Universidad Austral de Chile

Facultad de Ciencias de la Ingeniería

Escuela de Ingeniería Civil en Informática

Propuesta y evaluación de un modelo de re-configuración dinámica en un subsistema de entrada/salida redundante para un sistema de archivos distribuido y paralelo.

Tesis para opstar al título de

Ingeniero Civil en Informática

PROFESOR PATROCINANTE:

DR. RAIMUNDO EXEQUIEL VEGA VEGA

JUAN PABLO GARCÍA OJEDA

VALDIVIA – CHILE

2003

Universidad Austral de Chile
Instituto de Informática

Valdivia, 10 de Julio 2003

Sra.
Miguelina Vega R.
Directora Escuela Ing. Civil en Informática
Presente

Estimada Sra. Directora:

Informo a Ud. que he revisado el trabajo de Tesis "PROPUESTA Y EVALUACIÓN DE UN MODELO DE RE-CONFIGURACIÓN DINÁMICA EN UN SUBSISTEMA DE ENTRADA/SALIDA REDUNDANTE PARA UN SISTEMA DE ARCHIVOS DISTRIBUIDO Y PARALELO", realizado por el alumno SR. JUAN PABLO GARCÍA OJEDA.

Como consecuencia de lo anterior he podido comprobar que la solución propuesta cumple con los objetivos planteados, y muestra una alta coherencia y rigurosidad lógica. La precisión del lenguaje técnico en la exposición, composición, redacción e ilustración es muy adecuada.

Por tal motivo, he resuelto calificar con nota 7,0 (Siete) el trabajo del Sr. García.

Sin otro particular, le saluda atentamente a Ud.,

DR. RAIMUNDO VEGA V.

Dirección: General Lagos 2086, Campus Miraflores, Valdivia
Fono: (63)- 221427 – Fono-Fax : (63)- 293115

Universidad Austral de Chile

Instituto de Informática

Valdivia, 14 de julio de 2003
Comunicación Interna 090/03

De : Luis Hernan Vidal Vidal.
A : Sra. Miguelina Vega R.
Directora de Escuela de Ingeniería Civil en Informática
Ref. : Informa Calificación Trabajo de Titulación.

MOTIVO: Informar revisión y calificación del Proyecto de Título "Propuesta y evaluación de un modelo de re-configuración dinámica en un subsistema de entrada/salida redundante para un sistema de archivo distribuido y paralelo.", presentado por el alumno Juan Pablo Garcia Ojeda, que refleja lo siguiente:

Se logro el objetivo planteado de entregar un diseño de modelo de subsistema de re-configuracion dinámica para un sistema de entrada/salida redundante asi como la posterior evaluación del comportamiento de este en un sistema de archivos distribuido y paralelo.

La revisión hecha sobre las bases y tendencias actuales que subyacen a los distintos componentes insertos en la problemática, se presentan como una excelente referencia para el desarrollo de trabajos afines.

El trabajo constituye un aporte significativo a la investigación en el área de Ingeniería en Informática.

Por todo lo anterior expuesto califico el trabajo de titulación del señor Juan Pablo Garcia Ojeda con nota 7,0 (siete como cero).

Sin otro particular, se despide atentamente.

UNIVERSIDAD AUSTRAL DE CHILE
INSTITUTO DE INFORMÁTICA
FACULTAD DE CIENCIAS DE LA INGENIERÍA

Prof. Ing. Luis Hernan Vidal Vidal.
Instituto de Informática.
Universidad Austral de Chile.

General Lagos 2086 · Campus Miraflores · Valdivia · Chile
Casilla 567 · Fono: 56 63 221427 · Fax: 56 63 293115 · instituto@inf.uach.cl · www.inf.uach.cl

Universidad Austral de Chile

Instituto de Informática

Valdivia, 15 de Julio 2003
Comunicación Interna Nº 14/2003

DE : Eliana Scheihing, Profesora del Instituto de Informática

A : DIRECTORA ESCUELA INGENIERIA CIVIL EN INFORMATICA

MOTIVO:

INFORME TRABAJO DE TITULACION

Nombre Trabajo de Titulación: **"Propuesta y evaluación de un modelo de reconfiguración dinámica en un subsistema de entrada/salida redundante para un sistema de archivos distribuído y paralelo".**

Nombres del Alumno: **Juan Pablo García Ojeda**

.....

Nota : .....**7,0**..... ..... **siete coma cero** .....
(en números) (en letras)

FUNDAMENTO DE LA NOTA:

Trabajo muy interesante y excelente presentación

Atentamente,

Eliana Scheihing

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

Agradecimientos

Este trabajo está dedicado a mi Papi Juan, mi Mami Gudelia, y mis hermanos Miki y Nachito que tanta ayuda, inspiración y apoyo me brindaron durante todos estos años de estudio. Como no agradecer también a mi abuelita Gudelia, mis tios, tias, primos y primas que siempre tenían consejos y palabras de aliento, así también como a tantas amigas y amigos que aparecieron en mi vida en el momento preciso y dejaron un lindo recuerdo (Humberto, Andrés, Andrea, Karyn, Hernán, Rodrigo, Max, Michi, Enrich, Marcelo, Mónica, Levy, Erika, Roxy, Patita, José, Judith, Anita, Sofia).

Resumen

Desde los inicios de la informática ha existido la necesidad de almacenar los datos de tal forma que estos se encuentren siempre disponibles y su búsqueda sea simple y rápida. Combinar estas tres características de una manera eficiente presentan un gran desafío a las ciencias de la computación, y es por esto que han surgido distintas líneas de investigación que analizan este problema bajo distintos puntos de vista.

El presente trabajo toma una de estas líneas y estudia el problema centrándose en la disponibilidad de los datos. Para esto, se toma como base un simulador de un sistema de archivos distribuido y paralelo con tolerancia a fallos al cual se le añadirá una nueva funcionalidad. Esta nueva funcionalidad es la reconfiguración dinámica, es decir, la característica que permitirá al sistema de archivos distribuido y paralelo poder agregar mas nodos de almacenamiento sin necesidad de detener la normal entrega de servicios.

Finalmente y posterior a la implementación, se generan pruebas que permiten analizar los resultados y compararlos con otros estudios realizados anteriormente sobre el mismo sistema bajo condiciones que no incluyen reconfiguración.

Summary

Since informatics beginning has been existed the need of storage data in a way that this can be always available and his search be simple and fast. To combine this three characteristics in an efficient way presents a big challenge for computer sciences, and for this has arise distinctive investigation lines which analyze this problem under differents points of view.

The present work take one of this lines and study the problem centering in data availability. For this proposite, it take as base a fault tolerance distributed and parallel system simulator where a new functionality will be added. This new functionality is the dynamic reconfiguration, that means the characteristic which will let the parallel and distributed system be able to add new storage nodes without the need of stop the normal service delivery.

Finally and after the implementation, its generate proofs which let analyze the results and compare them with other studies developed previously under conditions where reconfiguration is not included.

Capítulo I

1. Introducción

La informática se define como: “El conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de ordenadores” [7]. Con el pasar de los años, esta ciencia ha aprendido que el valor de la información que obtiene depende en gran manera de la cantidad y calidad de los datos que la subyacen. Existen ocasiones en que la cantidad de datos es tan abrumadora que el acceso a estos no se puede obtener en un lapso razonable de tiempo por lo que la información que con ella se obtiene se considera obsoleta.

Es por lo anterior que los computadores encargados de manejar los datos deben tratar de optimizar el acceso a estos, para ello se ha utilizado una tecnología llamada sistema de archivos. Esta tecnología almacena los datos relacionados entre si en un conjunto de direcciones de memoria que forman lo que se conoce como archivo, lográndose de esta manera obtener agrupaciones ordenadas de datos en distintos archivos. La forma en que se organizan los archivos dentro de la memoria del computador es la característica principal que diferencia a cada sistema de archivos. Los primeros sistemas de archivos, a los que se puede denominar sistemas de archivos tradicionales, se ejecutaban en entornos monoprocesador y ofrecían sus servicios únicamente a los usuarios de dichos sistemas. Luego aparecieron las redes de interconexión de computadores que trajeron consigo nuevas oportunidades, como la posibilidad de compartir los archivos de un computador a otro [1,2], de aquí salió una nueva generación de sistemas de archivos, conocidos como Sistemas de Archivos Distribuidos, que tenía como objetivo principal abstraer al usuario de la ubicación física de los recursos a los que se deseaba tener acceso. Con el transcurrir del tiempo, la velocidad de procesamiento superó considerablemente a la velocidad de entrada y salida a los datos proporcionada por los

sistemas de archivos, lo que trajo consigo la llamada “Crisis de la E/S” [3], la cual se agrava aún mas en entornos paralelos donde el acceso concurrente a los archivos, es decir, dos o más procesos acceden a este, es muy frecuente [4,5] y tanto los sistema de archivos tradicionales como los distribuidos no están optimizados para este tipo de acceso [6]. De esta problemática aparecieron los Sistemas de Archivos Distribuidos y Paralelos, los cuales combinan soluciones tales como: paralelismo en el sistema de entrada y salida, interfaces paralelas y cache en el sistema de entrada y salida.

Estos últimos sistemas han sido hasta el momento la solución a la crisis de la E/S. Sin embargo aún queda por incorporar un tema que tiene que ver con la necesidad que tienen algunos sistemas de información de acceder en forma constante y continua a los recursos. Lo anterior es conocido como la disponibilidad de datos, cuestión que se torna crítica en los sistemas de archivos distribuidos y paralelos ya que, a medida que se incrementa el paralelismo y la cantidad de nodos de entrada/salida, también aumenta la probabilidad de fallo del sistema [8]. En vista del poco estudio que había en este campo, es que en [8] se propuso un modelo de redundancia de datos que además fue implementado en un simulador especialmente diseñado para el efecto [8].

A partir del estudio que ahí se realizó, esta tesis estudiará un método que permitirá agregar una nueva propiedad al sistema entonces propuesto. Esto es la capacidad de poder incorporar en forma dinámica nuevos dispositivos de redundancia (nodos de entrada/salida) sin detener la normal entrega de servicios del sistema de entrada/salida. La justificación a esto la encontramos en que es de imperiosa necesidad este tipo de características en los sistemas de redundancia de datos, ya que la escalabilidad total del sistema depende de la capacidad que se tiene de incorporar nuevos dispositivos sin atentar contra la propia naturaleza de la redundancia de datos que es tener los datos a disposición de quienes lo soliciten en cualquier tiempo. De esta forma se pretende atacar

este problema permitiendo el ingreso de nuevos dispositivos de entrada/salida para luego estudiar el efecto que esto provoca sobre la totalidad del sistema.

2. Antecedentes Generales

En la actualidad existen estudios acerca del tema realizados en distintas Universidades alrededor del mundo así como en departamentos de investigación de empresas líderes del sector informático. Entre las primeras se encuentran la universidad de Dartmouth en Estados Unidos que lleva proyectos en las líneas que investigan los tipos de cargas de datos asociados a los sistemas de archivos, y el estudio de un marco de trabajo que permita implementar mallas de computadores que hoy en día son utilizadas para el análisis de grandes cantidades de datos, como es en el caso de la bioinformática que analiza las gigantescas bases de datos generadas en torno a una proteína.

Por otro lado existen otras instituciones ocupadas del diseño, implementación y prueba de simuladores de sistemas de archivos distribuidos y paralelos que permiten estudiar el comportamiento de estos previo a su implementación en el mundo real. Este es el caso de la Universidad de Dartmouth que creó Starfish, la cual es una herramienta de simulación para la investigación avanzada de sistemas de archivos.

La Universidad Politécnica de Madrid en España creó ParFiSys, el cual es un sistema de archivos paralelos que a diferencia de muchos otros incluye el estudio de las caché para la mejora de los tiempos de lectura y escritura de los datos. Dentro de este último sistema, se enmarca el estudio realizado en [8] que aporta conocimientos en una línea poco estudiada como es el la tolerancia de fallos en sistemas de archivos distribuidos y que proporciona una base a esta tesis.

Otra línea de investigación, es la que tiene que ver con el estudio del diseño de un estándar para las interfaces de comunicación en un sistema de archivos distribuido. Este

estudio es liderado principalmente por organizaciones gubernamentales como es la Nasa en Estados Unidos y por empresas líderes en Informática como es el caso de IBM.

3. Motivación

La posibilidad de colaborar en la investigación de áreas menos estudiadas de la informática, como es la tolerancia a fallos en sistemas paralelos, y aportar en el desarrollo de un sistema, cuya finalidad es la entrega de mejores servicios que lleve a las personas a tener un acceso más rápido y efectivo a la información, junto al hecho de poder adquirir un conocimiento más profundo en el área de los sistemas distribuidos y paralelos que apoyan líneas de investigación tan fascinantes como aquellas que investigan y analizan gigantes bases de datos al estudiar una proteína o una célula y son capaces de descubrir la cura a enfermedades que hasta hace pocos años no se podía concebir debido a la enorme cantidad de cálculos asociados conforman los principales agentes motivadores de esta investigación.

4. Objetivos

4.1 Objetivos Generales

El objetivo general de esta tesis es el diseño de un modelo de subsistema de reconfiguración dinámica para un sistema de entrada/salida redundante así como la posterior evaluación del comportamiento de este en un sistema de archivos distribuido y paralelo.

4.2 Objetivos Específicos

La investigación de las áreas críticas en el funcionamiento del subsistema de entrada/salida propuesto en [8].

La propuesta de un modelo de subsistema de reconfiguración dinámica.

El desarrollo del modelo propuesto dentro del subsistema generado en Vega99.

La evaluación en forma estadística del rendimiento del subsistema propuesto.

La sugerencia de otros enfoques al problema, para futuras investigaciones.

5. Organización de la Tesis

Esta tesis esta organizada de manera tal que los contenidos se entregan de manera progresiva en cuanto a especialización y tiempo de desarrollo. Para empezar, el primer capitulo entrega un resumen y una introducción al tema así también como las motivaciones y objetivos planteados. Luego se describen los materiales y métodos utilizados para el estudio y análisis del problema.

