200Mhz
Placa Base	PAM0057I
Tarjeta de Red	3Com 3c90X
MAC	00:60:08:A3:C9:06

δ

Memoria	64 Mega Bytes
Disco Duro	8 Giga Bytes
Procesador	Pentium-MMX 200Mhz
Placa Base	PAM0057I
Tarjeta de Red	3Com 3c90X
MAC	00:60:08:A3:C7:5D

β

Memoria	64 Mega Bytes
Disco Duro	4 Giga Bytes
Procesador	Pentium-MMX 200Mhz
Placa Base	PAM0057I
Tarjeta de Red	3Com 3c90X
MAC	00:60:08:A3:C6:4F

ε

Memoria	32 Mega Bytes
Disco Duro	8.4Giga Bytes
Procesador	Pentium- MMX 200Mhz
Placa Base	PAM0057I
Tarjeta de Red	3Com 3c90X
MAC	00:60:08:A3:C3:94

χ

Memoria	32 Mega Bytes
Disco Duro	8 Giga Bytes
Procesador	AMD K6 233Mhz
Placa Base	PAM0073I
Tarjeta de Red	3com 3C90X
MAC	00:10:4B:CC:F4:AE

Los equipos deben tener conectados sus respectivos teclados por que las board no soportan arranque sin ellos.

La arquitectura del cluster consiste en los cinco equipos nombrados anteriormente unidos a través de un switch ethernet con una topología de red en bus. Se secciona a χ como el nodo que contiene la aplicación que recibe las órdenes desde la Web y reparte los trabajos entre los nodos del cluster, como se muestra la figura 5.3.

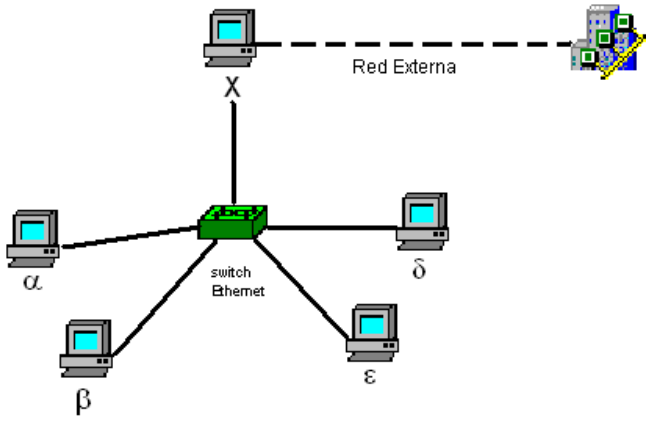


Figura 5.3 Configuración de red

5.1.2. Software Empleado

Para realizar las funciones de cluster se utiliza el kit OSCAR, mencionado en el capitulo tres de este documento, el cual está orientado a la creación de Cluster de alto rendimiento (HPC). OSCAR brinda un aplicación con un entorno gráfico que instala software para cluster de diferentes propósitos (librerías paralelas, software de gestión, etc) de forma automática; para emplear esta aplicación se debe seleccionar un nodo en el cual la aplicación instala los programas y crea una imagen del sistema operativo que es enviada a través de la red a los otros nodos del cluster. [OSCAR2005]

Los nodos tienen la versión 2.0 de OSCAR que se adapta a las características hardware de los equipos suministrados, es compatible con: RedHat 7.2, RedHat 7.3 y Mandrake 8.2 [OSCAR2005]. Esta aplicación instala herramientas de cluster de forma automática, la instalación de las herramientas del cluster se configura de forma manual sólo una vez y se instala automáticamente en los nodos que se adicionen al cluster, es un paquete RPM, no es necesario recompilar el núcleo de Linux. También permite adicionar y eliminar nodos

sin la reconfiguración de todo el sistema y a diferencia de otros kits de cluster, se ajusta al hardware suministrado. Esta versión instala los siguientes programas [OSCAR2005]:

- C3 versión 3.1: Es un conjunto de cerca de una docena de utilidades de línea de comando usada para ejecutar tareas de gestión comunes.

- HDF5[8]: Es un paquete software que tiene utilidades específicamente diseñadas para almacenar datos científicos.

- LAM/MPI (lam-6.5.6): Es un ambiente de programación MPI y sistema de desarrollo para computadores heterogéneos sobre una red.

- MPICH (4.5.6): Es una implementación completa de mpi 1.1.

- PVM versión 3.4: Es un paquete software que permite una colección heterogénea de computadores con sistemas operativo windows o unix, interconectados por una red, ser usados como un único computador paralelo.

- OpenPBS v2.3: Sirve como un despachador de tareas y tiene funcionalidades de sistema batch.

- Pfilter: es un programa para filtrar los paquetes de red, se utiliza para evitar que el tráfico de la red externa llegue a los nodos.

- SIS (System Imager): Es la herramienta que permite que Oscar instale Linux a través de la Red. Se crea una imagen del sistema y se puede transmitir por la Red de interconexión. SIS proporciona una base de datos desde la cual Oscar obtiene la información de configuración del cluster (número de nodos, direcciones IP, MACs).

- Switcher: es el programa que permite que Oscar configure los archivos Linux necesarios para que funcionen los programas que ha elegido el usuario del cluster.

[8] En esta versión de Oscar no soporta operación paralela.

5.2. Pruebas en el cluster

5.2.1. Descripción de las pruebas:

- **Prueba 1 (P.1)**

Objetivo:

Determinar mediante diferentes configuraciones de sistemas paralelos (variando la cantidad de nodos) las características principales del cluster construido mediante la observación de su comportamiento frente a un programa en el que cada nodo ejecuta un proceso independiente de los que se ejecutan en los otros nodos, con comunicaciones escasas y de paquetes de datos pequeños.

Algoritmo: Se utiliza un programa que calcula el número π, implementado con dos algoritmos diferentes: uno secuencial que se ejecuta sobre el nodo con el mejor procesador, este es tomando como el mejor algoritmo secuencial y se usa de marco de comparación, y otro algoritmo paralelizado con la librería LAM-MPI para que sea ejecutado por dos o mas nodos.

Los algoritmos se basan en la integral definida de la función arco tangente: se realizan divisiones del intervalo comprendido entre 1 y 0, luego se calcula el área bajo la gráfica mediante un aproximación con rectángulos en los que un lado es el valor de la derivada del arco tangente en el punto medio y el otro lado es la subdivisión del intervalo.

Algoritmo Secuencial:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[  ] )
{
       register double ancho,intervalos;
       double sum;
       register int i;


       intervalos=atoi(argv[1]);
```

```
	ancho=1.0/intervalos;
	sum=0;
	for (i=0;i<intervalos;i++){
		register double x = (i+0.5) * ancho;
		sum += 4.0 / (1.0 + x*x );
		}
	sum*=ancho;
	printf("El resultado de la estimación del pi es: %2.10f",sum);
	return(0);
}
```

El algoritmo paralelizado:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[  ] )
{
	register double ancho,intervalos;
	double sum, lsum;
	register int i;
	int nproc, iproc;

	if (MPI_Init(&argc, &argv)!=MPI_SUCCESS) exit(1);
	MPI_Comm_size(MPI_COMM_WORLD, &nproc);
	MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
	intervalos= atoi(argv[1]); // cambia en las pruebas
	ancho=1.0/intervalos;
	lsum=0;
	for (i=iproc;i<intervalos;i+=nproc){
		register double x = (i+0.5) * ancho;
		lsum += 4.0 / (1.0 + x*x );
		}
	lsum*=ancho;
	MPI_Reduce(&lsum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
	if (iproc==0)
		printf("El resultado de la estimación del pi es: %2.10f",sum);
	MPI_Finalize();
	return(0);
}
```

Variables observadas: Las variables de interés en esta prueba son los tiempos de ejecución, secuencial y paralelo de los dos programas variando el número de nodos y la cantidad de intervalos usados por el programa.

Ejecución de la prueba:

Se selecciona cierto tamaño de problema y se empieza a ejecutar el algoritmo paralelo comenzado con dos nodos hasta emplear todo el cluster. Luego se incrementa el tamaño del problema y se repite el procedimiento anterior.

Para representar las variaciones en el tamaño del problema se aumentó el número de intervalos empleados para aproximar el área bajo la curva, puesto que cada uno representa una interacción adicional en el bucle *for* aumentando la cantidad de pasos de computación ejecutados.

El algoritmo secuencial se ejecuta en el nodo cabeza para medir el tiempo tomado en resolver el problema sobre un elemento de procesamiento.

Las unidades de los tiempos de ejecución son segundos. De los tiempos de ejecución obtenidos se calcularon las métricas de mejora en el rendimiento, sobrecarga y eficiencia; descritas en el capitulo 2. La Mejora en el rendimiento se calculó con la relación del tiempo tomado en resolver el problema sobre un elemento de procesamiento, dividida entre el tiempo requerido para resolver el mismo problema aumentando el número de nodos. La Sobrecarga total del sistema se calculó con la resta de los costos de ejecutar el programa en el cluster y el tiempo de ejecución sobre el nodo cabeza. La Eficiencia se obtuvo de la relación entre la mejora en el rendimiento y el número de elementos de procesamiento.

Resultados:

Los resultados de los tiempos de ejecución obtenidos variando el número de divisiones y la cantidad de nodos, para el cálculo del número π se encuentran tabulados en la tabla 5.1 y se muestran en la figura 5.4.

	Tiempos de ejecución				
Número de divisiones	**Nodo cabeza**	**Dos nodos**	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
1000000	1,141	0,901	14,671	22,292	18,834
5000000	2,867	1,703	15,184	12,985	26,647
10000000	4,880	2,876	11,921	13,832	24,258
50000000	21,672	11,216	22,293	16,006	27,681
100000000	42,914	21,493	25,417	18,678	28,914
500000000	210,335	105,657	76,879	60,659	61,274
1000000000	420,852	210,726	145,885	117,362	95,882

Tabla 5.1. Tiempos de ejecución cálculo de pi.