En el segundo capítulo, se trata el problema de la reconfiguración junto con las bases y tendencias actuales que subyacen a los distintos componentes insertos en la problemática. Es decir se presenta la simulación de sistemas, las tendencias de los sistemas de almacenamiento actuales, los sistemas de archivos redundantes y los sistemas de archivos distribuidos.

El tercer capítulo revisa con mayor profundidad el tema central de este trabajo y presenta en teoría las ideas que se proponen y desarrollan para resolver la problemática asociada a la reconfiguración dinámica.

El cuarto capítulo presenta la evaluación de las ideas descritas en el capítulo anterior y los criterios utilizados para la selección definitiva de los distintos elementos que componen el sistema, es decir, los tamaños de las unidades de reparto, los tipos de carga de trabajo y las configuraciones de los RAID.

Finalmente en el quinto capítulo se muestran las conclusiones generadas en base a la comparación de estudios anteriores sobre el mismo tema y la bibliografía utilizada.

6. Materiales y Métodos

Para la realización de esta tesis, se contó con una ambiente cliente/servidor, en el cual se utilizó en primer lugar con un servidor que cuenta con las siguientes características:

Marca	: Dell
Modelo	: Poweredge 1300
Procesador	: Pentium III
Memoria RAM	: 128MB
Disco Duro	: 8GB
Sistema Operativo	: Linux Red Hat 6.2

En este servidor, se almacenan los cofigos fuentes que conforman el Sistema de Archivos Distribuido y Paralelo con Redundancia de Datos, desarrollado en [8] en el lenguaje C y que cuenta con los siguientes archivos:

Estos archivos se pueden clasificar en:

68 librerías que implementan los distintos componentes de un sistema de archivos

1 archivo principal

1 archivo readme

1 archivo todo

1 archivo makefile

4 archivos de testeo.

Por otro lado, se utilizó una estación de trabajo con las siguientes características:

Marca : DTK

Modelo	: DTK
Procesador	: Pentium II de 200MHZ Dual
Memoria RAM	: 128MB
Disco Duro	: 4GB
Sistema Operativo	: Windows 2000

Esta estación de trabajo se desempeñaba como el cliente que mediante el protocolo FTP enviaba al servidor los archivos que se iban modificando y permitía además obtener los resultados de las pruebas. Después de las modificaciones, se utilizaba el protocolo TELNET para enviar los comandos necesarios que permitían compilar, depurar y ejecutar el conjunto de archivos que componen el sistema.

Capítulo II

7. Descripción del Problema

Un sistema de archivos distribuido y paralelo es aquel en el cual los datos se encuentran repartidos en múltiples nodos dentro de una red de computadores y estos son accedidos de manera simultánea para mejorar el rendimiento. Para estudiar el desempeño de estos sistemas se ha construido un sistema simulador de este tipo de arquitecturas que además permite medir el ancho de banda y los tiempos de lectura y escritura para cargas de tipo científica y transaccionales (OLPT).

Este sistema es configurable mediante el ingreso de distintos parámetros que permiten identificar el tipo de distribución que tienen los distintos elementos de este, tales como: el número de nodos en la red, tamaño de las unidades de reparto, nivel de RAID, etc.

Esta tesis se centra en la formulación de un modelo que incorpore una nueva funcionalidad al sistema y este pueda así incorporar nuevos nodos a la red de manera que la información se redistribuya de forma balanceada. Para cumplir con esto, se describirán a partir del próximo capítulo los distinto elementos que constituyen tanto el problema como la solución.

8. Simulación de Sistemas

Un programa de simulación puede ser definido como una representación, o modelo, de algún sistema que esta bajo investigación. Un sistema puede ser definido, en términos generales, como una colección de elementos interdependientes, o entidades, que operan juntas para lograr alguna meta específica. Las entidades dinámicas en un modelo son entidades que se emplean en actividades. Las entidades dinámicas pueden cambiar el estado del sistema el cual esta representado por variables de estado.

Un modelo estocástico es uno en el cual uno o mas variables de estado toman valores de acuerdo a una distribución de probabilidad. Estas variables estocásticas son generalmente introducidas para definir algún tipo de incerteza en el modelo.

El estado del sistema cambia en el tiempo. Un sistema de simulación continuo es aquel en el cual una o mas variables de estado cambian de manera continua con el tiempo. Un sistema de simulación discreto es aquel en que las variables de estado cambian solo es ciertos puntos en el tiempo. Un evento es el cambio instantáneo en el valor de una variable de estado. De manera más general, el término evento es utilizado para describir una ocurrencia en el modelo que tiene asociado un tiempo. Esta tesis se basa en un sistema de simulación discreta, el cual puede ser modelado empleando procesos que representen a las entidades dinámicas concernientes. Los modelos de simulación continua generalmente requieren de un manejo matemático que haría demasiado extensivo el enfoque del problema debido a que se producen innumerables fenómenos del tipo fork-join. La ventaja de los modelos orientados a procesos es que establecen una relación directa entre el comportamiento del programa y el comportamiento del modelo que representan. Esta relación hace que el programa sea mas fácil de entender y mucho mas fácil de construir y modificar. Este último factor es

particularmente importante por cuanto los programas de simulación son frecuentemente modificados cuando se experimenta con variaciones en el modelo.

De forma operacional, la simulación discreta involucra la administración de colecciones de eventos que son identificados a medida que el programa de simulación es ejecutado. Cada evento tiene un tiempo asociado. Dentro del modelo, el tiempo puede ser incrementado en intervalos de largo fijo, esto es conocido como el método de avance en unidades, o bien de manera mas común avanzar desde un evento al siguiente, lo cual es conocido como el método manejado por eventos o método de eventos discretos.

Simulación del tiempo

Las entidades dinámicas en un modelo de simulación están envueltas en actividades. Algunas actividades toman un tiempo determinado en completarse mientras la duración de otras dependerá del comportamiento de varias entidades dinámicas en el modelo.

En términos de programación, una entidad dinámica puede ser representada por un proceso. El empleo de una entidad dinámica en una actividad de duración conocida puede ser implementada retardando la ejecución del proceso en que se encuentra y utilizando un módulo de reloj para el tiempo ocupado en completar la actividad que esta siendo modelada. Para actividades cuya duración no es conocida, el proceso que modela la entidad es suspendido por un mecanismo separado del módulo de reloj. Lo anterior, implica que un módulo de reloj debe proporcionar al menos dos operaciones:

a) Suministrar el valor actual del seudotiempo

b) Suspender un proceso de entidad empleado en alguna actividad específica hasta que el seudotiempo alcance el tiempo de completación de la actividad.

Modelos Estocásticos

Muchos modelos de simulación son estocásticos, esto significa que el comportamiento de las entidades en el sistema no es conocido de manera exacta, o más específicamente, que las salidas de ciertas operaciones y los tiempos de ciertos eventos no son conocidos de forma precisa. Esta incerteza es modelada por una o mas variables de estado que toman valores de acuerdo a una distribución de probabilidad esto es, toman valores que tienen una probabilidad de ocurrencia asociada. El rango de posibles valores y sus probabilidades estimadas está generalmente dado en la forma de una distribución de probabilidad. Seleccionar los valores de cierta distribución de probabilidad tiende a ser uno de los aspectos que mayor tiempo consume dentro de un programa de simulación. Para reducir este consumo de tiempo es una práctica común ajustar la distribución de probabilidad a una función estándar de probabilidad de tal manera que sea mas fácil seleccionar los valores.

Reloj

La unidad de tiempo definida para un modelo de simulación varia de una aplicación a otra. En un caso pueden ser microsegundos y en otros días o años. La unidad de tiempo no se representa de manera explícita pero es un requerimiento básico que todas las partes del modelo asuman la misma unidad de esta forma todas las entidades dinámicas del modelo utilicen el mismo reloj como medio común de sincronización. El requerimiento general para que un reloj simulado avance un intervalo de tiempo es que todos los procesos en el programa estén suspendidos.

9. Tendencias en los Subsistemas de Almacenamiento

Existen varias tendencias en el mundo de la computación que están dirigiendo el diseño de subsistemas de almacenamiento hacia niveles de paralelismo mas altos. Esto significa que los sistemas actuales y futuros adquirirán un mejor rendimiento de entrada salida incrementando el número de discos en vez de aumentar el rendimiento de cada uno de los discos de forma individual. [Patterson88].

Tasas de Rendimiento

En primer lugar, los procesadores están incrementando su rendimiento a una tasa mucho mayor que los discos. Los microprocesadores están aumentando su rendimiento a una tasa entre 30% y 50% por año.

Las unidades de disco, en cambio, han estado incrementado su rendimiento a una tasa mucho menor. Comparando el estado del arte entre 1981 y 1993 muestra que el tiempo de búsqueda promedio para una unidad de disco mejoró desde 16 ms a 10 ms y el 2002 llega a 8,7 ms , la latencia rotacional disminuyó desde 8.5 ms a 5 ms y el 2002 llegó a 4.17 ms. Combinando estos datos, el tiempo tomado para ejecutar un acceso promedio de 8KB mejoró de 27,1 ms a 15 ms, o lo que es cerca del 45% en un periodo de 20 años y hasta el 2002 el aumento tampoco ha sido considerable. Esto corresponde a una tasa de mejora de menos del 5% al año.

Un incremento en el rendimiento del procesador lleva directamente a un incremento en la demanda de ancho de banda de entrada/salida [12]. Debido a que la tecnología de los discos no avanza de la misma manera que la tecnología de los procesadores, es necesario utilizar paralelismo en el subsistema de almacenamiento

para cumplir con el incremento en la demanda de la entrada y salida. Esta ha sido, y continua siendo, la motivación principal de la tecnología conocida como matrices de discos.

Dimensiones Físicas

En los primeros años de los 80's, la tecnología de almacenamiento era llevada por unidades de un diámetro de 14 pulgadas [13] utilizados por mainframes de gran tamaño y escala, aplicados a bancos, compañías de seguros y líneas aéreas. Estas eran las únicas unidades que ofrecían la suficiente capacidad para cumplir con los requerimientos de estas aplicaciones [14]. Esto cambio dramáticamente con el crecimiento del mercado de los computadores personales. La enorme demanda de discos relativamente baratos produjo una tendencia descendente en el tamaño de los discos. Esta tendencia se vio posibilitada principalmente por el rápido crecimiento en la densidad de almacenamiento adquirida durante este periodo, lo cual permitió que la capacidad incrementara desde unas pocas decenas de Megabytes hasta las decenas de Gigabytes de hoy en día. Además se vio facilitado por el rápido crecimiento en los niveles de integración VLSI, los cuales, permitieron que los controladores electrónicos fuesen implementados en paquetes de menor tamaño. Las ventajas de estos nuevos discos eran:

- Platos de disco de menor tamaño y brazos de los cabezales mas livianos, lo que lleva a operaciones de búsqueda mas rápidas.

- Menor masa en cada plato de los discos lo que lleva a una rotación mas rápida

- Platos mas lisos que permiten a los cabezales volar a una menor altura lo que mejora la densidad de almacenamiento.

- Un menor consumo de energía reduce los problemas de ruido.

Estas ventajas, junto con los agresivos esfuerzos de desarrollo han causado la desaparición de los discos de gran tamaño. En 1994 la mejor tasa precio/rendimiento era adquirida utilizando discos de 3½" y los discos de 14" habían todos desaparecidos. Los discos de 2½" son comunes en laptops [15], y hoy en día hay disponibles discos de 1". Estos pequeños discos permiten la creación de matrices de gran escala y de alto rendimiento.

Para resumir, las inherentes ventajas de los discos pequeños, junto con la habilidad de proporcionar un muy alto rendimiento de entrada/salida por medio de la tecnología de matrices de discos, lleva a la conclusión de que los subsistemas de almacenamiento son y continuarán, construyéndose con un gran número de discos pequeños, en vez de en pequeños números de discos poderosos.

El advenimiento de nuevas aplicaciones con alta demanda de entrada/salida.

El incremento en la capacidad de almacenamiento y la proporcional reducción de los costos por megabyte alientan el desarrollo de nuevas tecnologías que demanden niveles aún mayores de rendimiento de entrada/salida. Los mas visibles ejemplos los encontramos en las aplicaciones de audio y video digital tales como el video sobre demanda, también en las emergentes bases de datos científicas con servidores que proporcionan acceso a gigantescas bases de datos. Estas aplicaciones tienen todas en común que si fuesen implementadas a gran escala, la demanda de almacenamiento y

ancho de banda de entrada/salida excedería por mucho las capacidades de los actuales subsistemas de almacenamiento. Estas aplicaciones llevarán a las tecnologías de almacenamiento a buscar mayores niveles de paralelismo para poder satisfacer las siempre presentes necesidades de ancho de banda.

La discusión anterior demuestra la necesidad de incremento en los niveles de paralelismo necesarios para poder cumplir así con las demandas de almacenamiento de los sistemas actuales y futuros. La discusión ha evitado de manera deliberada identificar el tipo específico de organización que ha ser utilizado en los subsistemas de almacenamiento, puesto que este tema se verá más adelante, pero si ha hablado acerca del relativamente alto número de discos del que debiese estar compuesto.

Sin embargo, la construcción de un subsistema de almacenamiento constituido por un gran número de discos tiene un significativo revés: la confiabilidad de tales sistemas empeora a medida que se incrementa el número de discos.

10. Sistemas de Archivos Redundantes

Arreglos de discos redundantes

Un método para asegurar un almacenamiento estable es a través de los Arreglos de Discos. Los arreglos de discos han sido propuestos como un medio efectivo de simular discos de alta capacidad. En estos, los datos son esparcidos sobre múltiples discos utilizando una tecnología conocida como la Intercalación de Bits, esto es, diferentes bits de una misma palabra de datos son almacenados en discos distintos. La intercalación de bits proporciona un alto rendimiento de entrada/salida, debido principalmente a que la lectura de diferentes partes de los datos puede ser ejecutada en paralelo. Sin embargo, un arreglo de discos utilizado de esta manera no es confiable, puesto que la falla de uno de los discos puede causar que la totalidad de los datos queden indisponibles.

Si se asume que un RAID consiste de solo un grupo de discos con N discos de datos y C discos de chequeo, es decir existen N+C discos en el grupo. Si los tiempos de falla y reparación de cada disco están distribuidos de forma exponencial, el tiempo medio de falla de cada disco es F y el tiempo medio de reparación es R, entonces el tiempo medio de falla del grupo o RAID esta dado por:

$$TMFG = \frac{F}{N+C} * \frac{\frac{F}{N+C-1}}{R}$$

Ecuación 10.1 Tiempo medio de fallas en un RAID.

El primer término es el tiempo medio de falla del primer disco, el cual es el tiempo medio de falla de un disco dividido por el número total de discos. El segundo

término se refiere a la probabilidad de otra falla en el grupo antes de que sea reparado el disco que falló con anterioridad.

Así como el número de discos del sistema se incrementa, la confiabilidad de este decae. Mas específicamente, asumiendo que las tasas de falla de los discos es idéntica, independiente y distribuida aleatoriamente de manera exponencial, un simple calculo de confiabilidad muestra que el tiempo medio entre perdida de datos para un grupo de N discos es 1/N veces el tiempo para un único disco [Patterson88].

La solución es tener algunos discos redundantes en la matriz, de forma tal que la falla de unos pocos discos no cause que la totalidad de los discos se tornen indisponibles. Este método es llamado RAID (Arreglo de Discos de Bajo Costo).

Esto es generalmente alcanzado mediante la técnica de discos espejos [16] o por codificación de paridades [Patterson88]. En el primero, una o mas copias de cada unidad de datos es almacenada en discos separados. En el segundo, (comúnmente conocido como Arreglo Redundante de Discos Independientes RAID, del inglés Redundant Arrays of Inexpensive Disks), una porción de la capacidad física del arreglo es utilizada para almacenar un código corrector de errores calculado sobre los datos almacenados en el arreglo.

Existen diferentes niveles de implementar un RAID. Las diferentes implementaciones proporcionan distintos niveles de costo beneficio, confiabilidad y rendimiento. La técnica de discos espejos puede ser tratado como el mas simple método de RAID, puesto que utiliza un arreglo de dos discos, uno de los cuales es redundante. Sin embargo, este es el método más costoso ya que tiene un 100% de sobreutilización, puesto que presenta un disco extra por cada disco.

Los estudios han demostrado que debido al superior rendimiento en las operaciones de pequeñas lecturas y escrituras, un arreglo de discos espejos, también conocido como RAID nivel 1, puede entregar un mas alto rendimiento a muchas e importantes cargas de trabajo que un arreglo basado en paridad [17]. Desafortunadamente, los discos espejos son substancialmente mas caros -- su sobrecarga de almacenamiento para redundancia es de 100% mientras que la sobrecarga de almacenamiento en un arreglo de paridad codificada es generalmente menor al 25% y puede ser incluso menor de 10%. Además, muchos estudio recientes [18] han demostrado técnicas que permiten que el rendimiento en las operaciones de pequeñas escrituras en los arreglos basados en paridad alcancen e incluso superen a los rendimientos de los discos espejos.

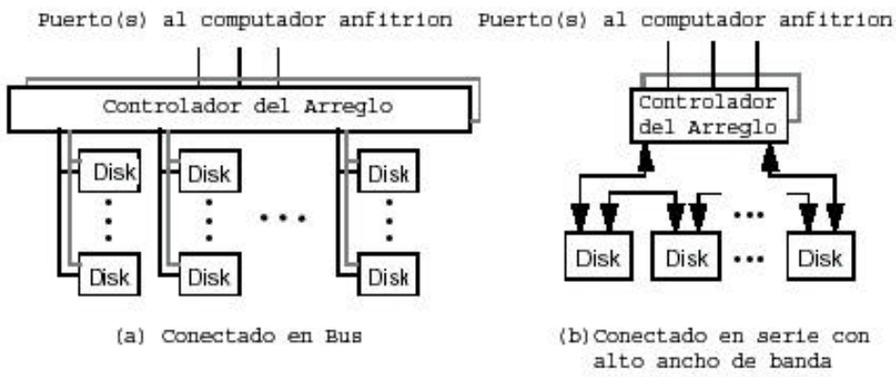

Figura 10.1 Arquitecturas de arreglos de discos.

Arquitectura de los arreglos de discos.

La figura 10.1 ilustra las dos posibles arquitecturas de arreglos de discos. Hoy en día los sistemas utilizan la arquitectura de la figura 10.1(a) en la cual los discos están conectados vía enlaces de bajo costo y bajo ancho de banda como por ejemplo SCSI hacia un controlador del arreglo, el cual esta conectado vía uno o mas buses paralelos de alto ancho de banda hacia uno o mas computadores anfitriones. Los controladores de

arreglos y buses de discos son frecuentemente duplicados (indicados por las líneas punteadas) de manera que ellos no representen un único punto de falla [16]. La funcionalidad del controlador además puede ser distribuida entre los discos del arreglo [19].

A medida que los discos se tornan más pequeños, los grandes cables utilizados por SCSI y otras interfaces de buses se vuelven aún menos atractivos. El sistema mostrado en la figura 10.1(b) ofrece una alternativa. Utiliza enlaces bidireccionales de alto ancho de banda para la interconexión de los discos. Esta arquitectura se puede llevar a arreglos de gran escala de manera mas fácil porque elimina la necesidad de que el controlador del arreglo incorpore un gran número de conexiones. Además, haciendo cada enlace bidireccional, proporciona dos caminos para cada disco sin duplicación de buses. En ambas organizaciones, el controlador del arreglo es el responsable de todas las actividades relacionadas con el sistema: Controlar los discos individuales, mantener la información redundante, ejecutar las transferencias solicitadas, y recuperación de discos o enlaces malos. La funcionalidad del controlador del arreglo puede también ser implementada en software que se ejecute en el anfitrión o anfitriones del subsistema.

Niveles de RAID

Un controlador del arreglo implementa la abstracción de un espacio lineal de direcciones. El arreglo aparece para el anfitrión como una secuencia lineal de unidades de datos, numeradas de 0 a N*(B-1), donde N es el número de discos en el arreglo y B es el número de unidades de datos del usuario en un disco. Las unidades que tienen código de corrección de errores no aparecen en el espacio de direcciones exportado por el controlador del arreglo; ellas no son direccionables por los programas de aplicaciones. El controlador del arreglo traduce direcciones de este espacio lineal en localizaciones

físicas del disco (identificador del disco y dirección del disco) y ejecuta el acceso solicitado. Es además responsable de ejecutar la mantención de la redundancia en los accesos de aplicaciones que impliquen operaciones de escritura. Esto se refiere al mapeo de una unidad lógica de almacenamiento de datos de una aplicación en localizaciones físicas del disco, y las localizaciones de los códigos de corrección de errores asociadas en la distribución del arreglo de discos.

Fundamental a todos los arreglos de discos esta el concepto de reparto (del inglés stripe) que consiste en la unión de unidades de datos de usuario a través de los discos del arreglo [3]. El reparto está definido como el trozado del espacio lineal de direcciones exportado por el controlador del arreglo en bloques de algún tamaño, y la asignación de bloques consecutivos en discos consecutivos en vez de llenar cada disco con datos consecutivos antes de cambiar al siguiente. La unidad de reparto [20] es el máximo monto de datos consecutivos asignados a un único disco. El controlador del arreglo tiene la libertad de configurar la unidad de reparto de manera arbitraria; la unidad puede ser tan pequeña como un único bit, o tan grande como un disco entero. El reparto tiene dos beneficios: balance de carga automático en cargas de trabajo concurrente y un alto ancho de banda para grandes transferencias secuenciales realizadas por un único proceso.

Los arreglos de discos consiguen el balance de carga en cargas de trabajo concurrente (aquellas que tienen varios procesos concurrentes accesando los datos almacenados) seleccionando la unidad de reparto de manera que sean mas grandes que la mayoría de los accesos pequeños que son servidos por un único disco. Entonces, un arreglo de N discos puede servir N solicitudes de entrada/salida en paralelo, utilizando cada una de ellas al ancho de banda de un único disco. Los arreglos adquieren altas tasas de transferencia en cargas de trabajo de baja concurrencia generando las unidades de reparto de un tamaño pequeño como por ejemplo un byte o un sector. Estos arreglos son

utilizados cuando la carga de trabajo esperada es un único proceso que solicita datos en bloques muy grandes. Las unidades de reparto de pequeño tamaño aseguran que cada acceso utiliza todos los discos en el arreglo, lo cual maximiza el rendimiento cuando la concurrencia de las cargas de trabajo (números de procesos) es uno. Después de las retardos asociados a la búsqueda inicial y al retardo rotacional de cada acceso, el arreglo con unidades de reparto de pequeño tamaño transfiere datos desde o hacia la CPU a N veces la tasa de un único disco. Por lo tanto, los arreglos con unidades de reparto de pequeño tamaño puede servir solo una entrada/salida en un tiempo determinado pero es capaz de leer o escribir el dato a una tasa muy alta.

Patterson, Gibson y Katz [3] clasificaron los arreglos de discos redundantes en cinco tipos llamados Raid de nivel 1 a 5, basados en la organización de la información redundante y la organización de los discos de datos. Esta terminología ha ganado una alta aceptación [21] y es la utilizada a través de toda esta tesis. El término RAID de nivel 0 se ha vuelto de común utilización para indicar un arreglo sin redundancia. La figura 10.2 ilustra la distribución de los datos para un RAID de nivel 0.

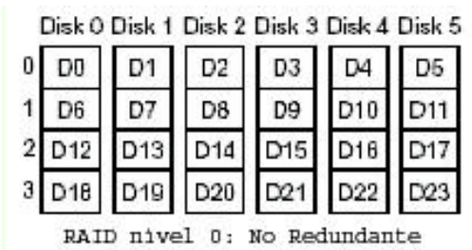

Figura 10.2 Raid de nivel 0.

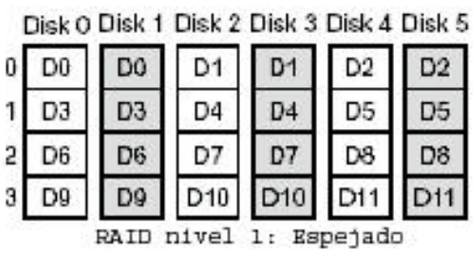

Figura 10.3 Raid de nivel 1.

Los Raid de nivel 1 (figura 10.3), también llamados discos espejos, son la técnica estándar utilizada para lograr tolerancia a fallos en los subsistemas tradicionales de almacenamiento de datos[16]. Los discos están agrupados en pares espejos, y una copia de cada bloque de datos es almacenado en cada uno de los dos discos. Los RAID de nivel 1 son una organización con una alta confiabilidad, puesto que el sistema puede tolerar múltiples fallos de discos (hasta N/2) sin perdida de datos, siempre y cuando no fallen los 2 discos de un mismo par espejado. Puede ser generalizado para proporcionar tolerancia a múltiples fallos manteniendo mas de 2 copias de cada unidad de datos. La desventaja de este nivel es que su costo por megabyte de almacenamiento es al menos el doble que un RAID de nivel 0.

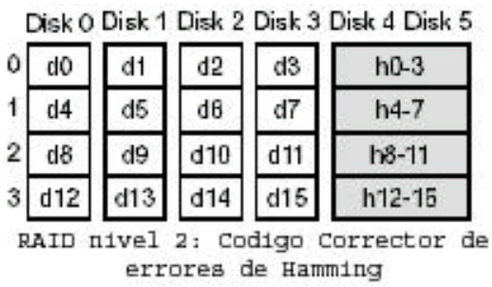

Figura 10.4 Raid de nivel 2

Los Raid de nivel 2 (figura 10.4) proporcionan una alta disponibilidad a bajo costo por megabyte utilizando técnicas bien conocidas usadas para proteger la memoria principal de perdidas transientes de datos. Los discos que conforman el arreglo son divididos en discos de datos y discos de chequeo. La información es repartida entre los discos de datos mientras que el disco de chequeo contiene código de Hamming de corrección de errores calculado sobre los datos en los correspondientes bits o bytes de los discos de datos. Esto reduce la sobrecarga de almacenamiento para redundancia desde un 100% en el espejado hacia un valor que varia entre 25% y 40% en el Raid de nivel 2 (dependiendo del número de discos de datos), pero reduce el número de fallas

que pueden ser toleradas sin pérdida de datos. Como veremos, la confiabilidad y rendimiento de tales sistemas puede ser muy alto. Puede ser ampliado para soportar tolerancia a múltiples fallos utilizando un código de Hamming tolerante a n fallas, el cual por supuesto incrementa tanto la cantidad de información necesaria en la redundancia como los cálculos necesarios para obtener los códigos.

Puesto que los discos contienen una amplia funcionalidad en detección y corrección de errores, y que estos se comunican con el mundo exterior mediante complejos protocolos, el controlador del arreglo puede identificar de manera directa los discos con fallo desde su información de estado o por su falla para trabajar con el protocolo de comunicación. Un sistema en el cual los componentes que fallas son autoidentificados es llamado un canal de borrado, para distinguirlo de un canal de error, en el cual las localizaciones de los errores no son conocidas.

Un código detector de n fallas para un canal de error se convierte en un código corrector de n fallas cuando es aplicado en un canal de borrado [22]. Los Raid de nivel 3 (figura 10.5) utilizan esta ventaja para reducir aun más la cantidad de información necesaria en la redundancia.