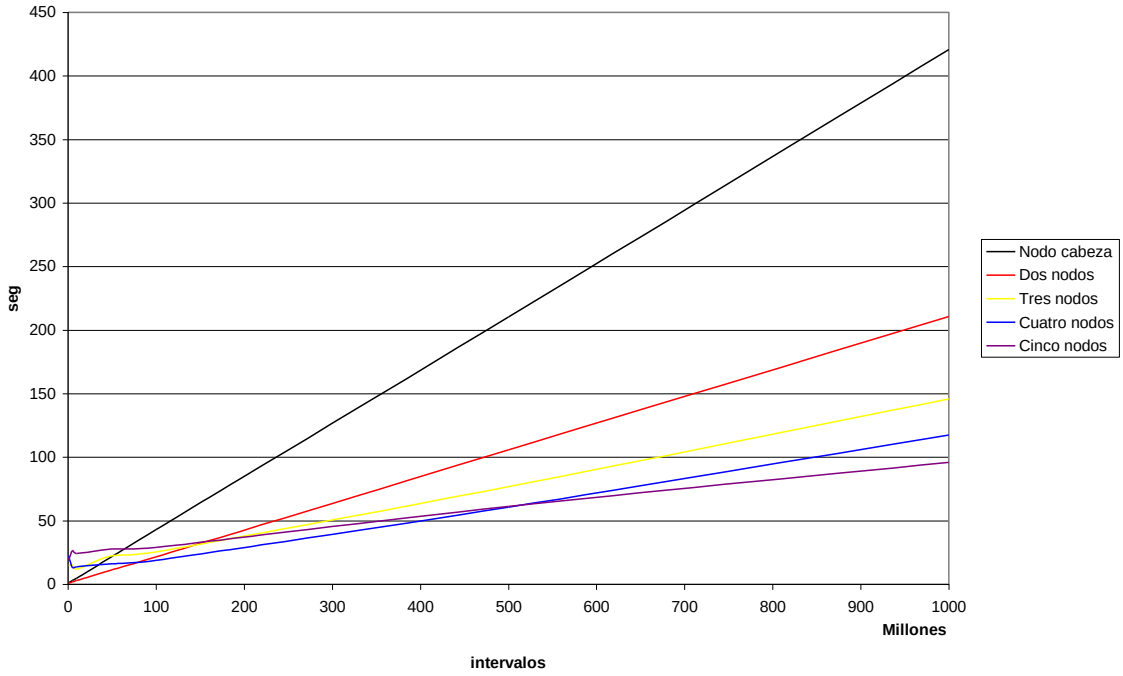


Figura 5.4. Tiempos de ejecución cálculo de pi

La tabla 5.2 contiene los valores de la mejora en el rendimiento calculada variando el número de nodos y el número de intervalos de integración, estos valores se visualizan en la figura 5.5:

	Mejora en el rendimiento			
Número de divisiones	**Dos nodos**	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
1000000	1,266	0,078	0,051	0,061
5000000	1,683	0,189	0,221	0,108
10000000	1,697	0,409	0,353	0,201
50000000	1,932	0,972	1,354	0,783
100000000	1,997	1,688	2,298	1,484
500000000	1,991	2,736	3,468	3,433
1000000000	1,997	2,885	3,586	4,389

Tabla 5.2. Mejora en el rendimiento del cálculo de pi.

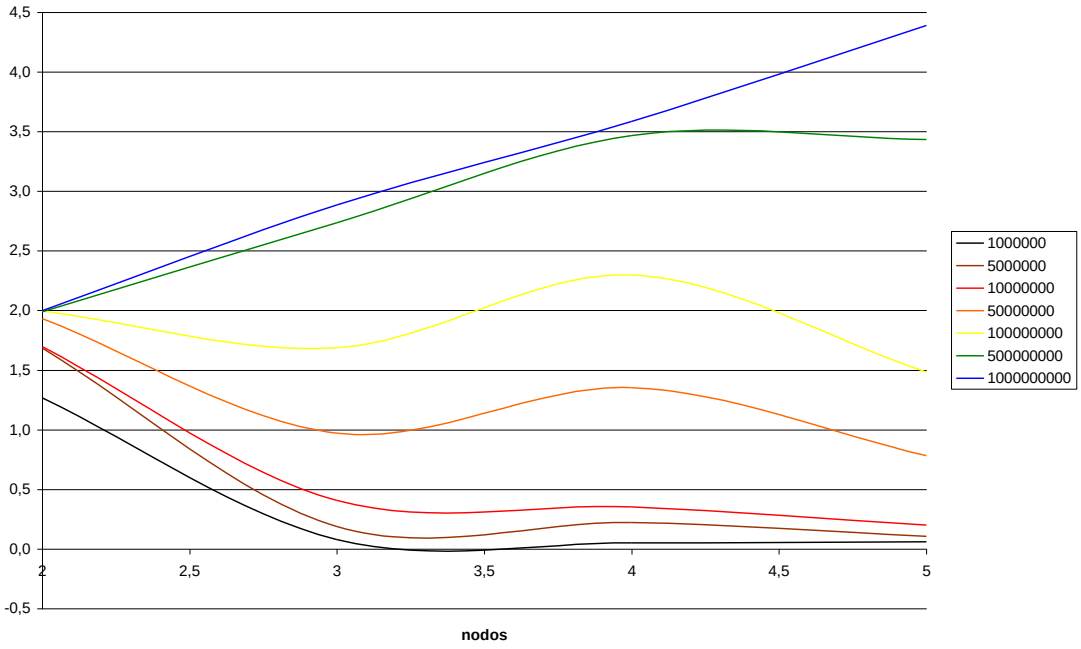


Figura 5.5. Mejora en el rendimiento contra número de nodos

La sobrecarga encontrada variando el número de nodos y el número de intervalos se encuentra en la tabla 5.3 y se grafica en la figura 5.6.

	Sobrecarga			
Número de divisiones	**Dos nodos**	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
1000000	0,661	42,871	88,027	93,029
5000000	0,540	42,686	49,074	130,370
10000000	0,873	30,884	50,447	116,410
50000000	0,759	45,207	42,353	116,735
100000000	0,071	33,337	31,797	101,654
500000000	0,979	20,302	32,300	96,037
1000000000	0,600	16,802	48,596	58,558

Tabla 5.3. Sobrecarga del cálculo de pi

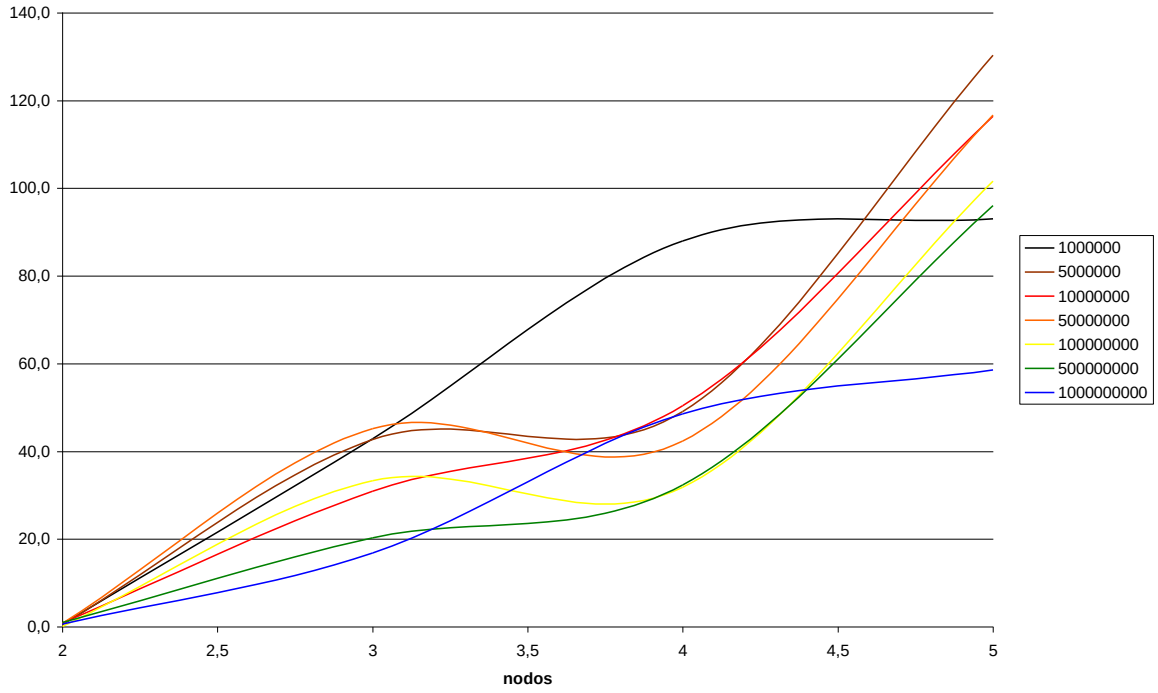


Figura 5.6. Sobrecarga contra número de nodos

La eficiencia calculada variando el número de nodos y el número de intervalos se encuentra en la tabla 5.4 y se grafica con respecto al número de intervalos en la figura 5.7 y con respecto al número de nodos en la figura 5.8.

	Eficiencia			
Número de divisiones	**Dos nodos**	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
1000000	0,633	0,026	0,013	0,012
5000000	0,842	0,063	0,055	0,022
10000000	0,848	0,136	0,088	0,040
50000000	0,966	0,324	0,338	0,157
100000000	0,998	0,563	0,574	0,297
500000000	0,995	0,912	0,867	0,687
1000000000	0,999	0,962	0,896	0,878

Tabla 5.4. Eficiencia del cálculo de pi

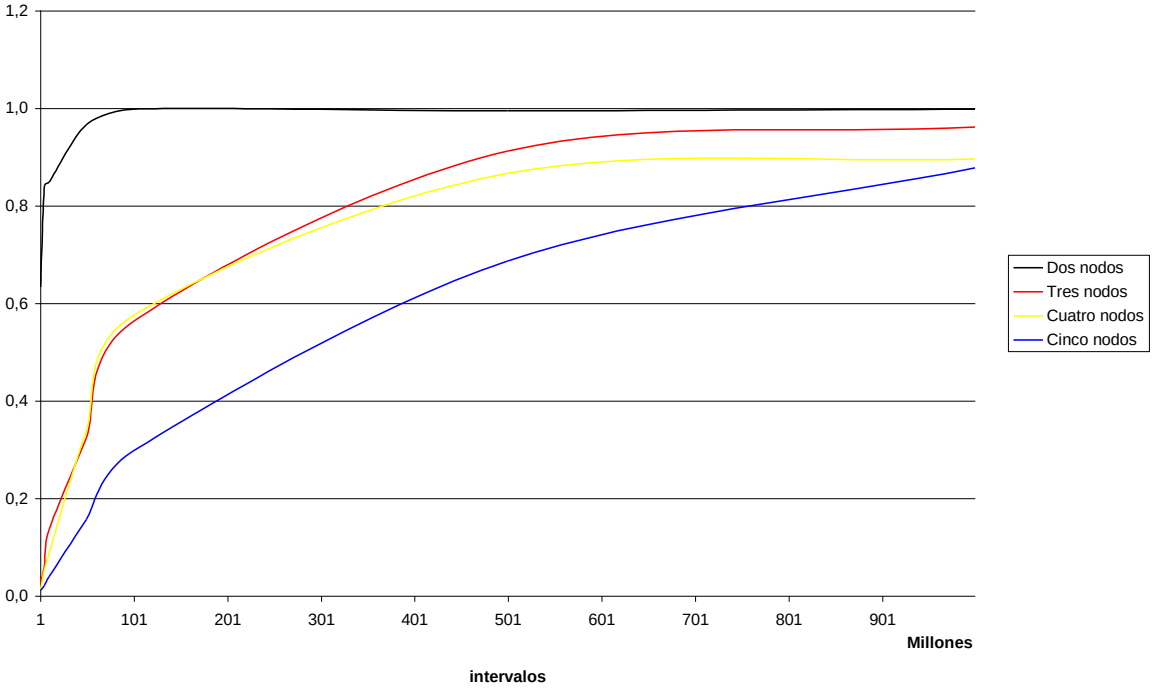


Figura 5.7. Eficiencia contra intervalos

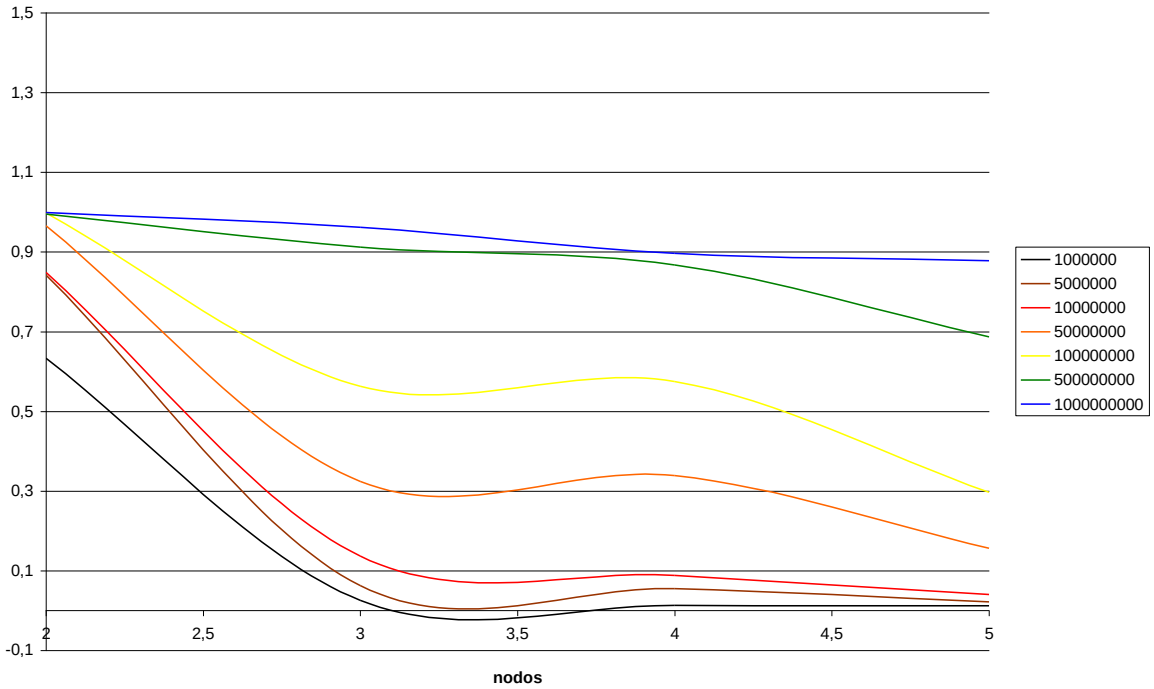


Figura 5.8. Eficiencia contra nodos

Análisis de Resultados:

De la tabla 5.1 y la figura 5.4 se observa la disminución en el tiempo de ejecución para un número de intervalos grande a medida que aumenta la cantidad de nodos. Para un número pequeño de intervalos dos nodos ejecutan el programa más rápidamente que uno solo, a diferencia de los otros sistemas que utilizan un tiempo mayor. Esto indica que el problema se ajusta adecuadamente a una solución paralela, porque sin importar el número de intervalos, el programa se ejecuta en menores tiempos que la solución secuencial en sólo un elemento de procesamiento. La cantidad de nodos que se debe usar depende de la cantidad de intervalos de la integral.

En la figura 5.5 se muestra que el problema presenta una mejora en el rendimiento con una pendiente creciente para cualquier cantidad de nodos del sistema, en el caso de 1000000000 de intervalos, este comportamiento también se observa con 5000000000 intervalos pero sólo hasta cuatro nodos. Para los otros intervalos no se observa un comportamiento proporcional a la cantidad de nodos. Con intervalos mayores prácticamente el código se distribuye uniformemente entre los nodos, esto es, el tiempo empleado en código no paralelizable se hace menor respecto al tiempo en el cual los nodos realizan ejecuciones logrando una mejora casi lineal.

La figura 5.6 muestra que los menores cambios de concavidad de la sobrecarga ocurren para 1000000 y 1000000000 hasta cuatro nodos; este comportamiento ocasiona que para cinco nodos tengan menor valor de sobrecarga y por lo tanto tienen menos pérdidas de tiempo en interacciones entre los nodos. Además, se muestra que con dos nodos la sobrecarga es prácticamente la misma. En este caso la sobrecarga es debida principalmente a los tiempos de comunicación y sincronización entre los procesos, lo que hace que para dos nodos se tenga prácticamente una comunicación full dúplex y el valor de sobrecarga es constante sin importar la cantidad de intervalos.

Al incrementar el tamaño del problema la eficiencia aumenta, como se muestra en la figura 5.7; la eficiencia más cercana a la unidad se obtiene con dos nodos sin importar el intervalo evaluado.
En la figura 5.8 se observa una disminución de la eficiencia en general al aumentar el número de nodos; pero la pendiente es menos pronunciada con el intervalo de

1000000000. A partir de las figuras 5.5, 5.7 y 5.8; se deduce que el sistema es escalable a partir de 1000000000.

Prueba 2 (P.2)

Objetivo: Determinar mediante diferentes configuraciones de sistemas paralelos las características principales del cluster construido, por medio de una aplicación que intercambie cantidades significativas de información entre los nodos con el fin de observar la influencia de los dispositivos de interconexión de la red en el sistema.

Algoritmo: Se emplea el programa parallel Bladeenc para la conversión de música de formato wav a mp3, su implementación paralelizada se basa en la librería mpi, funciona con la estructura de maestro y esclavo, debido a esto no se desarrollaron pruebas con dos nodos ya que es inviable tener un maestro con un solo esclavo [BLADEENC2002], se toma como mejor algoritmo secuencial una versión del programa que no utiliza referencias a la librería mpi ejecutandose sobre el nodo de mejor procesador.

Variables observadas: Las variables de interés en esta prueba son los tiempos de ejecución de la aplicación variando el número de nodos y el tamaño del problema.

Ejecución de la prueba:

El archivo de audio seleccionado para realizar las pruebas es un fragmento del Himno Nacional de Colombia con una duración de 3 minutos y 39 segundos y un tamaño de 38651960 bytes. Para representar las variaciones en el tamaño del problema se dividió el archivo en fragmentos de una cuarta parte, media parte y tres cuartas partes de duración, por lo cual los resultados se tabulan con 25%, 50%, 75% y 100% del tamaño total de la canción. Al igual que el cálculo del número Pi, para la ejecución de este programa se midieron los tiempos para un tamaño del problema dado variando el número de nodos.

Resultados:

Los resultados de los tiempos de ejecución obtenidos variando el tamaño del archivo de audio y la cantidad de nodos se encuentran tabulados en la tabla 5.5 y se muestran en la figura 5.9.

	Tiempos de ejecución			
Número de divisiones	**Nodo cabeza**	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
0,25	90,176	221,672	79,603	96,076
0,50	178,704	422,201	179,975	150,100
0,75	224,988	585,047	313,915	190,127
1,00	360,436	713,308	484,615	276,609

Tabla 5.5. Tiempos de ejecución bladeenc paralelo

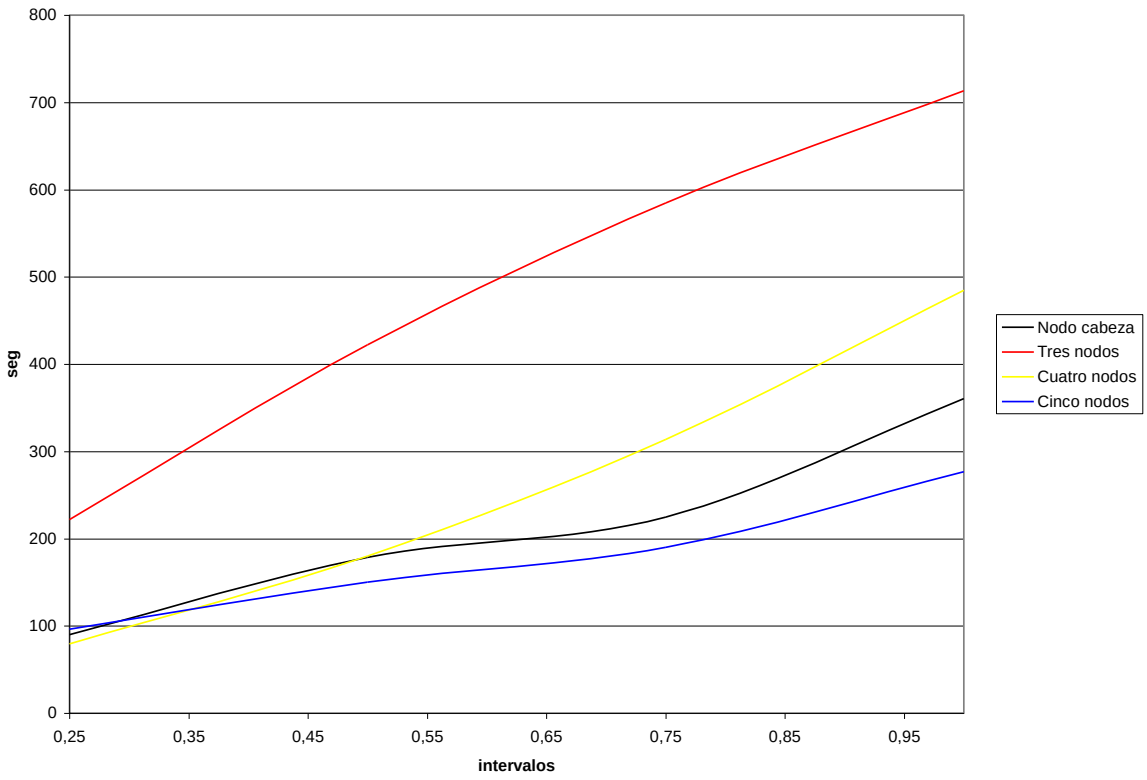


Figura 5.9. Tiempos de ejecución conversión de wav a mp3 con Bladeenc

La tabla 5.6 contiene los valores de la mejora en el rendimiento calculada variando el número de nodos y el tamaño del archivo de audio, estos valores se visualizan en la figura 5.10:

	Mejora en el rendimiento		
Número de divisiones	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
0,25	0,407	1,133	0,939
0,50	0,423	0,993	1,191
0,75	0,385	0,717	1,183
1,00	0,505	0,744	1,303

Tabla 5.6. Mejora en el rendimiento

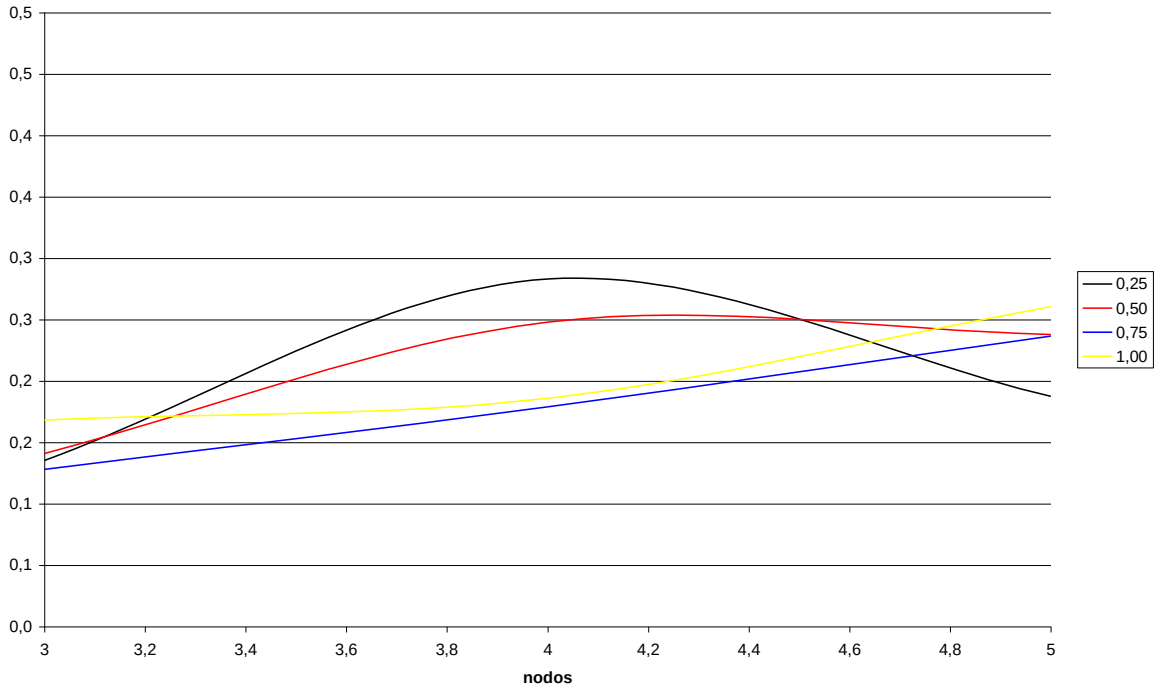


Figura 5.10. Mejora en el rendimiento contra número de nodos

La sobrecarga encontrada variando el número de nodos y el número de intervalos se encuentra en la tabla 5.7 y se grafica en la figura 5.11.

	Sobrecarga		
Número de divisiones	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
0,25	574,840	228,236	390,204
0,50	1087,899	541,196	571,796
0,75	1530,153	1030,672	725,647
1,00	1779,488	1578,024	1022,609

Tabla 5.7. Sobrecarga

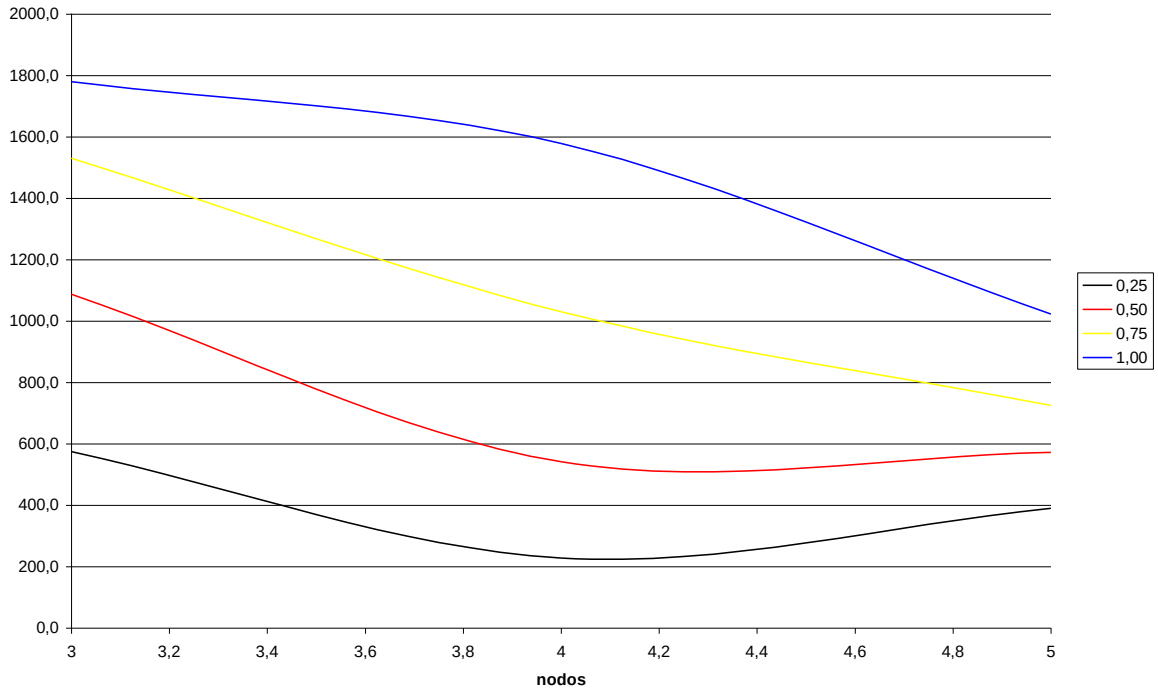


Figura 5.11. Sobrecarga contra número de nodos

Los resultados de eficiencia obtenidos variando el número de nodos y el tamaño del archivo se tabulan en la tabla 5.8 y se grafican en la figuras 5.12 y 5.13 con relación al tamaño del problema y al número de nodos respectivamente.

	Eficiencia		
Número de divisiones	**Tres nodos**	**Cuatro nodos**	**Cinco nodos**
0,25	0,136	0,283	0,188
0,50	0,141	0,248	0,238
0,75	0,128	0,179	0,237
1,00	0,168	0,186	0,261