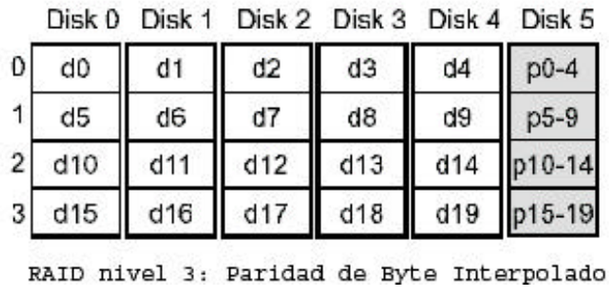

Figura 10.5 Raid de nivel 3

En los Raid de nivel 3, la información es repartida a través de los discos de datos y se utiliza un simple código de paridad para la protección contra la pérdida de datos. Un

único disco de chequeo (llamado disco de paridad) almacena la paridad (acumulación de or exclusivos) calculada sobre los correspondientes bits de los discos de datos. Esto reduce la cantidad de información necesaria en la redundancia a 1/N. Cuando el controlador identifica un disco como fallido, puede recuperar cualquier unidad con datos perdidos leyendo las correspondientes unidades desde todos los discos en buen estado, incluyendo el disco de paridad y aplicándole la operación XOR a todos ellos juntos.

Para ver esto, asumamos que el disco 2 en el Raid de nivel 3 mostrado en la figura ha fallado y nótese que si se asume que d_i es el dato en el disco i y p_{j-k} es la paridad entre los discos j y k entonces:

$$p_{0-4} = d_0 \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_4$$

Ecuación 10.2 Calculo de la paridad en un arreglo de discos.

Lo que implica:

$$d_2 = d_0 \oplus d_1 \oplus p_{0-4} \oplus d_3 \oplus d_4$$

Ecuación 10.3 Cálculo del contenido de un disco en un arreglo de discos.

La tolerancia a múltiples fallos puede ser lograda en los Raid de nivel 3 utilizando mas de un disco de chequeo y un código de detección/corrección de errores mas complejo tal como el Reed-Solomon [22] o código MDS [23]. Los Raid nivel 3 utilizan poca cantidad de información en la redundancia y proporcionan altas tasas de transferencia de datos. Puesto que los datos son divididos en pequeños unidades de reparto, cada acceso de usuario utiliza todos los discos del arreglo, y es por esto que solo se puede servir un acceso en un tiempo determinado. Por lo tanto este tipo de

organización se adapta mejor en aplicaciones del tipo científico, en las cuales un único proceso solicita grandes montos de datos secuenciales desde el arreglo.

Debido a que en el Raid nivel 3 todos los accesos utilizan todos los discos, los cabezales de los discos se mueven al unísono, y por lo tanto el cilindro sobre el cual los cabezales se encuentran localizados en un momento dado es siempre el mismo para todos los discos en el arreglo. Esto asegura que el tiempo de búsqueda para un acceso será siempre el mismo en todos los discos, lo cual elimina la condición en la cual algunos discos están ocupados esperando a otros que finalicen su porción del acceso. Para asegurar que la latencia rotacional sea también la misma para cada acceso en cada uno de los discos, los sistemas que utilizan Raid de nivel 3 normalmente utilizan una tecnología conocida como phase locked loop circuitry para sincronizar la rotación de las ejes que pertenecen al arreglo.

Figura 10.6 Raid de nivel 4.

Los Raid nivel 4 (figura 10.6) son idénticos a los de nivel 3 excepto que la unidad de reparto es relativamente mas grande (podría ser 32 Kb o más [20]), en vez de un único bit o byte. El bloque de paridad que protege un conjunto de unidades de datos es llamado unidad de paridad. Un conjunto de unidades de datos y su correspondiente unidad de paridad es llamada una franja de paridad. Los Raid de nivel 4 se adaptan a aplicaciones tales como procesamiento transaccional en línea (OLTP), en el cual un gran

número de procesos independientes solicitan de manera concurrente relativamente pequeñas unidades de datos desde el arreglo. Debido a que la unidad de reparto es pequeña, la probabilidad de que un acceso de bajo tamaño utilice mas de un disco es baja, y por esto el arreglo puede servir a un gran número de accesos concurrentes. Esta organización es además efectiva para cargas de trabajo que predominantemente tienen accesos de pequeño tamaño y alguna fracción de accesos de mayor tamaño. El arreglo sirve a los accesos de pequeño tamaño en paralelo pero logra una alta tasa de transferencia de datos en los ocasionales accesos de gran tamaño utilizando varios brazos de los discos. En los Raid de nivel 4, cada disco sirve normalmente a un acceso distinto y entonces, siempre y cuando las cargas de trabajo aplicadas no contengan una fracción significante de accesos de gran tamaño, los cabezales no permanecen sincronizados. El problema con los Raid de nivel 4 es que el disco de paridad puede tornarse un cuello de botella en las cargas de trabajo que contienen una fracción significativa de pequeñas operaciones de escritura. Cada actualización a una unidad de datos implica que la correspondiente unidad de paridad debe ser actualizada para reflejar el cambio. Entonces, el disco de paridad ejecuta una operación para cada actualización en cada disco de datos, y su utilización debida a operaciones de escritura es N-1 veces mayor que aquella para los discos de datos. Esto no ocurre en los Raid de nivel 2, puesto que cada acceso utiliza todos los discos. Para resolver este problema, los Raid de nivel 5 (figura 10.7) distribuyen la paridad entre los discos del arreglo. Esto asegura que la carga de trabajo debida a la actualización de la paridad es balanceada a través de los discos de igual forma que la carga de trabajo que actualiza los discos de datos.

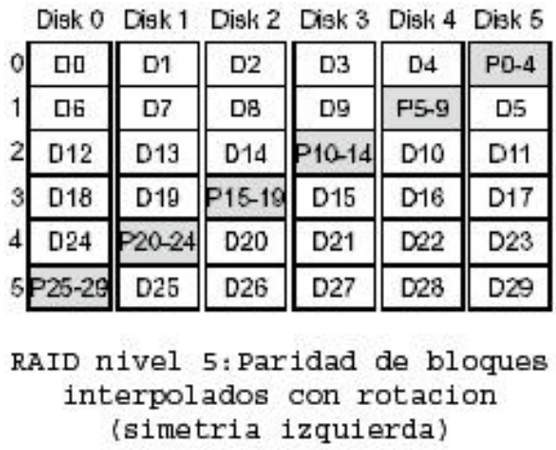

Figura 10.7 Raid de nivel 5.

En los Raid de nivel 5, existen una variedad de formas de distribuir los datos y la paridad sobre los discos del arreglo [24]. La estructura mostrada en la figura 10.7 es llamada la organización con simetría izquierda. Este método para asignar unidades de datos a las unidades de discos asegura que, si existe algún acceso en la carga de trabajo lo suficientemente grande para abarcar varias unidades de stripe, se utilizara el máximo número de discos posibles para servirlos. Existen muchos otros tipos de distribuciones de un RAID que no están dentro del alcance de esta tesis pero que pueden ser vistos en [9]

11. Unidades de Reparto.

La unidad de reparto es un bloque de datos de tamaño fijo que constituye la unidad en que los datos serán leídos y escritos en el sistema de archivos. Por ejemplo un archivo de 80 MB puede ser dividido en 40 unidades de reparto de 2 MB como se muestra en la figura 11.1. Dos unidades de reparto consecutivas tienen como característica fundamental el hecho de que son almacenadas en distintas unidades físicas, como por ejemplo discos distintos en un RAID o nodos distintos en un VRAID.

0	1	2	3	4	5	6	7	8	9	10	11	12	.	.	.	.	.	.	.	39

Figura 11.1 Unidades de reparto de un archivo de 80MB.

En la figura se aprecia la distribución de las unidades de reparto dentro de los distintos nodos de un VRAID, lo cual trae como principal ventaja el aumento del paralelismo ya sea al leer o al escribir datos. En el ejemplo de la figura, es posible leer al mismo tiempo las primeras 5 unidades de reparto desde un nodo cliente.

Nodo 0							
0	5	10	15	20	P4	30	35

Nodo 1							
1	6	11	16	P3	25	31	36

Nodo 2							
2	7	12	P2	21	26	32	37

Nodo 3							
3	8	P2	17	22	27	33	38

Nodo 4							
4	P1	13	18	23	28	34	P6

Nodo 5							
P0	9	14	19	24	29	P5	39

Figura 11.2 Distribución de los bloques dentro de un RAID 6 de nodos

De la misma manera, dentro del nodo 0, la unidad de reparto 0 se distribuye a través de los discos del nodo. Si se supone que el tamaño de la unidad de reparto del RAID de discos internos del nodo 0 es de 250 KB, entonces la unidad de reparto 0 del VRAID se encontrará dentro de las unidades de reparto 0 a 7 del RAID. De la misma forma, la unidad de reparto 5 del VRAID se encontrará dentro de las unidades de reparto 8 al 15 del RAID.

Nodo 0								
Disco 0		Disco 1		Disco 2		Disco 3		Disco 4
0		1		2		3		P0
4		5		6		P1		7
8		9		P2		10		11
12		P3		13		14		15
P4		16		17		18		19
20		21		22		23		P5
24		25		26		P6		27

Figura 11.3 Detalle de los bloques de un nodo dentro de un RAID.

La unidad de reparto que contiene un cálculo de paridad junto a las unidades de reparto que dependen de ella se encuentran agrupadas de forma lógica en un conjunto denominado Franja de Paridad. Por ejemplo: las unidades de reparto 0,1,2,3 y P0 forman la franja de paridad 0, las unidades de reparto 4,5,6,P1 y 7 forman la franja de paridad 1 y así sucesivamente.

12. Sistemas de Archivos Distribuidos

Un sistema de archivos distribuidos es aquel en el cual los archivos se encuentran repartidos en distintas unidades lógicas o físicas dentro de un conjunto de estas. Un sistema de archivos independiente de su tipo de distribución tiene características que lo definen tales como el servicio de archivos, el servidor de archivos, etc. A continuación se describen estas características y como éstas se encuentran implementadas dentro del simulador.

Servicio de archivos

El servicio de archivos, especifica los servicios que el sistema ofrece a sus clientes, además especifica la interfaz del servidor y las primitivas disponibles junto con sus parámetros y acciones. Dentro del simulador, se encuentra el servicio de archivos y su interfaz dentro de la librería fs.h, esta implementación entrega funcionalidades de inicialización donde se definen el tamaño de los bloques y el ancho de banda. En cuanto a las primitivas disponibles, estas se encuentran en la librería action.h, y se permiten lecturas, escrituras, bloqueos, desbloqueos y cálculos de paridad.

Servidor de archivos

El servidor de archivos es uno o más procesos que se ejecutan en alguna máquina y ayuda a implantar el servicio de archivos. Corre en el espacio de usuario, por lo que el sistema puede contener varios servidores de archivos con servicios de archivos diferentes. Además puede haber uno o más servidores de archivos, pero debe ser transparente a los clientes. Esta última situación es la que implementa el simulador en el archivo serv.h entregando funcionalidades de lectura, escritura, bloqueo y desbloqueo en los servidores. Sin embargo existe otra característica fundamental a los servidores de

archivos que debe implementarse, esta es la red que une a los nodos, la cual está representada en el archivo net.h, lugar donde se describe la topología, número de nodos y ancho de banda.

12.1 Diseño.

Un servidor de archivos generalmente tiene dos componentes importantes: el servicio de archivos y el servicio de directorios.

a) Servicio de archivos

Este ofrece operaciones sobre archivos individuales: leer, escribir, agregar, permite la administración de los atributos (información que no es parte de el archivo en si mismo), es responsable de la protección contra accesos no permitidos, a través de capacidades o listas de control de acceso y también es responsable del modelo de acceso. Estos últimos pueden ser:

Carga/descarga

En este modelo (figura 12.1), el cliente lee el archivo (rectángulo plomo de la figura) desde el servidor (1), lo copia en el disco del cliente de forma local, lo procesa y finalmente lo copia nuevamente a su lugar de origen dentro del servidor (2) (rectángulo negro en la figura).

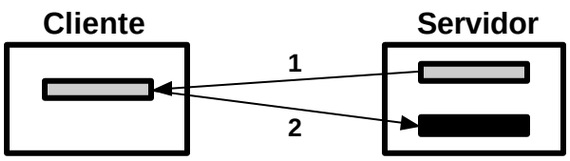

Figura 12.1 Modelo de lectura carga/descarga.

Acceso Remoto:

En este caso (figura 12.2), el cliente no copia el archivo (rectángulo plomo de la figura) en un disco duro local sino que trabaja directamente con el servidor.

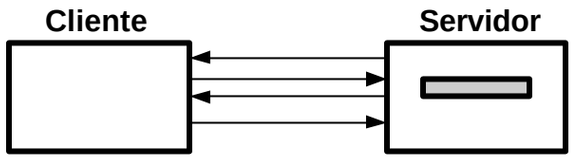

Figura 12.2 Modelo de lectura acceso remoto.

b) Servicio de directorios:

Este ofrece operaciones sobre directorios: crear y borrar directorios, copiar y mover archivos entre directorios, es responsable de resolver el nombre de los archivos, proporciona transparencia de los nombres con respecto a la localización, ofrece nombres de dos niveles: nombres simbólicos y nombres binarios, administra la organización de los archivos generalmente mediante un sistema jerárquico de archivos y maneja los enlaces lógicos y físicos. El simulador no proporciona servicio de directorios, pero si implementa los enlaces lógicos y físicos mediante la librería raidmap.h donde se encuentra el mapa que permite al sistema de archivos determinar la posición física de los archivos lógicos, es decir localiza el número del nodo y el bloque en el que se encuentran los datos.

c) Semántica de archivos compartidos:

Se refiere a cómo secuencializar las lecturas y escrituras sobre archivos compartidos de manera tal que no se lean valores obsoletos o se pierdan actualizaciones. Existen 5 formas bien estudiadas de afrontar esta problemática:

Semántica unix:

La premisa fundamental de este enfoque es que se impone en todas las operaciones un orden absoluto en función del tiempo y ante la lectura se retorna

el valor más reciente del dato. Cada operación en un archivo es visible a todos los procesos en forma instantánea, sin embargo esto solo se logra fácilmente si existe un único servidor de archivos y los clientes no hacen “caching” de sus archivos, es decir, no guardan copias locales de los archivos para su manipulación y posterior sincronización con el lugar de origen. En este caso todas las operaciones de lectura y escritura pasan directamente por el servidor de archivos que las procesa en forma secuencial. El desempeño de este método es pobre. Para mejorar el desempeño se puede permitir a los clientes tener copias locales de los archivos de uso frecuente en sus caches. Pero hay problemas de lecturas obsoletas los cuales solo se podrían evitar si se pudiese propagar inmediatamente las modificaciones al servidor o bien relajando la semántica de compartimiento.

Semántica de sesión:

En este enfoque, ningún cambio es visible a otros procesos hasta que el archivo se cierre, sin embargo por esto no todas las lecturas retornan el valor más reciente del dato. Si dos procesos tienen copias locales del mismo archivo y lo modifican al mismo tiempo, el resultado final depende de quién lo cierre más rápido.

Archivos inmutables:

Aquí no existen actualizaciones, es más fácil compartir y replicar. Las únicas operaciones permitidas son crear y leer, se pueden actualizar los directorios y reemplazar los archivos uno por uno. Sin embargo con esta metodología existen dos problemas a resolver: El primero tiene que ver con que sucedería si dos procesos intentan reemplazar el mismo archivo a la vez y el segundo con lo que ocurriría si un proceso reemplaza un archivo mientras otro lo está leyendo

12.2 Implantación.

a. Uso de archivos

Antes de implantar un sistema de archivos, es necesario conocer cual es el comportamiento en cuanto a la utilización de archivos. Por esto, Satyanarayanan en 1981 realizó algunas mediciones para estudiar los patrones de uso de los archivos utilizando dos metodologías:

Mediciones estáticas:

Este tipo de mediciones son una foto instantánea del sistema en ciertos momentos, los que revelan:

- La distribución de tamaños de los archivos.
- La distribución de los tipos de archivos
- La cantidad de espacio que ocupan los archivos.

Mediciones dinámicas:

En este tipo de mediciones, es el mismo servidor de archivos quien registra en una bitácora (log) todas las operaciones que realiza, para ser analizadas en forma posterior. Esta manera de medir el comportamiento del sistema de archivos revela:

- La frecuencia de las operaciones.
- El número de archivos abiertos
- La cantidad de archivos compartidos.

Las mediciones de Satyanarayanan fueron llevadas a cabo en una universidad utilizando sistemas Unix tradicionales, por lo que no responden a la pregunta de si este comportamiento era el mismo en un laboratorio de investigación, una oficina, un sistema empresarial o un entorno distribuido. A pesar de esto, fue posible concluir lo siguiente:

- La mayoría de los archivos está por debajo de 10K. Esto hace suponer que es mejor transferir entre cliente-servidor archivos completos en lugar de bloques de discos.

- La mayoría de los archivos tienen una corta vida, lo que lleva a suponer que es mejor crear el archivo en el cliente y mantenerlo ahí hasta su eliminación, disminuyendo de esta manera el tráfico entre cliente y servidor.

- Es poco usual compartir archivos por lo que es mejor usar semántica de sesión y hacer “caching” de los archivos en el cliente.

- Existen distintas clases de archivos con propiedades diferentes. Esto hace suponer que deben existir diferentes mecanismos para manejar diferentes clases de archivos.

- La lectura es más común que la escritura lo que favorece a la semántica de sesión.

- La lectura y escritura son secuenciales, es raro el acceso aleatorio.

b. Estructura del sistema:

Una segunda característica a conocer antes de implantar el sistema de archivos se refiere a la organización interna de los archivos y los directorios. Esto es responder a la pregunta: ¿Cómo estructurar el servicio de archivos y el servicio de directorios?. La respuesta a esto depende de si estos se encuentran combinados en un mismo servidor, caso en el cual las operaciones son directas, o si estos servicios se encuentran separados en distintos servidores, lo que significa que el abrir un archivo implica ir al servidor de directorios, localizar el archivo y luego ir al servidor de archivos para llevar a cabo la lectura o escritura. Para esto se requiere una mayor cantidad de datos que deben ser comunicados para sincronización pero se tiene la ventaja que es más flexible y el software que lo implementa es más sencillo. En la figura 12.3 se muestra el proceso a seguir cuando el servidor de archivos y el servidor de directorios se encuentran separados en servidores distintos. Lo primero que se realiza es entregar el nombre simbólico al servidor de directorios (1), el segundo paso es responder con el nombre binario (2), el tercer paso es enviar el nombre binario al servidor de archivos (3) quien finalmente responderá con el archivo solicitado (4).

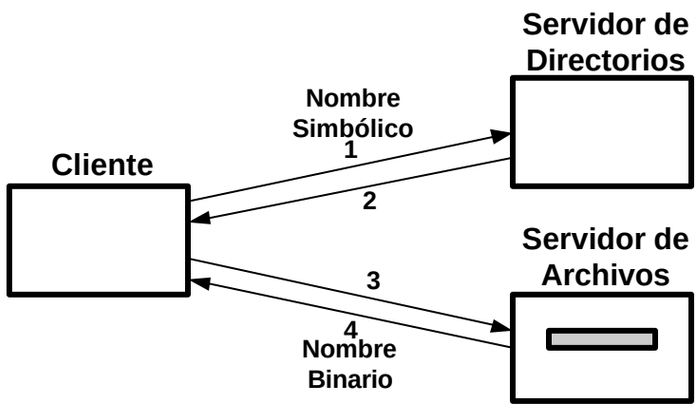

Figura 12.3 Modelo de lectura acceso remoto.

Los métodos existentes para la búsqueda de archivos cuando los servicios de archivos y directorios están separados son:

Búsqueda iterativa (figura 12.4):

El cliente en este caso está consciente de cual servidor contiene cual directorio, sin embargo se requiere un mayor intercambio de mensajes. En la figura se aprecia un ejemplo en el cual el cliente debe consultar a todos los servidores de directorios acerca de la ubicación de tres archivos.

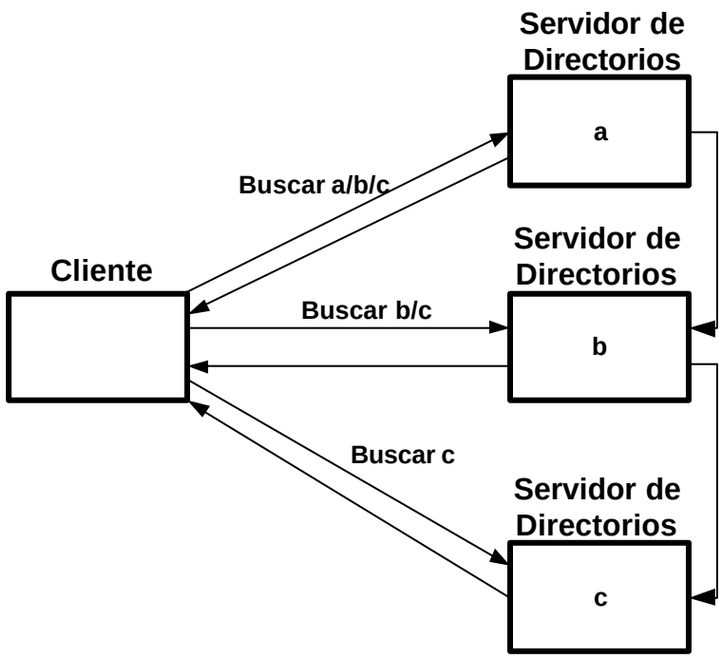

Figura 12.4 Modelo de búsqueda iterativa.

Búsqueda automática (figura 12.5):

Este tipo de búsqueda es más eficiente y transparente puesto que la solicitud se envía a solo uno de los servidores de directorios quien se encarga de continuar la

búsqueda en el resto de los servidores. Una vez encontrados todos los archivos, el último servidor devuelve las direcciones al cliente.

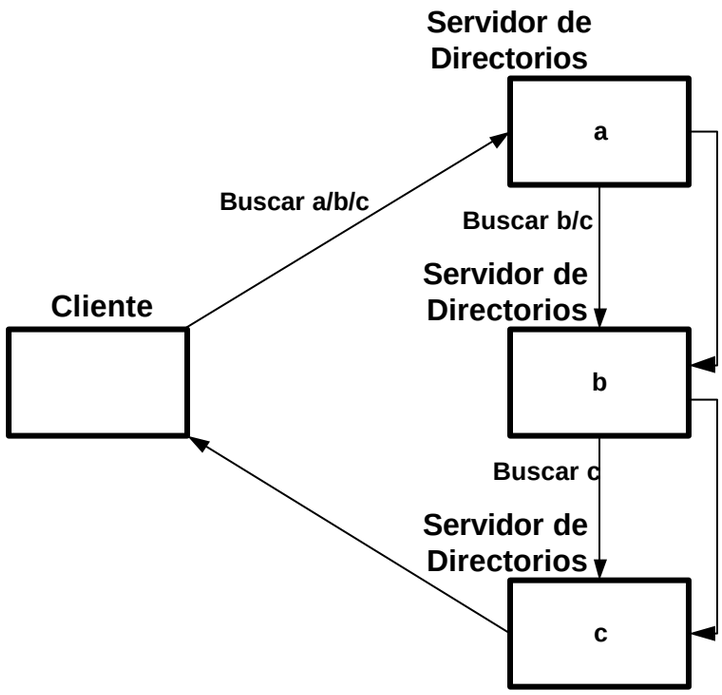

Figura 12.5 Modelo de búsqueda automática.

Otro tópico importante a considerar en cuanto a la estructura del sistema es el hecho de si los servidores de archivos y directorios deben o no contener la información de estado de los clientes. Para esto a continuación se explican cada una de las alternativas y se comparan:

Servidores sin estado (del ingles stateless)

Cuando un cliente envía una solicitud a un servidor, éste la lleva a cabo, envía la respuesta y elimina de sus tablas internas toda la información relativa a dicha solicitud. No guarda información del cliente entre solicitudes. Cada solicitud debe estar autocontenida.

Servidores con estado

Los servidores guardan información del estado de los clientes entre solicitudes: tabla que asocia los descriptores de archivos con los archivos propiamente dichos.

Ventajas de los servidores sin estado	Desventaja de los servidores con estado
Tolerante a fallas	La recuperación quedara a cargo del cliente
No se desperdicia espacio en el servidor	Se pueden desbordar las tablas y no se podrán abrir mas archivos
No existe limite para el número de archivos abiertos	Se pueden desbordar las tablas y no se podrán abrir mas archivos
No hay problemas si un cliente falla	Si un cliente falla después de abrir un archivo, el servidor está ante un dilema: - Sus tablas se llenan de basura. - Eliminar archivos inactivos
Mensajes mas cortos	Mensajes autocontenidos
Mejor desempeño porque las tablas pueden estar en memoria principal o en caché.	Desempeño pobre

Es posible realizar lecturas adelantadas	No es posible realizar lecturas adelantadas
Es fácil reconocer operaciones idempotentes	Es difícil reconocer operaciones idempotentes
Puede manejar el bloqueo de archivos	Se requiere un servidor de bloqueos especial

Tabla 12.1 Comparación entre servidores con estado y servidores sin estado.

c. Caching:

El caching se refiere al método que se utiliza para mejorar las prestaciones de la entrada/salida utilizando memoria adicional (figura 12.6) de manera que intente obtener por adelantado las futuras solicitudes y sirva de intermediaria entre el cliente y los discos. El problema que debe resolverse es determinar el lugar en el cual se realiza el caching, las alternativas son:

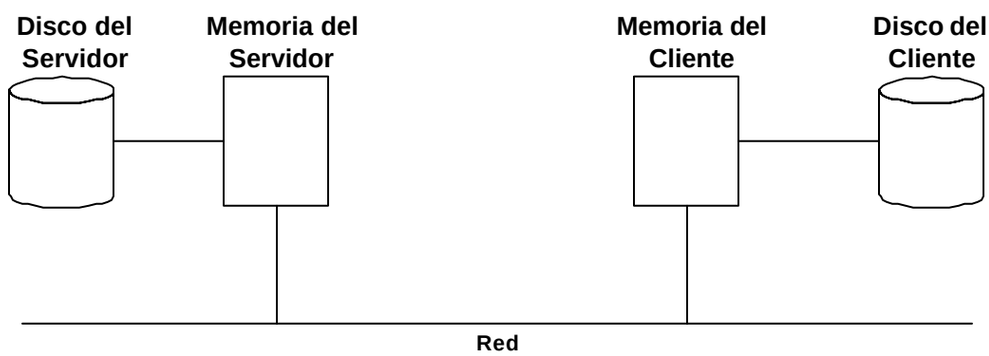

Figura 12.6 Componentes de Caching.

Guardarlo en el disco del servidor:

En este caso , la lectura adelantada se almacena en el disco del servidor donde existe un alto espacio de almacenamiento que permite adelantar una mayor cantidad de datos. Las ventajas de esta alternativa son que se requiere sólo una

copia de cada archivo, los archivos son accesibles a todos los clientes y no existen problemas de consistencia. Por otro lado se tiene que solo se puede conseguir un pobre desempeño.

Caching en memoria principal del servidor:

Aquí las ventajas son las mismas que la alternativa anterior solo que además se mejora notablemente el desempeño. El problema es que debido al limitado espacio de almacenamiento, se requiere determinar el tamaño de la unidad que administra el caché, es decir si será todo el archivo o solo algunos bloques. Otra desventaja es que se necesita un algoritmo de reemplazo cuando el caché esté lleno (LRU).

Caching en el cliente:

Cuando se realiza en el disco es lento, en general más lento que en el servidor sin embargo se logra un excelente desempeño cuando son muchos datos. Si el caching se realiza en memoria principal, este se realizará en el espacio de direcciones del cliente y será administrado por librerías con llamadas al sistema. Además cuando el proceso termina, los archivos modificados serán actualizados en el servidor.

El definitiva, el caching en el servidor no tiene efectos en la semántica del sistema de archivos mientras que el caching en el cliente ofrece mejor desempeño a costa de mayor complejidad y posible semántica más difusa.