Tabla 5.8. Eficiencia

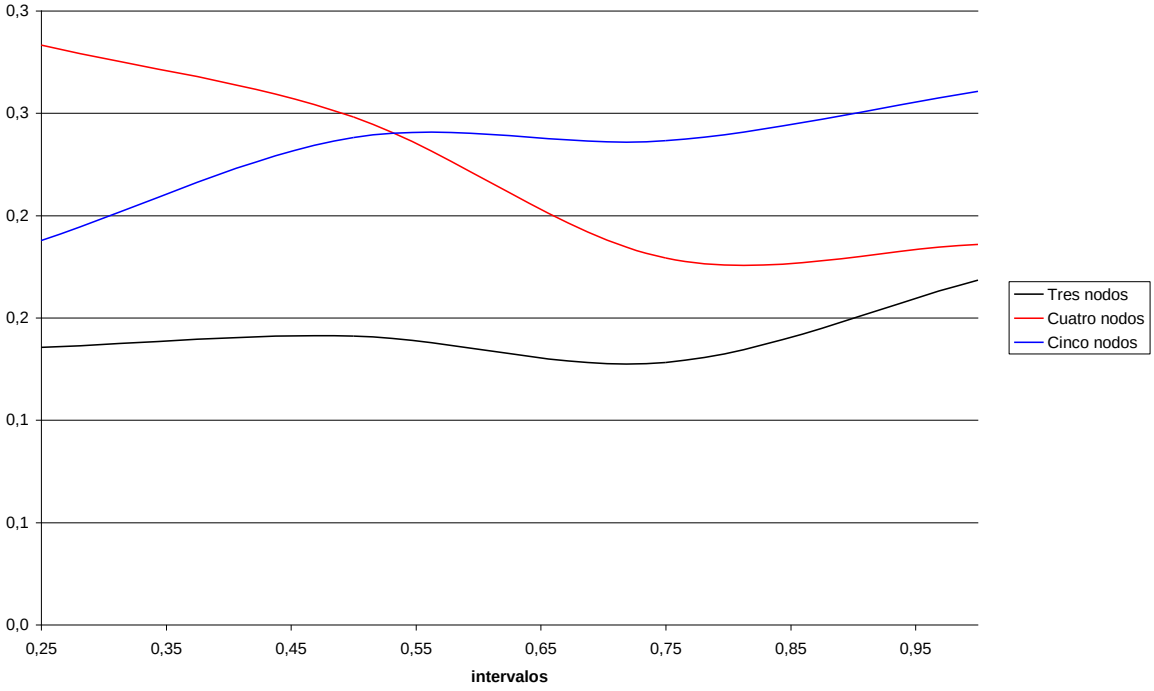


Figura 5.12. Eficiencia contra tamaño

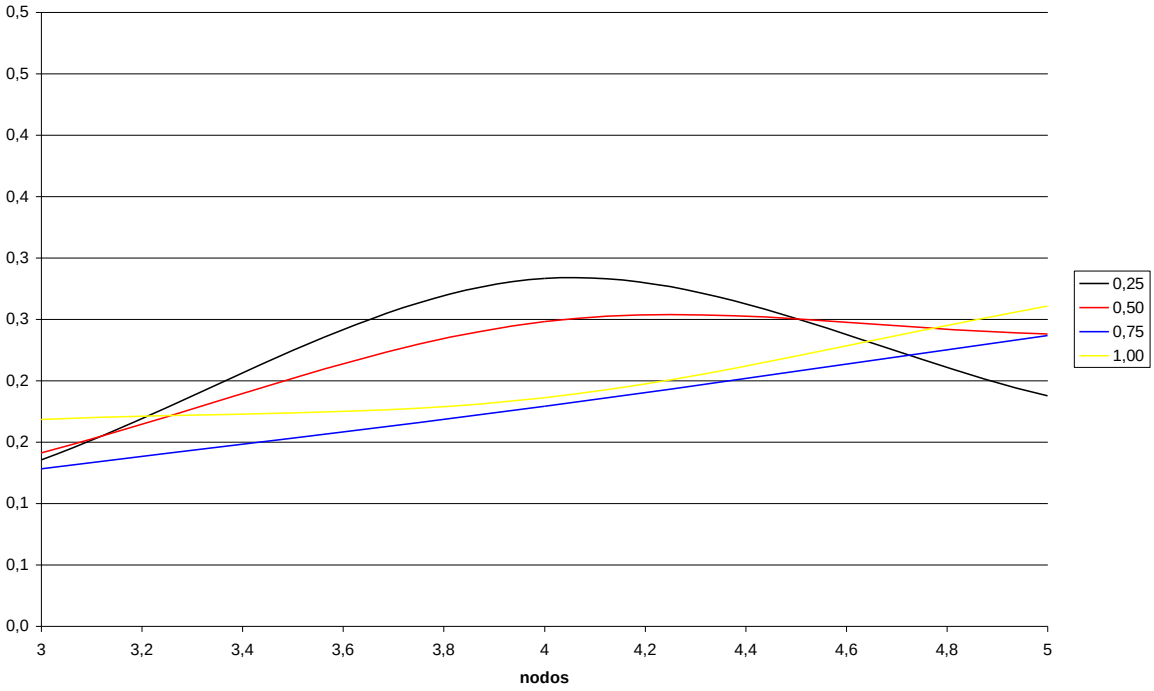


Figura 5.13. Eficiencia contra nodos

Análisis de Resultados:

Se aprecia en la figura 5.9 que los tiempos de ejecución para cinco nodos son los únicos de la implementación paralela que se mantienen por debajo de los de la ejecución con sólo un nodo, pero no son tan significativamente menores como los presentados por el problema del cálculo del número Pi.
Además la pendiente de la grafica de la curva de cinco nodos y la curva de un solo nodo es muy similar, indicando que el problema no se ajusta adecuadamente a una solución paralela para una cantidad baja de nodos.
En la figura 5.10 se muestra que el problema presenta una mejora en el rendimiento con una pendiente creciente para un archivo de un tamaño del 75% y del 100% del tamaño original.

En la figura 5.11 se muestra que la sobrecarga es mayor cuando se emplean tres nodos, sin importar el tamaño del archivo de sonido, por otra parte, cuando se tiene en cuenta el tamaño de la canción se obtiene que la mayor sobrecarga ocurre cuando se convierte la canción completa. De lo anterior se infiere que a un mayor tamaño de la porción que debe procesar cada nodo aumenta la sobrecarga, este efecto se combina con el observado en el la figura 5.10, lo que genera el cambio de pendiente el las curvas de 25% y 50% respectivamente. Para los otros dos valores (25 y 50) se observa una disminución en la mejora del rendimiento que indica una limitación al aprovechar todo el grado de paralelismo, esto es debido a la naturaleza del algoritmo el cual no puede ser distribuido en partes iguales sobre cada elemento de procesamiento. En las figuras 5.12 y 5.13 el valor de la eficiencia es bajo por lo que el uso efectivo de los nodos no es el óptimo, esto nos confirma que la solución no se adecua a este sistema.

- **Prueba 3. P.3**

Objetivo: Medir la cantidad de usuarios soportados por el sistema total, la aplicación Web y el cluster de prueba que se está empleando, por medio de un programa que ejecuta procesos independientes en cada nodo.

Algoritmo: Se utiliza el programa paralelo que calcula el número π con 1000000000 intervalos de integración para lograr un mayor uso del procesador, el código se muestra a continuación:

Pi.c

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[  ] )
{
       register double ancho,intervalos;
       double sum, lsum;
       register int i;
       int nproc, iproc;

       if (MPI_Init(&argc, &argv)!=MPI_SUCCESS) exit(1);
       MPI_Comm_size(MPI_COMM_WORLD, &nproc);
       MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
       intervalos= 1000000000;
       ancho=1.0/intervalos;
       lsum=0;
       for (i=iproc;i<intervalos;i+=nproc){
              register double x = (i+0.5) * ancho;
              lsum += 4.0 / (1.0 + x*x );
              }
       lsum*=ancho;
       MPI_Reduce(&lsum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
       if (iproc==0)
              printf("El resultado de la estimación del pi es: %2.10f",sum);
       MPI_Finalize();
       return(0);
}
```

Variables observadas: Las variables de interés en esta prueba son el tiempo de ejecución del programa y el número de usuarios simultáneos que soporta.

Ejecución de la prueba:

La prueba se realiza incrementando el número de usuarios que ejecutan el programa, de manera simultánea en el cluster por medio de la aplicación Web.
Los tiempos se miden en segundos y se cuentan a partir del momento en que los usuarios ejecutan la aplicación hasta que obtienen el resultado o se despliega el mensaje que indica que el cluster no puede ejecutar las peticiones.

Resultados:
Los tiempos de ejecución obtenidos se muestran en la tabla 5.9

	usuario 1	usuario 2	usuario 3	promedio
un usuario	111,78			111,78
dos usuarios	110,21	196,28		153,245
tres usuarios	248,04	-----	------	248,04

Tabla 5.9. Tiempos de ejecución prueba 3

Los tiempos de ejecución promedio variando el número de usuarios se muestran en la figura 5.14:

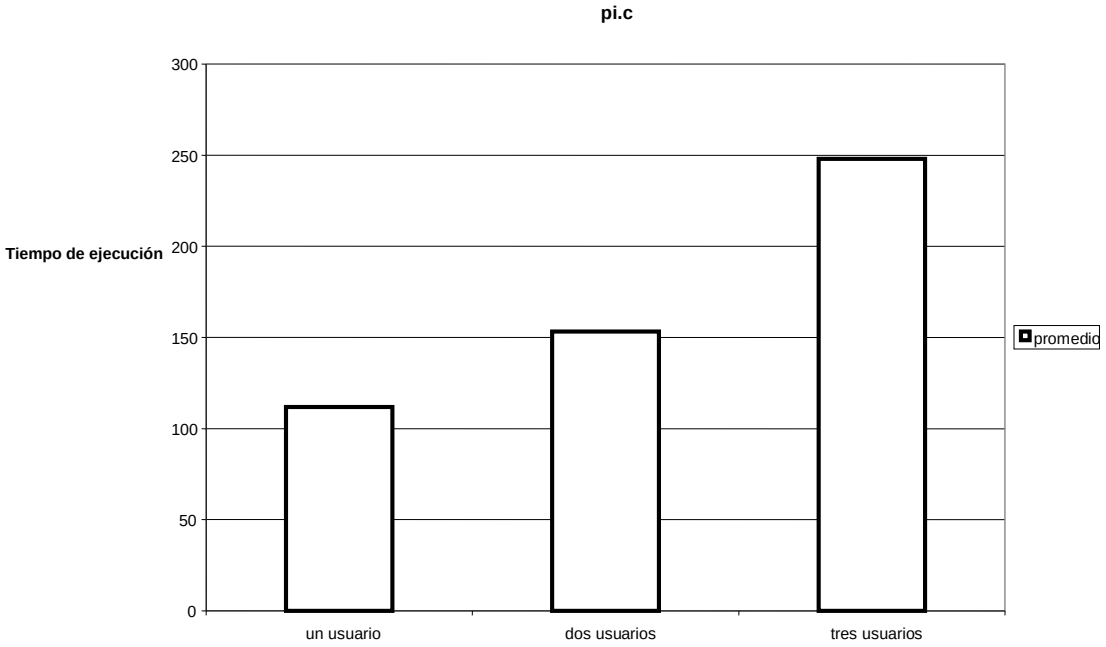


Figura 5.14. Tiempos de ejecución prueba 3.

Análisis de Resultados:

Cuando 3 usuarios ejecutaron este código, sólo uno de ellos pudo obtener el resultado, lo cual indica que el sistema se satura y no puede continuar con las otras ejecuciones. Esto indica que recursos limitados no son convenientes para una ejecución remota vía Web de manera simultánea.

Observado el tiempo de ejecución para el mismo programa pero con solo un usuario directamente en el cluster que fue de 95,882 segundos y comparándolo con el logrado por medio del acceso remoto que es de: 111,78 segundos, se muestra que el comportamiento de la red de comunicaciones ocasiona un retardo de 15,898 segundos por lo cual es recomendable que el ambiente de operación de la aplicación no sea una red de baja velocidad.

- **Prueba 4. P.4**

Objetivo: Medir la cantidad de usuarios soportados por el sistema total, la aplicación Web y el cluster de prueba que se está empleando, por medio de un programa que ejecuta procesos independientes en cada nodo pero con un procesamiento menor para permitir un mayor número de usuarios. Se emplea un código que combina funciones de comunicación de uno a todos y de todos a uno, y en cada nodo ser realiza la multiplicación y la suma en un rango determinado para calcular una integral, combinando comunicación y procesamiento.

Algoritmo: Se utiliza un código para calcular la integral de x^2 dividiendo el área en rectángulos, calculando el área de cada rectángulo y luego sumando los resultados. Para encontrar el alto se evalúa la función en el valor del centro del rectángulo y este valor se multiplica por el ancho para hallar el área.

integral.c