Un punto importante a ser considerado dentro de los estudios acerca del caché tiene que ver con la consistencia que este debe presentar, es decir el hecho de tener que

mostrar siempre un mismo archivo para todos los clientes. Para esto, existen los siguientes algoritmos:

a. Escritura al cierre

Es la misma metodología que se utiliza en la Semántica de sesión. Esto consiste en que los archivos solo se actualizan cuando son cerrados por los clientes.

b. Algoritmo de escritura a través del caché (del ingles write-through caché)

Para mantener la consistencia en las otras caches, el administrador de caches debe verificar en el servidor antes de dar un archivo que está en caché a un nuevo cliente. Puede hacerlo mediante la comparación de fechas de última actualización, número de versión o sumas de verificación.

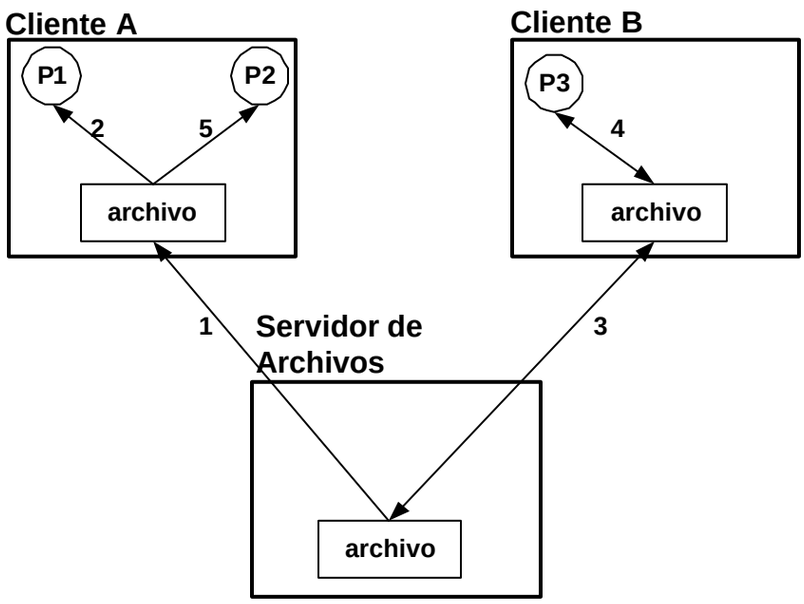

Figura 12.7 Inconsistencias de caché.

En el siguiente ejemplo (figura 12.7) se muestra la inconsistencia que puede ocurrir con este método. Se tiene en primer lugar que el proceso P1 solicita el

archivo desde el servidor (1) por lo que este se copia en su caché local (2). Luego, el proceso P3 también solicita el mismo archivo al servidor(3) y lo copia a su caché local. El proceso P3 entonces procesa el archivo solicitado y lo vuelve a escribir al servidor (4 y 3). Ahora ocurre que P2 solicita la utilización del archivo por lo que lo va a buscar lugar mas cercano que resulta ser el caché local, donde se encuentra una copia obsoleta del archivo produciéndose un problema de consistencia.

c. Escrituras retardadas.

En esta metodología, se envían las actualizaciones cada cierto tiempo, lo que mejora el desempeño, pero genera ambigüedades en la semántica.

d. Control centralizado:

El servidor de archivos da permisos de accesos a los archivos dependiendo de si está abierto por otros procesos para lectura y/o escritura. Aquí es posible soportar la semántica Unix, pero no es robusto y es poco escalable.

d. Réplicas:

Las razones para el servicio de réplicas es el aumento de confiabilidad al disponer de respaldos independientes de cada archivo, lo que trae una mayor disponibilidad de los datos puesto que permite el acceso aunque falle uno de los servidores y además permite repartir la carga de trabajo entre los estos últimos. Las formas de crear réplicas son:

Réplica explícita:

En esta situación (figura 12.8) es el cliente quien controla el proceso ya que cuando crea un archivo lo hace en un servidor específico y adicionalmente puede

crear copias en otros servidores. Luego se registran en el cliente las copias y si posteriormente se quiere abrir de nuevo un archivo, este se busca el primer servidor disponible.

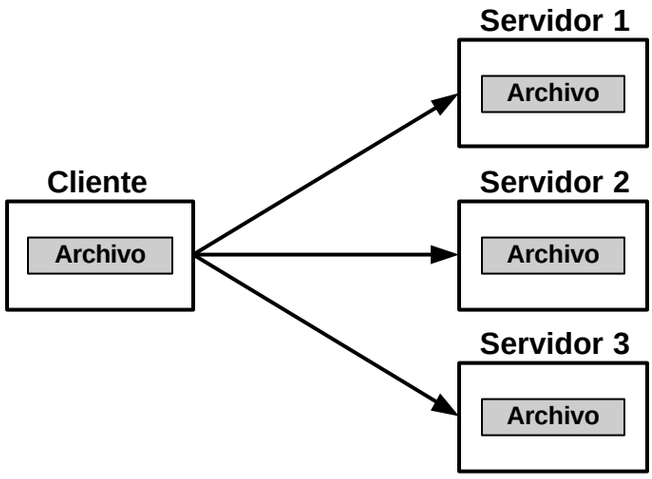

Figura 12.8 Replicación explícita.

Réplica retrasada:

En esta opción (figura 12.9), el cliente crea el archivo en un servidor, luego en forma automática el servidor crea las copias sin conocimiento del cliente.

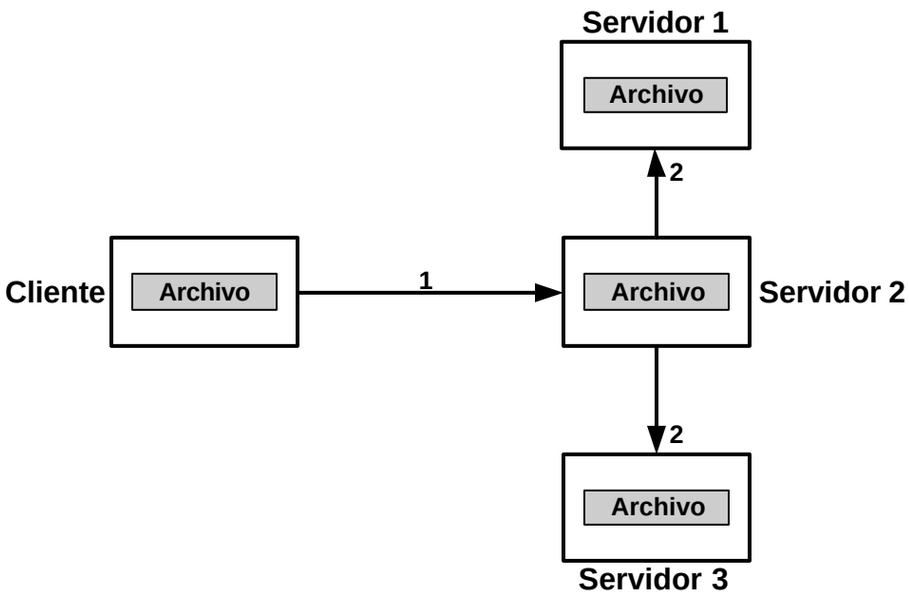

Figura 12.9 Replicación retrasada.

Método del voto:

Aquí (figura 12.10) los clientes solicitan y requieren permiso de varios servidores antes de leer o escribir en un archivo replicado. Por ejemplo para leer de un archivo de N réplicas, un cliente necesita un quórum de lectura de Nr servidores o más. Por otro lado, para modificar se requiere un quórum de escritura de Nw servidores y se debe cumplir que la suma entre el quórum de lectura Nr y el quórum de escritura Nw debe ser mayor que el número total de servidores N. El voto consiste del número de la versión asociado al archivo.

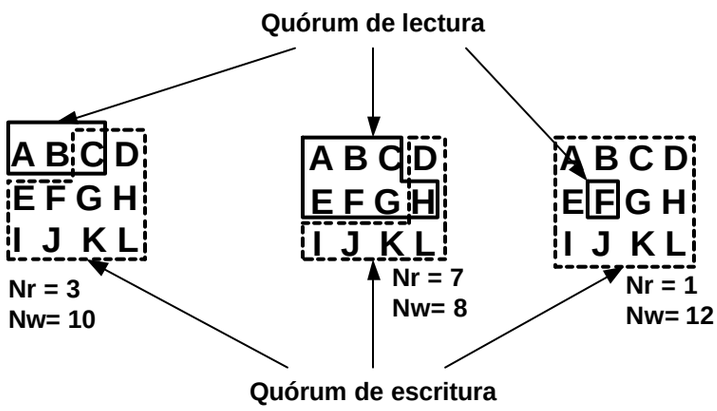

Figura 12.10 Método del voto.

El problema que surge con este método tiene que ver con lo que ocurre cuando fallan muchos servidores y no se puede cumplir que la suma de los quórum sea mayor que el total.

Voto con fantasmas:

Este método nació con el objetivo de solucionar el problema que surgía del método anterior. La idea es crear un servidor fantasma sin espacio de

almacenamiento para cada servidor real que falle. En las lecturas no se permiten fantasmas y las escrituras sólo tienen éxito si al menos uno de los servidores es real. Cuando se levanta un servidor, se obtiene un quórum de lectura para localizar la versión más reciente.

Capítulo III

13.Modelo de un Sistema de Archivos Distribuido y Paralelo

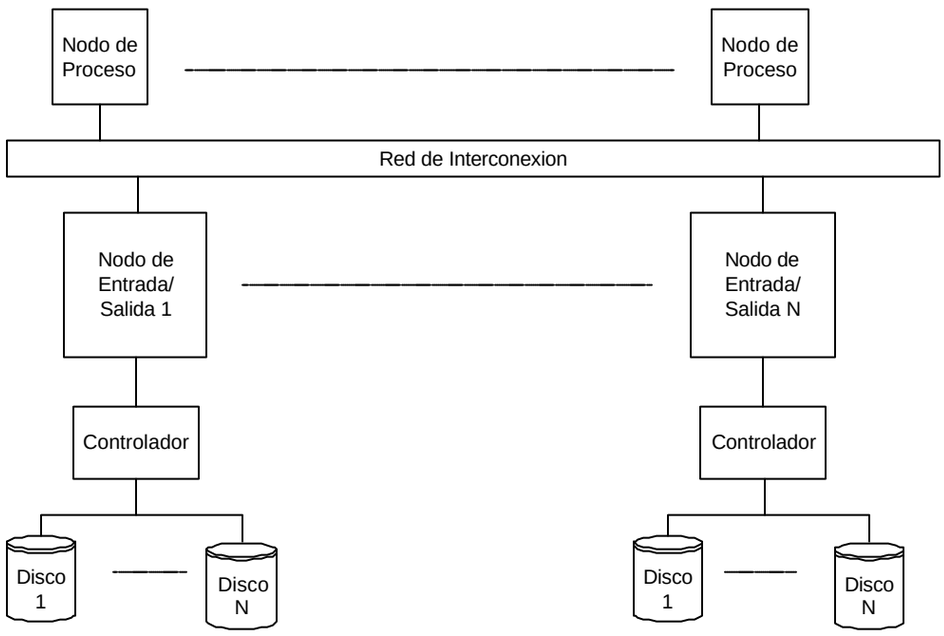

Figura 13.1 Esquema de un sistema de archivos distribuido y paralelo.

Como se explicaba en capítulos anteriores, un sistema de archivos distribuido y paralelo es aquel en el cual la información se encuentra distribuida dentro de un determinado número de nodos dentro de una red (figura 13.1). El paralelismo dentro de estos sistemas de archivo se asocia a su capacidad de enviar o recibir información a uno o mas nodos dentro de la red en un mismo instante de tiempo.

Lo anterior se representa por una matriz de m filas y n columnas (Figura 13.2).

$$\begin{array}{c} \\ F0 \\ F1 \\ F2 \\ F3 \\ F4 \\ F5 \\ F6 \\ F7 \\ F8 \\ F9 \end{array} \begin{array}{c} \begin{array}{cccc} N0 & N1 & N2 & N3 \end{array} \\ \left[\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 \\ 36 & 37 & 38 & 39 \end{array}\right] \end{array}$$

Figura 13.2 Matriz de un sistema de archivos distribuido y paralelo.

Donde n es el número de nodos de almacenamiento existentes dentro de la red y m es el número de bloques de almacenamiento dentro de cada uno de los nodos. Cada elemento de la matriz es un bloque dentro del sistema de archivos distribuido existiendo de esta manera n*m bloques de almacenamiento.

Para simular los procesos y módulos que intervienen dentro y fuera de un sistema de archivos distribuido y paralelo se propuso un modelo con las siguientes capas de software (Figura 13.3):

Figura 13.3 Bloques de un sistema de archivos distribuido y paralelo.

La descripción de cada uno de los módulos es la siguiente:

Trabajadores: Este módulo se encarga de simular la generación de todas las solicitudes de entrada/salida al sistema de archivos.

Sistema de Archivos Paralelo: Este módulo representa un sistema de archivos que lanza las peticiones hacia los dispositivos subyacentes [9]. Para esto simula los procesos en que deben dividirse las solicitudes de entrada/salida con el fin de ser enviadas a los nodos respectivos.

Red: Modelo de una red de trabajo que interconecta nodos con CPUs, puede configurarse de diferentes formas, lo cual determina sus prestaciones. Interconecta y crea una cantidad de nodos que componen el sistema [9].

Servidores: Implementa un servidor de entrada/salida el cual esta encargado de monitorear los tiempos de transferencia por la red.

Raid: Modela un dispositivo Raid de niveles 0,4 o 5, los que abarcan el conjunto de todos los discos de un nodo. Distribuye la operación solicitada en operaciones a los discos involucrados. Este módulo despacha las operaciones que se le solicitan en el orden en que llegan, de forma secuencial, pero explotando la concurrencia entre los discos [9].

Bus de Datos: Modelo de un bus de entrada/salida, puede configurarse de variadas formas que determinan sus prestaciones. Un bus transmite información desde un nodo a un disco o viceversa. Pueden existir varios buses por nodo [9].

Disco Duro: Modelo de un disco duro. Los discos pueden ser de distintos tipos lo cual determina sus prestaciones. Un disco ira conectado a un bus. Puede haber varios discos por bus [9].

14. Modelo de un Sistema de Archivos Distribuido y Paralelo con redundancia de datos.

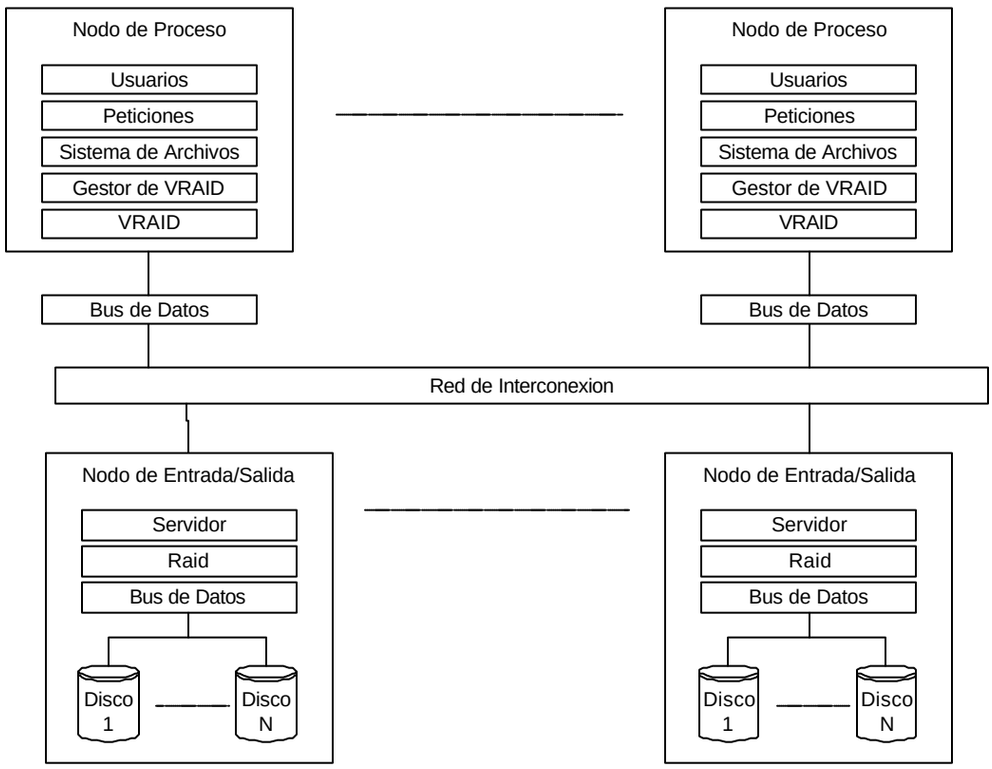

Figura 14.1 Esquema de un sistema de archivos distribuido y paralelo con redundancia de datos.

Un sistema de archivos distribuido y paralelo con redundancia de datos es aquel en el cual la información que se encuentra distribuida dentro de los nodos incorpora información redundante que le proporciona al sistema características de tolerancia a fallos (figura 14.1). Es decir, el sistema es capaz de permitir la falla de uno o más de los nodos dependiendo de la cantidad de información redundante que se desee incorporar sin perder la disponibilidad de la información. Un sistema de archivos distribuido y paralelo con redundancia de datos que implemente un algoritmo conocido

como matriz de discos redundantes de nivel 5 se representa por una matriz de m filas y n columnas (Figura 14-2).

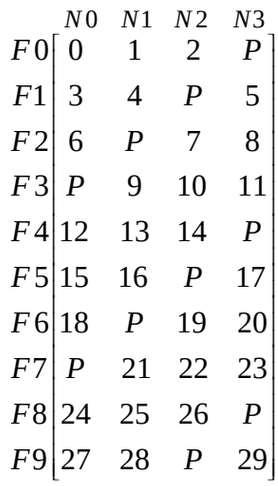

Figura 14.2 Matriz de un sistema de archivos distribuido y paralelo con tolerancia de fallos.

Donde n es el número de nodos de almacenamiento existentes dentro de la red y m es el número de bloques de almacenamiento dentro de cada uno de los nodos. Cada fila de la matriz es conocida como Franja de Paridad y cada elemento de la matriz es un bloque dentro del sistema de archivos distribuido. En cada una de las franjas de paridad existe un bloque representado por una letra P que almacena la información redundante calculada en base al resto de los bloques que comparten las misma franja de paridad. De esta manera existen m*(n-1) bloques de almacenamiento.

En cuanto al modelo de capas de software que representa los procesos y módulos involucrados en un sistema de archivos distribuido y paralelo con redundancia de datos (Figura 14-3) este consta con las siguientes capas adicionales respecto al modelo de la figura 13.3:

Figura 14.3 Bloques de un sistema de archivos distribuido y paralelo con redundancia de datos.

Gestor de Virtual Raid: Un dispositivo gestionador de virtual Raid que agrupa componentes de VRAID que permite al sistema redundante aumentar su disponibilidad de datos [9].

VRAID: Modela un dispositivo virtual Raid de niveles 0, 4 o 5. Este módulo despacha las operaciones que le solicitan los usuarios en el orden en que llegan, de forma secuencial, pero explotando la concurrencia entre los nodos[9].

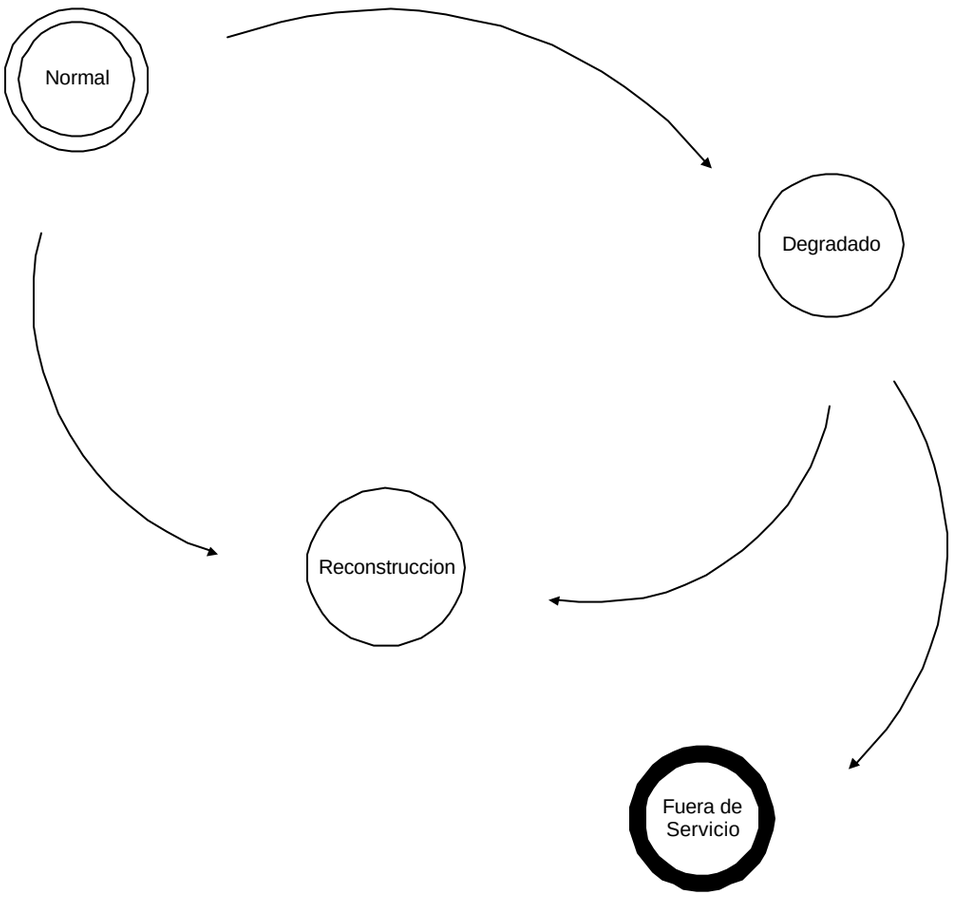

Figura 14.4 Estados de un sistema de archivos distribuido y paralelo con tolerancia de fallos.

En un sistema de archivos distribuido y paralelo con redundancia de datos se distinguen los siguientes estados en el cual puede encontrarse el sistema en un instante determinado (figura 14.4):

- **Estado Normal**:

  Este estado indica que todos los servicios del sistema se encuentran en correcto funcionamiento. Esto quiere decir que no existe ningún nodo que

se encuentre con bloques de datos no disponibles debido a un fallo de los discos o de los mismos nodos.

- **Estado Degradado**:

  Este estado indica que el sistema esta presentando un fallo en algún nodo o en algún disco dentro de un nodo que impide el normal funcionamiento del VRAID. Sin embargo, esto no significa que el sistema no puede entregar sus servicios, sino que estos tomarán un mayor tiempo debido al necesario cálculo de los datos que no se encuentran explícitos sino que deben ser interpolados con los códigos correctores de errores.

- **Estado Reconstrucción**:

  Al igual que el estado anterior, este estado indica que se existe un nodo o un disco en algún nodo que no permite la normal entrega de los servicios, sin embargo en este estado el sistema se encuentra con el fallo corregido pero sin los datos que debe contener, y es por esto que se ejecuta un proceso de reconstrucción de datos que regenera mediante los códigos correctores de errores los datos que se encontraban en la unidad fallida.

- **Estado Fuera de Servicio**:

  Este estado indica que el sistema se encuentra fuera de servicio y le es imposible entregar sus servicios ni siquiera de manera parcial o degradada. La ocurrencia de este estado la determina el fallo de mas de un nodo o disco dentro un nodo.

Criterio de Descomposición de Acciones

Dentro del simulador de un sistema de archivos distribuido y paralelo con redundancia de datos creado en [8], es necesario gestionar de una misma manera distintos tipos de dispositivos, por lo que se ocultaron los detalles de la implementación bajo una capa cuya interfaz es un dispositivo abstracto de almacenamiento[8]. Así, este dispositivo abstracto de almacenamiento es el proveedor de un conjunto de acciones de entrada/salida para una determinada cantidad de unidades de almacenamiento de un tamaño especifico. Esta interfaz, proporciona el siguiente conjunto de acciones básicas que permite descomponer operaciones de lectura y escritura de cualquier dimensión en las siguientes primitivas:

LOCK

Esta operación permite bloquear una unidad determinada para impedir que otras solicitudes lean o escriban mientras se realizan modificaciones que pudiesen generar un error debido a la obsolescencia del bloque.

READ

Esta operación lee un número de bloque o unidades a partir de una unidad dada.

XOR

Esta operación calcula el código corrector de errores mediante el uso del operador or-exclusivo dentro de la franja de paridad indicada.

WRITE

Esta operación permite escribir un número de unidades o bloques determinados a partir de una unidad dada.

UNLOCK

Esta operación permite desbloquear una unidad especifica que se encuentre bloqueada debido a la anterior utilización de la primitiva LOCK.

Cuando un dispositivo abstracto realiza una acción de lectura o escritura, esta se descompone en un conjunto de operaciones menores dirigidas a sus dispositivos subyacentes, aplicando para cada franja de paridad el siguiente criterio:

Acción Lecturas:

READ: Lectura de las unidades solicitadas

Acción Escritura Total:

LOCK: Bloqueo de la paridad en la franja donde se esta realizando la solicitud.

XOR: Cálculo de la nueva paridad en base a los nuevos datos.

WRITE: Escritura de las unidades y la nueva paridad.

UNLOCK: Desbloqueo de la paridad de la franja.

Acción Escritura Parcial:

LOCK: Bloqueo de la paridad en la franja donde se esta realizando la solicitud.

READ: Lectura de las unidades que no serán reescritas dentro de la franja de paridad.

XOR: Cálculo de la nueva paridad en base a los nuevos datos y los leídos en la operación anterior.

WRITE: Escritura de las unidades respectivas y de la nueva paridad.

UNLOCK: Desbloqueo de la paridad de la franja.

Acción Escritura Larga:

LOCK: Bloqueo de la paridad en la franja donde se está realizando la solicitud.

READ: Lectura de las unidades que no serán reescritas dentro de la franja de paridad.

XOR: Cálculo de la nueva paridad en base a los nuevos datos y los leídos en la operación anterior.

WRITE: Escritura de las unidades respectivas y de la nueva paridad.

UNLOCK: Desbloqueo de la paridad de la franja.

Como se puede ver, las operaciones de escritura utilizan cerrojos (operaciones LOCK y UNLOCK) para asegurar la consistencia entre los datos y la unidad de paridad. Dependiendo del tamaño, una acción de entrada/salida puede descomponerse en un gran número de subacciones sobre un conjunto de unidades de almacenamiento de los dispositivos subyacentes. Para reducir la cantidad de subacciones individuales y para optimizar el acceso a los dispositivos subyacentes, este conjunto es reordenado juntando subacciones que son lógicamente contiguas. Esto es:

1) Se refieren al mismo dispositivo de almacenamiento subyacente.
2) Son del mismo tipo de acción (lock, read, xor, write or unlock)
3) Conciernen a un conjunto contiguo de unidades.

El conjunto resultante de subacciones es procesado ejecutándose en paralelo las acciones para cada uno de los dispositivos, haciéndolos en el siguiente orden: todos los bloqueos, todas las lecturas, el cálculo interno del Xor, todas las escrituras y finalmente los desbloqueos. Este método tiene las siguientes propiedades:

a) Asegura la consistencia entre los datos y la paridad de cada franja de paridad involucrada.

b) Minimiza el número final de acciones y por tanto (en el caso de VRAID) el tráfico de red.

c) Las acciones finales por dispositivo son mas compactas y puede ser realizadas mas eficientemente.

15. Gestión Distribuida de Cerrojos en el VRAID

En el VRAID, el cálculo de la paridad se realiza en el nodo que hace la solicitud de entrada/salida, por tal motivo es necesario un mecanismo de cerrojos para asegurar el orden correcto de los accesos remotos paralelos Se ha decidido localizar el servicio de cerrojos en la librería de la capa Servidores y bloquear de esta manera solo la unidad de paridad involucrada. Por tanto, la distribución de los cerrojos será la misma que la de las unidades de paridad. Esto significa tres cosas:

a) La distribución de los cerrojos asegurara consistencia en la unidad de paridad.

b) Esto no supone un cuello de botella mayor que el acceso a la unidad de paridad en si misma.

c) Habrá también un consenso distribuido acerca de cual servidor de cerrojos usar en caso de que el sistema se degrade.