```
#include "mpi.h"
#include <stdio.h>

#define f(x) ((x) * (x))

int main( int argc, char * argv[  ] )
```

```
{
   /* MPI variables */
   int noProces, pId;

   /* variables */
   int         i, numero;
   double      area, at, alto, inferior, ancho, total, rango;
   double      liminferior, limsuperior;

   /* MPI */
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &noProces);
   MPI_Comm_rank(MPI_COMM_WORLD, &pId);

   if (pId == 0)         /* Si 0, reune los parámetros */

   {
      numero=100;
      liminferior=2;
      limsuperior=5;
   }

   MPI_Bcast(&numero, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&liminferior, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(&limsuperior, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   /* ajuste de tamaño*/
   rango = (limsuperior - liminferior) / noProcesses;
   width = rango / numero;
   inferior = liminferior + rango * processId;

   /* calcula el area del subproblema */
   area = 0.0;
   for (i = 0; i < numero; i++)
   {  at = inferior + i * ancho + ancho / 2.0;
      alto = f(at);

      area = area + ancho * alto;
   }

   MPI_Reduce(&area, &total, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

   /* reúne información e imprime resultados */

   if (pId == 0)

   {  fprintf (stderr, "El area desde %f a %f es: %f\n",
               liminferior, limsuperior, total );
       fflush(stdout);
   }

   MPI_Finalize( );
   return 0;
}
```

Variables observadas: Las variables de interés en esta prueba son el tiempo de ejecución del programa y el número de usuarios simultáneos que soporta.

Ejecución de la prueba:

La prueba ser realiza incrementando el número de usuarios que ejecutan el programa, de manera simultánea, en el cluster por medio de la aplicación Web.

Los tiempos se miden en segundos y se cuentan a partir del momento en que los usuarios ejecutan la aplicación hasta que obtienen el resultado o se despliega el mensaje que indica que el cluster no puede ejecutar las peticiones.

Resultados: Los tiempos de ejecución se muestran en la tabla 5.10.

	Tiempos de ejecución					
	usuario 1	usuario 2	usuario 3	usuario 4	usuario 5	promedio
un usuario	21,47					21,47
dos usuarios	23,32	22,08				22,7
tres usuarios	22,24	21,63	32,33			25,4
cuatro usuarios	91,55	89,54	78,42	87,64		86,7875
cinco usuarios	120,94	120,94	115,02	126,83	122,24	121,194

Tabla 5.10. Tiempos de ejecución prueba 4.

Los tiempos de ejecución variando el número de usuarios se muestran en la figura 5.15.

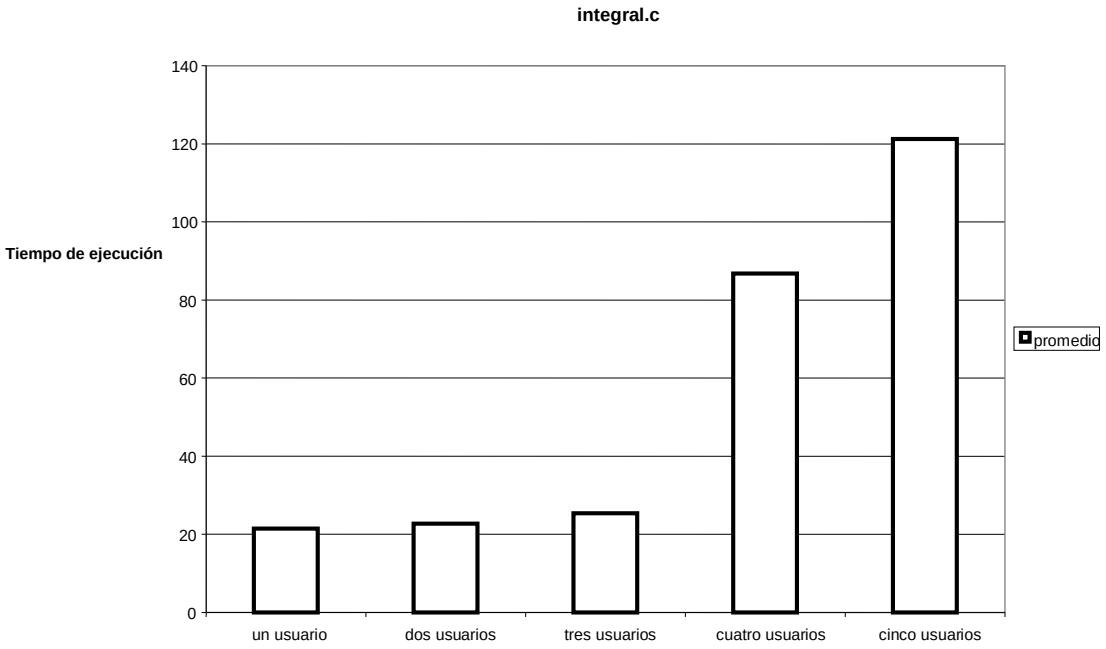


Figura 5.15. Tiempos de ejecución prueba 4.

Análisis de Resultados:

De la figura 5.15 se observa que el tiempo de ejecución aumenta considerablemente a partir de cuatro usuarios, lo cual indica que los recursos computacionales empiezan a alcanzar su límite operacional; más específicamente los procesadores de cada uno de los nodos disminuyen el rendimiento de su capacidad de cálculo

- **Prueba 5. P.5**

Objetivo: Determinar el comportamiento de los componentes hardware y medir la cantidad de usuarios soportados por el sistema total, la aplicación Web y el cluster de prueba que se está empleando, frente a un programa intensivo en uso de memoria.

Algoritmo: El programa solicita memoria para datos de entrada y de salida con la función *malloc()* del tamaño necesario para guardar enteros (int). Luego las posiciones de entrada se copian en las de salida y se miden al tiempo que toma esto. Si no es posible asignar memoria se muestra un mensaje de error. Al finalizar el programa se liberan los recursos con free(). No se utilizan funciones de comunicación MPI.
El código es el siguiente:

Mem.c

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int main( argc, argv )
int argc;
char **argv;
{
    double t1, t2;
    double tmin;
    int    size, *in_data, *out_data;
    int    j, nloop, k;
    int    iproc;

    MPI_Init( &argc, &argv );
    printf( "tamaño (bytes) Tiempo (sec)\tRate (MB/sec)\n" );
      fflush(stdout);
    for (size = 1; size < 1000000; size *= 2 ) {
      in_data = (int *)malloc( size * sizeof(int) );
      out_data = (int *)malloc( size * sizeof(int) );
      if (!in_data || !out_data) {
```

```
            fprintf( stderr, "No pudo localizar espacio para %d ints\n",
size );
            break;
        }
        tmin = 1000.0;
        nloop = 100000/size;
        if (nloop == 0) nloop = 1;
        for (k=0; k < 10; k++) {
            t1 = MPI_Wtime();
            for (j=0; j<nloop; j++)
              memcpy( out_data, in_data, size * sizeof(int) );
            t2 = (MPI_Wtime() - t1) / nloop;

            if (t2 < tmin) tmin = t2;
        }
        printf( "%d\t%f\t%f\n", size * sizeof(int), tmin,
              1.0e-6*size*sizeof(int)/tmin );
        fflush(stdout);
        free( in_data );
        free( out_data );
    }
    MPI_Finalize( );
    return 0;
}
```

Variables observadas: Las variables de interés en esta prueba son el tiempo de ejecución del programa y el número de usuarios simultáneos que son soportados.

Ejecución de la prueba:

La prueba se realiza incrementando el número de usuarios que ejecutan el programa, de manera simultánea, en el cluster por medio de la aplicación Web.
Los tiempos se miden en segundos y se cuentan a partir del momento en que los usuarios ejecutan la aplicación hasta que obtienen el resultado o se despliega el mensaje que indica que el cluster no puede ejecutar las peticiones.

Resultados:
Los tiempos de ejecución obtenidos se muestran en la tabla 5.11. Los resultados obtenidos variando el número de usuarios se muestran en la figura 5.16.

	Tiempos de ejecución					
	usuario 1	usuario 2	usuario 3	usuario 4	usuario 5	promedio
un usuario	13,99					13,99
dos usuarios	55,72	74,68				65,2
tres usuarios	37,52	52,97	46,65			45,7133333
cuatro usuarios	29,17	31,46	46,28	34,66		35,3925
cinco usuarios	191,74	191,74	183,65	203,56	174,2	188,978

Tabla 5.11. Tiempos de ejecución prueba 5.

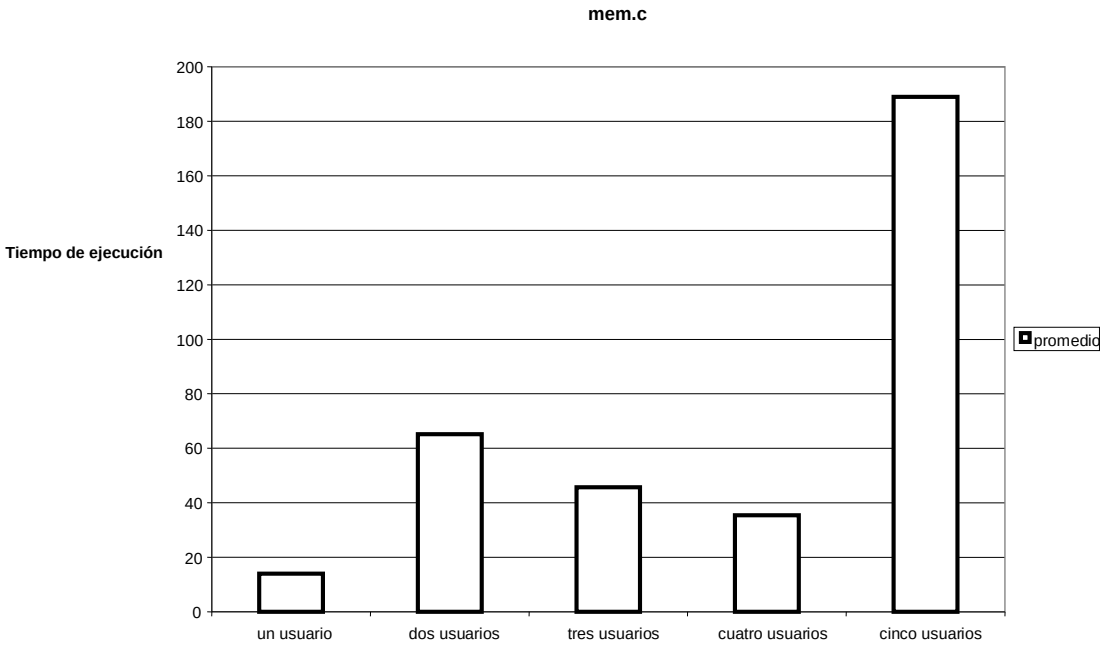


Figura 5.16. Tiempos de ejecución prueba 5.

Análisis de Resultados:

Los resultados varían según la cantidad de usuarios se observa una disminución del tiempo de ejecución cuando acceden cuatro usuarios respecto al tiempo empleado por dos. En cinco usuarios el sistema llega a su límite disparándose el tiempo de ejecución, esto se debe a las características que presenta la arquitectura hardware de cada uno de los nodos, es decir a la capacidad de la board y la memoria RAM de funcionar más eficientemente en conjunto.

- **Prueba 6. P.6**

Objetivo: Medir la cantidad de usuarios soportados y el comportamiento del sistema total que se está empleando, por medio de un programa que realiza una comunicación en anillo. Esta aplicación permite observar el efecto de la comunicación en el sistema; en este caso se pasa un valor *int* para ver la influencia de la red en el cluster.

Algoritmo: Este código pasa un valor de un proceso a otro hasta que llegue al último proceso como el juego “el teléfono roto”. Este programa utiliza las funciones MPI_Send y MPI_Recv que son funciones de envío y recepción bloqueantes, es decir, MPI_Recv se queda esperando el mensaje y no permite continuar con las otras líneas del código.

Ring.c

```
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value, size;
    MPI_Status status;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
      if (rank == 0) {
          value=100000;
          MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
      }
      else {
          MPI_Recv( &value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
                  &status );
          if (rank < size - 1)
            MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
      }
      printf( "Proceso %d recibe %d\n", rank, value );
    MPI_Finalize( );
    return 0;
}
```

Variables observadas: Las variables de interés en esta prueba son el tiempo de ejecución del programa y el número de usuarios simultáneos que soporta.

Ejecución de las pruebas:

La prueba ser realiza incrementando el número de usuarios que ejecutan el programa, de manera simultánea, en el cluster por medio de la aplicación Web.

Los tiempos se miden en segundos y se cuentan a partir del momento en que los usuarios ejecutan la aplicación hasta que obtienen el resultado o se despliega el mensaje que indica que el cluster no puede ejecutar las peticiones.

Resultados:

Los resultados obtenidos son los de la tabla 5.12.

	Tiempos de ejecución					
	usuario 1	usuario 2	usuario 3	usuario 4	usuario 5	promedio
un usuario	14,45					14,45
dos usuarios	37,82	54,04				45,93
tres usuarios	38,71	46,44	37,07			40,74
cuatro usuarios	137,21	137,21	129,13	133,88		134,3575
cinco usuarios	82,76	82,76	96,1	93,61	96,22	90,29

Tabla 5.12. Tiempos de ejecución prueba 6.

Los tiempos de ejecución variando el número de usuarios se muestran en la figura 5.17.

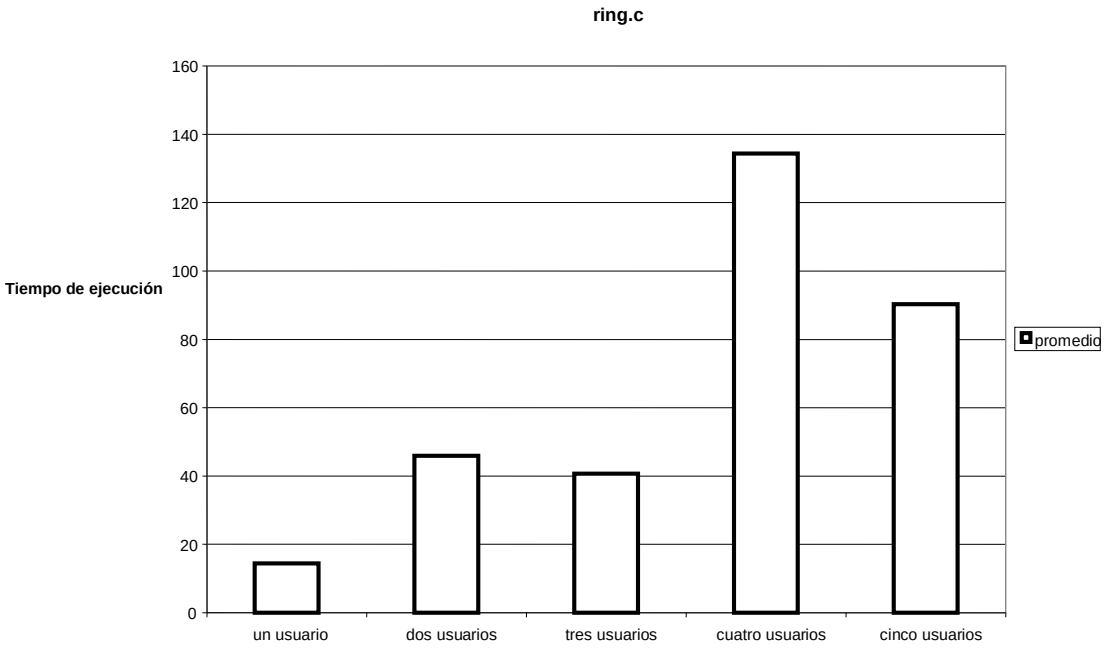


Figura 5.17. Tiempos de ejecución prueba 6.

Análisis de Resultados:

Se observa que el tiempo de ejecución aumenta cuanto el programa es ejecutado por cuatro usuarios. Con esto se infiere que la red es un factor que varia de forma tal que depende de los tamaños de los paquetes a transmitir y las comunicaciones entre los nodos, pero a pesar de esto se observa la misma tendencia general que en las pruebas anteriores.

Para concluir es importante agrupar todas las pruebas que tienen por objeto analizar todo el sistema y determinar la cantidad de usuarios que este puede soportar. Para esto se agrupan los resultados de todos los programas mediante la figura 5.18 en la que se muestran los tiempos de ejecución de cada código según el número de usuarios que lo ejecute.

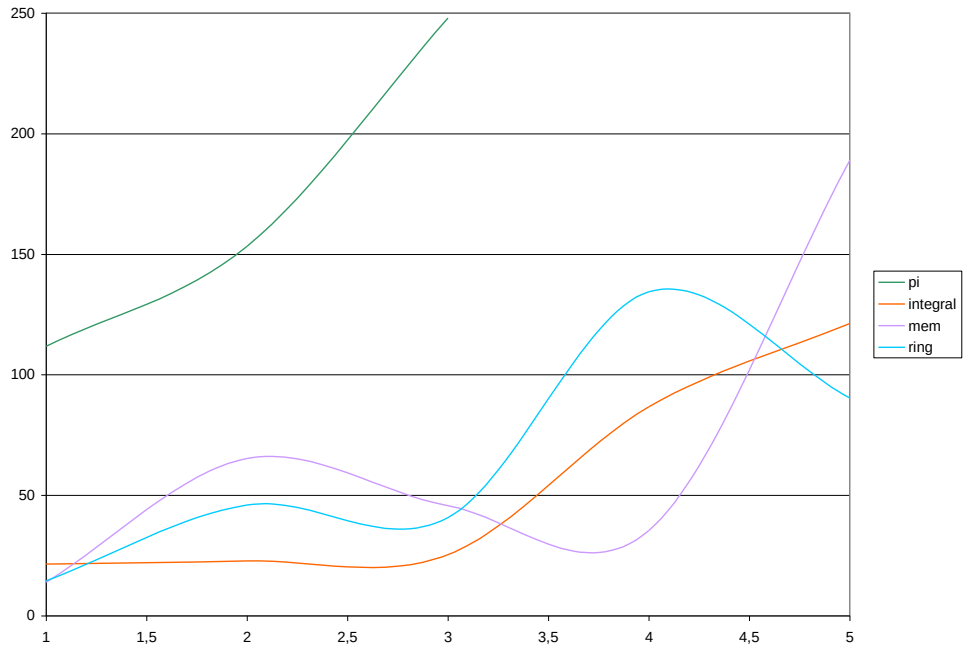


Figura 5.18. Tiempos de ejecución de todas las pruebas.

De la figura 5.18 se puede observar el comportamiento variable del sistema de acuerdo a la aplicación que se ejecute. El tiempo de ejecución sube muy rápidamente para el programa que calcula el número Pi, debido a que este hace uso de grandes cantidades de recursos en varios factores críticos del sistema paralelo, es por esto necesario evaluar

aspectos de manera individual para obtener una percepción general de cómo se comportará el sistema según las características que tenga el algoritmo que se va a ejecutar, permitiendo determinar el punto en el cual se pierde rendimiento y las consideraciones a tener en cuenta al desarrollar implementaciones paralelas que se ejecutarán sobre este tipo específico de soluciones.

Capitulo 6 CONCLUSIONES

En esta sección se resumen las deducciones obtenidas del desarrollo del proyecto; así como también se plantean recomendaciones y posibles trabajos futuros para quienes se interesen en el tema.

6.1. Conclusiones

- Es posible implementar sistemas cluster con computadores de características limitadas empleando software de procesamiento en paralelo. Si se realiza un programa de procesamiento en paralelo de forma adecuada, los procesos generados pueden ser ejecutados por todos los nodos logrando la disminución del tiempo de ejecución respecto al de un único computador.

- Los dispositivos de interconexión del cluster representan uno de los factores más importantes en el rendimiento del sistema, debido a que la comunicación entre los nodos aumenta la sobrecarga, las aplicaciones en la cuales todos los nodos se comunican entre si intercambiando un número considerable de información no son viables en un cluster.

- A pesar de que los cluster se diseñan para aplicaciones específicas, se puede crear un cluster de propósito general mediante el uso de librerías genéricas que no estén ligadas a un tipo de implementación en particular.

- La creación de una aplicación Web empleando tecnología Java Server Pages, que permita la ejecución de código paralelo desde un navegador convencional en un cluster remoto, evita el uso de métodos nativos exclusivos de una arquitectura hardware y de un lenguaje particular.

- Gracias a los desarrollos en software libre de los últimos años es posible que las personas puedan configurar sistemas paralelos; pero es necesario que estén familiarizadas con los entornos operativos libres.

- El modelo de acceso a un cluster planteado en este trabajo, permite que las personas no vinculadas directamente con un sistema paralelo de este tipo, puedan emplearlo y obtener información sin la necesidad de desplazarse al lugar físico de su ubicación, ni instalar aplicaciones adicionales en el computador desde el cual se accede.

6.2. Recomendaciones

- Al escoger el software para la implementación de un cluster es necesario sopesar la versión adecuada, esto es, la versión más reciente que se adapte al sistema disponible (sistema operativo, capacidad de almacenamiento, capacidad de memoria RAM, etc), con total compatibilidad y la mayor estabilidad. Para un cluster compuesto por nodos de 200Mhz se recomienda emplear RedHat 7.3 y Oscar 2.0 debido a que es el software más estable con este hardware.

- El primer paso que se debe dar en la creación de programas paralelos es una planeación cuidadosa recurriendo a paradigmas de programación diferentes a los

que tradicionalmente se usan en máquinas secuenciales, de dicha planeación depende la eficiencia de la solución del problema que se pretende resolver. Se deben realizar diagramas de dependencias para identificar qué se puede resolver en paralelo y seleccionar el modelo de comunicación y los algoritmos que se adecuen a dichos diagramas.

6.3. Trabajos Futuros

Se proponen como trabajos futuros:

- Realizar trabajos relacionados con sistemas tipo Grid que permitan tener una dimensión práctica de su aplicabilidad en nuestro entorno.

- Aumentar las características de la plataforma Web desarrollada en este trabajo que permitan la ejecución sobre múltiples cluster, sistemas MMP, y sistemas Grid.

BIBLIOGRAFÍA

[ANGUITA2005] Arquitectura de computadores, Julio Ortega, Mancia Anguita, Alberto Prieto; Thomsom editores España, 2005

[BCD2005] Portal Web: http://bccd.cs.uni.edu, 2005

[BEVILACQUA2004] Multiprocesadores (MIMD), Roberto J. G. Bevilacqua, http://www-2.dc.uba.ar/materias/so/datos/cap06.pdf, 2004

[BLADEENC2002] Portal Web: http://www.osl.iu.edu/~jsquyres/bladeenc/, 2002

[BUYTAERT2003] The openMosix HOWTO, Kris Buytaert y otros, http://www.faqs.org/docs/Linux-HOWTO/openMosix-HOWTO.html#AEN77, 2003

[C32006] Portal Web: http://www.csm.ornl.gov/torc/C3/, 2006

[CHIRINOV2003] Articulo Proyecto Cluster Openmosix (Linux), Roumen Chirinov, Noticias 3D, http//www.noticias3d.com, 2003

[CHOY2002] Building a Beowulf, Ron Choy, http://www-math.mit.edu/ ~edelman/18.337/notes02/032002.ppt, 2002

[CILK2006] Portal Web: http://supertech.csail.mit.edu/cilk/, 2006

[CLUSNFS2000] Portal Web: http://clusternfs.sourceforge.net/, 2000

[CONDOR2006] Portal Web: http://www.cs.wisc.edu/condor/, 2006

[CUMPLIDO2004] Computo Paralelo, René Cumplido, Claudia Feregrino; http://ccc.inaoep.mx/~cferegrino/cursos/comparal/ Ch1_Intro.pdf, 2004

[DEVANEY1995] The Parallel Applications Development Environment (PADE) User's Manual, Judith E. Devaney, Robert Lipman, Minwen Lo,William F. Mitchell, Charles W. Clark; http://math.nist.gov/mcsd/savg/pade/pade_man/pade_man.html, 1995

[DIETZ1998] Linux Parallel Processing HOWTO Clusters Of Linux Systems, Hank Dietz, http://aggregate.org/PPLINUX/19980105/ pphowto.html,1998

[FFTW2006] Portal Web: http://www.fftw.org, 2006

[FLYNN2006] Documento de Internet:
http://en.wikipedia.org/wiki/Flynn%27s_taxonomy, 2006

[FREDMAN1994] Diccionario de computación, Alan Freedman, McGraw-Hill,1994

[GANGLIA2003] Documento de Internet:
http://www.cecalc.ula.ve/HPCLC/slides/day_06/Monitoring/Exercises_Monitoring/Practica_de_GANGLIA_TOOLKIT.pdf, 2003

[GOVEA2002] ¿Qué es un clusters? Edgar Govea, http://www.consol.org.mx/2002/ponencias/conferencias/Edgar_Govea_-_Clusters.html. 2002

[GRAMA2003] Introduction to Parallel Computing, Second Edition. Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Addison Wesley 2003

[GRID2004] Portal Web:
http://www.grid.org/about/gc/, 2004

[GROPP2003] A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard; William Gropp, Ewing Lusk, Nathan Doss, Anthony Skjellum; http://www-unix.mcs.anl.gov/ mpi/mpich1/papers/ mpichimpl.pdf, 2003

[HDF52006] Portal Web:
http://hdf.ncsa.uiuc.edu/HDF5/, 2006

[HENNESY2002] Computer Architecture. A Quantitative Approach. 3rd Edition, John Hennesy, David A. Patterson; Morgan Kaufmann Inc, 2002

[IBMGPFS2006] Marketing Communications Systems Group, http://www-03.ibm.com/servers/eserver/clusters/software/gpfs.pdf, IBM Corporation, 2006.

[INTRODUC1999] Documento de Internet:
Introducción a los Multiprocesadores, http://panou.act.uji.es/ AIC/tema4.1.ppt, 1999

[KANNOPIX2004] Portal Web:
http://bofh.be/clusterknoppix, 2004

[KVM2006] Documento de Internet:
http://en.wikipedia.org/wiki/KVM_Switch, 2006.

[LUSTRE2006] Portal Web:
http://www.lustre.org/, 2006

[MAUI2006] Portal Web:
http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php, 2006

[MODELOS2005] Documento de Internet:
Modelización analítica de los Programas Paralelos,
http://weblidi.info.unlp.edu.ar/catedras/paralela/Teorias/Clase9-2005.ppt, 2005

[MORGADO2003] Apuntes de arquitecturas avanzadas de computadores, Arturo Morgado, http://www.uca.es/area_conocimiento/arquitec_comp/aac/apuntes/Tema2/apuntes_AAC_Tema2.pdf Estévez, 2003

[MPI2006] Portal Web:
http://www.mcs.anl.gov/mpi/, 2006

[NARAYAN2005] Introducing the basic concepts of High Performance Computing with Linux cluster technology, Aditya Narayan, http://www-128.ibm.com/ developerworks/linux/library/l-cluster1/#main, 2005

[NAS2006] Documento de Internet: http://en.wikipedia.org/wiki/Network-attached_storage, 2006

[NFS2006] Documento de Internet: http://en.wikipedia.org/wiki/Network_File_System, 2006

[NIETO2002] Cluster Heterogéneo de Computadoras, Emilio José Laza Nieto, 2002

[NPACI2006] Portal Web: http://rocks.npaci.edu/, 2006

[OPENGFS2004] Portal Web: http://opengfs.sourceforge.net/, 2004

[OPENM2004] Documento de Internet: El manual para el clustering con openMoxis, http://alumnes.eps.udl.es/~b4767512/07.openMosix/manual_html/manual.html, 2004

[OPENMP2004] Portal Web: http://www.openmp.org/drupal/, 2004

[OPENPBS2003] Portal Web: http://www.openpbs.org, 2003

[OSCAR2005] Portal Web: http://oscar.openclustergroup.org/, 2005

[PBS2006] Portal Web: http://www.pbspro.com, 2006

[PCL2003] PCL - The Performance Counter Library, Rudolf Berrendorf, Bernd Mohr; http://www.fz-juelich.de/zam/PCL/doc/pcl/ pcl.html, 2003

[PENGUIN2006] Portal Web: http://www.penguincomputing.com, 2006.

[PGDB2005] Portal Web: http://www.pgroup.com/products/pgdbg.htm, 2005

[PLAPACK2000] Documento de Internet: http://www.cs.utexas.edu/users/plapack/, 2000

[PLATA2003] Clusters Arquitecturas Distribuidas, Oscar Plata, Universidad de Málaga, http://www.ac.uma.es/educacion/cursos/informatica/ ArqDist/pdfs/04-ClustersBW.pdf Junio de 2003

[PROCESO2006] Documento de Internet: http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29, 2006

[PVFS2006] Portal Web: http://www.pvfs.org/, 2006

[RACK2006] Documento de Internet: http://en.wikipedia.org/wiki/19-inch_rack, 2006

[ROCKS2006] Portal Web:

www.rocksclusters.org, 2006

[SHENDE2005] The TAU Parallel Performance System, Sameer S. Shende y Allen D. Malony; www.cs.uoregon.edu/research/paracomp/ papers/ijhpca05.tau/ TAU_ACTS_SS.pdf, 2005

[SLOAN2004] High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI; Joseph D. Sloan, O´Relley editors, 2004

[SPECTOR2000] Building Linux Cluster, David HM Spector, O´Relley editors, 2000

[SPRNG2003] Portal Web: http://sprng.cs.fsu.edu/, 2003

[TARRICONE2004] Grid Computing for Electromagnetics, Luciano Tarricone, Alessandra Esposito, Artech House Inc, 2004

[TOTALVIEW2005] Documento de Internet: http://www.absoft.com/corporate/pressreleases/050427_01.html, 2005

[VRENIOS2002] Linux Cluster Architecture. Alex Vrenios, Sams,2002

[WIKI2005] Enciclopedia en línea: http://www.wikipedia.org, 2005

[WIKI2006] Documento de Internet: http://en.wikipedia.org/wiki/Grid_computing, 2006

[WIKIIDE2006] Documento de Internet: http://es.wikipedia.org/wiki/Integrated_Drive_Electronics, 2006

[XPVM1994] Documento de Internet: http://www.netlib.org/utk/icl/xpvm/xpvm.html, 1994

[YAÑEZ2004] Introducción a las tecnologías de clustering en GNU/Linux, Rosa María Yáñez Gómez, http://talika.eii.us.es/~rosa/ clustering.pdf, 2004