16. Modelo del problema.

El problema se puede modelar como un arreglo de m filas y n columnas (Figura 16-1) en el cual las columnas representan los nodos del VRAID y las filas las franjas de paridad dentro de cada uno de dichos nodos. Cada elemento de la matriz identifica un bloque con información, si el elemento esta representado por una P significa que ese bloque es utilizado para almacenar la paridad de la franja respectiva.

	$N0$	$N1$	$N2$	$N3$
$F0$	0	1	2	P
$F1$	3	4	P	5
$F2$	6	P	7	8
$F3$	P	9	10	11
$F4$	12	13	14	P
$F5$	15	16	P	17
$F6$	18	P	19	20
$F7$	P	21	22	23
$F8$	24	25	26	P
$F9$	27	28	P	29

Figura 16.1 Matriz de un sistema de archivos distribuido y paralelo con tolerancia de fallos.

El problema a resolver radica en otorgarle al sistema la capacidad de incorporar nuevos nodos (Figura 16.2), es decir agregar mas columnas a la matriz de manera que se reordenen los bloques dentro de la nueva matriz, cuidando que los bloques que almacenan información de paridad sean recalculados y no redistribuidos. Este proceso es el que se conoce como reconfiguración (Figura 11.3).

	N0	N1	N2	N3	N4	N5
F0	0	1	2	P	–	–
F1	3	4	P	5	–	–
F2	6	P	7	8	–	–
F3	P	9	10	11	–	–
F4	12	13	14	P	–	–
F5	15	16	P	17	–	–

Figura 16.2 Matriz de un sistema de archivos distribuido y paralelo con tolerancia de fallos antes del proceso de reconfiguración dinámica.

	N0	N1	N2	N3	N4	N5
F0	0	1	2	3	4	P
F1	5	6	7	8	P	9
F2	10	11	12	P	13	14
F3	15	16	P	17	18	19
F4	20	P	21	22	23	24
F5	P	25	26	27	28	29

Figura 16.3 Matriz de un sistema de archivos distribuido y paralelo con tolerancia de fallos después del proceso de reconfiguración dinámica.

En base a lo mostrado, se han planteado dos nuevos modelos que permitan implementar la solución al problema.

17. Primer Modelo de Reconfiguración Dinámica.

Este primer modelo incorpora una nueva capa de software al sistema de archivos distribuido y paralelo con redundancia de datos (Figura 17.1).

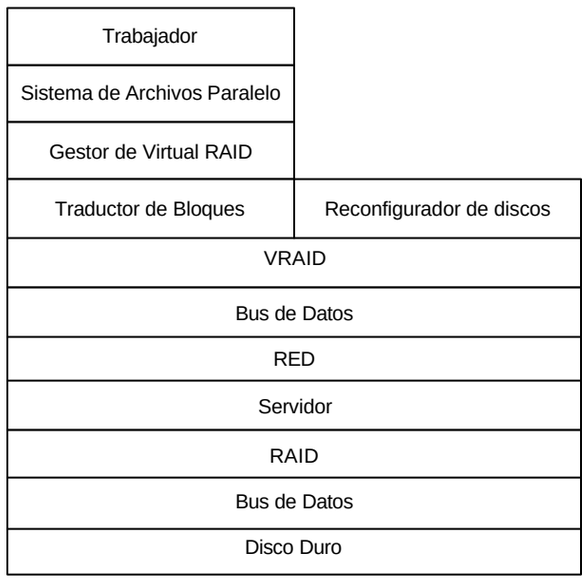

Figura 17.1 Bloques de un primer modelo de sistema de archivos distribuido y paralelo con tolerancia a fallos y reconfiguración dinámica

Esta nueva capa contiene 2 procesos que trabajan en forma concurrente, el primero de ellos denominado *Traductor de Bloques* se encarga de transformar las solicitudes de lectura/escritura que vienen dirigidas a nodos determinados desde los trabajadores a nuevas solicitudes dirigidas a la nueva configuración del raidmap. El segundo bloque llamado *Reconfigurador de discos*, es un proceso que tiene como misión realizar todas las operaciones necesarias para que el sistema incorpore los nuevos nodos y redistribuya la información. Desde el punto de vista de las librerías que componen el sistema de archivos distribuido y paralelo con redundancia de datos la nueva disposición de las capas queda representada en la figura 17.2

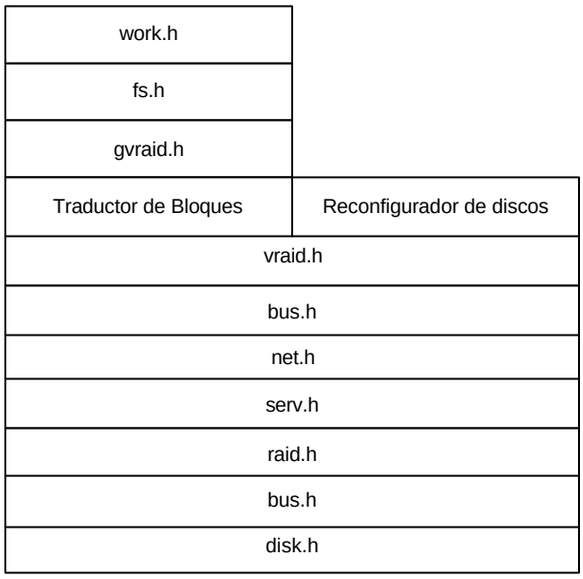

Figura 17.2 Librerías de un primer modelo de sistema de archivos distribuido y paralelo con tolerancia a fallos y reconfiguración dinámica

La descripción de los nuevos bloques es la siguiente:

Traductor de Bloques:

Como se explicó anteriormente, este proceso se encarga de redireccionar las solicitudes de entrada/salida provenientes desde los trabajadores. La motivación para crear este proceso viene de la necesidad de resolver la problemática consistente en que el sistema de archivos no debe detener la entrega de los servicios mientras se encuentre en un estado de reconfiguración. A medida que el proceso de reconfiguración va avanzando por la matriz, esta queda particionada en tres zonas. La primera zona contiene las franjas de paridad que han sido reconfiguradas y cuya distribución abarca la totalidad de los nodos. La segunda zona esta conformada por las franjas de paridad que se encuentran bloqueadas puesto que el proceso reconfigurador esta trabajando sobre ellas. Finalmente la tercera zona esta compuesta por las franjas de paridad que aún no han sido reconfiguradas y que todavía se encuentran distribuidas como en un principio.

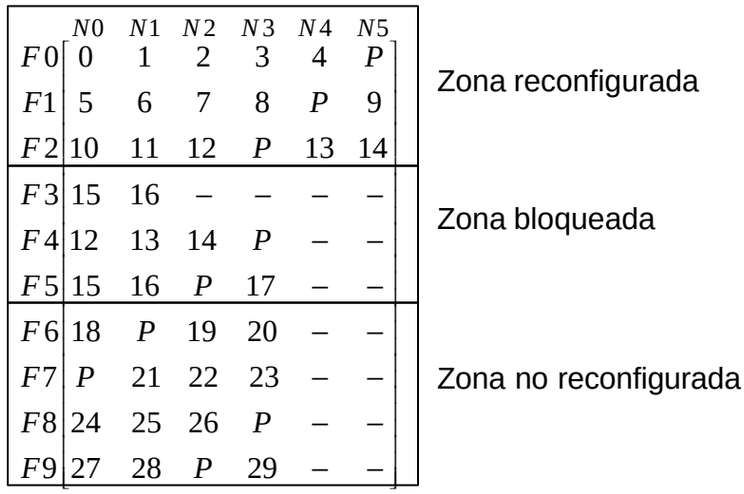

Figura 17.3 Zonas de reconfiguración.

Debido a que la matriz se encuentra dividida en tres zonas (figura 17.3), el proceso traductor de direcciones debe ser capaz de discriminar a cual zona va dirigida la solicitud para de esta manera redireccionar la petición. Si la solicitud va dirigida a la zona reconfigurada el traductor de direcciones enviará esta solicitud a una nueva posición que será calculada en base a las expresiones de la ecuación 17.1:

$$j\_proy = \text{int}\left(\frac{bloque}{n-1}\right)$$

$$i\_proy = \left(bloque\%\left(n-1\right)\right) + \text{int}\left(\frac{bloque}{\left(\left(n-2\right)*\left(j\_proy+1\right)\right)+1+\left(n*\text{int}\left(\frac{j\_proy}{n}\right)\right)}\right)$$

donde:

n: número de nodos del sistema.

bloque: número del bloque a trasladar [0..(m*(n-1))-1]

j_proy: número de la franja de paridad en la cual será trasladado el bloque [0..m-1].

i_proy: número del nodo en el cual será trasladado el bloque[0..n-1].

Ecuación 17.1 Ecuaciones de traducción.

Si la solicitud va dirigida a la zona bloqueada, el traductor de direcciones se encargará de postergar la solicitud hasta que dicha zona sea reconfigurada, para esto se utiliza una técnica de programación conocida como Semáforos.

Finalmente si la solicitud va dirigida a la zona no reconfigurada, el traductor de direcciones solo se encarga de retransmitir la petición tal cual fue realizada puesto que los bloques aún se encuentran en la misma posición.

Reconfigurador de Discos:

Este módulo se encarga de redistribuir la información contenida en los nodos de forma tal que ahora incorpore a los nuevos nodos que se han agregado al sistema. Para esto hace un recorrido por la matriz, bloqueando las franjas de paridad que contienen a los bloques que en ese momento se están reconfigurando, con el fin de que ningún otro proceso lea o escriba en dichos bloques. Un parámetro importante para el rendimiento del sistema durante el funcionamiento de este módulo, es el número de franjas de paridad que deben bloquearse (figura 17.4) la cual esta determinada por el número de bloques que contendrá el grupo a reconfigurar.

Es en este proceso, donde se plantean dos alternativas de solución posible, la primera de ellas consiste en lo siguiente:

1) Detener en una cola de espera todas las nuevas operaciones de entrada/salida que provengan desde los trabajadores.

2) Esperar que finalizen todas las operaciones de entrada/salida que ya se encuentran en el VRAID.

3) Iniciar un reseteo del VRAID con la finalidad de incorporar los nuevos nodos.

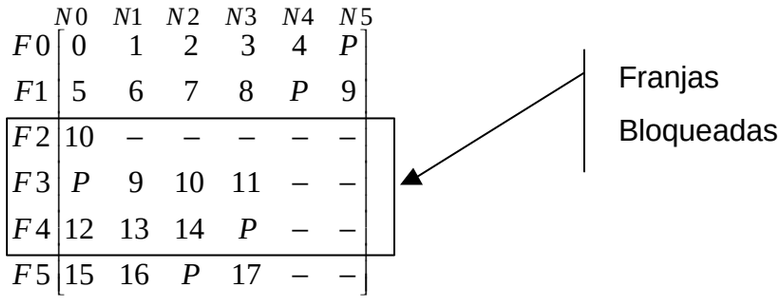

Figura 17.4 Bloqueo de franjas de paridad.

18. Segundo Modelo de Reconfiguración Dinámica.

La segunda solución (figura 18.1) consiste en:

1) Crear un nuevo VRAID que apunte a los mismos nodos pero que además incorpore los nuevos.

2) En forma paralela finalizar en el VRAID original todas las operaciones de entrada/salida existentes y además redireccionar todas las nuevas operaciones procedentes desde los trabajadores hacia el nuevo VRAID.

3) Eliminar el puntero al VRAID original y dejar el nuevo VRAID como el original.

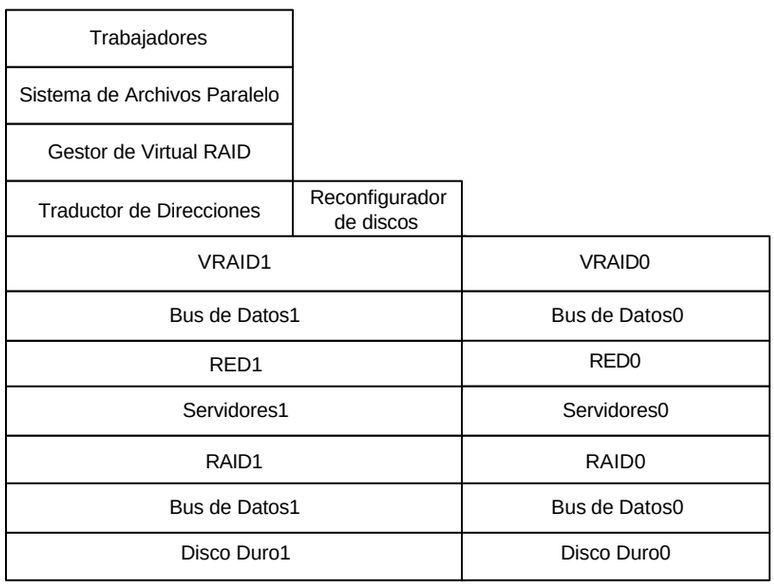

Figura 18.1 Bloques de un segundo modelo de sistema de archivos distribuido y paralelo con tolerancia a fallos y reconfiguración dinámica

A su vez, dentro del proceso *Reconfigurador de discos*, se tienen dos alternativas para solucionar el problema de cómo redistribuir los bloques a las

nuevas posiciones. La primera de ellas consiste en trasladar mediante las ecuaciones mostradas en la ecuación 17.1 el bloque desde la posición original a su nueva posición.

La segunda opción de resolver este problema consiste en recorrer el arreglo de inicio a fin incrementando de uno en uno el número del bloque que esta siendo trasladado y el número del nodo al cual debe ser trasladado.

En forma paralela se planteó el nuevo diagrama de estados del sistema en la figura 18.2:

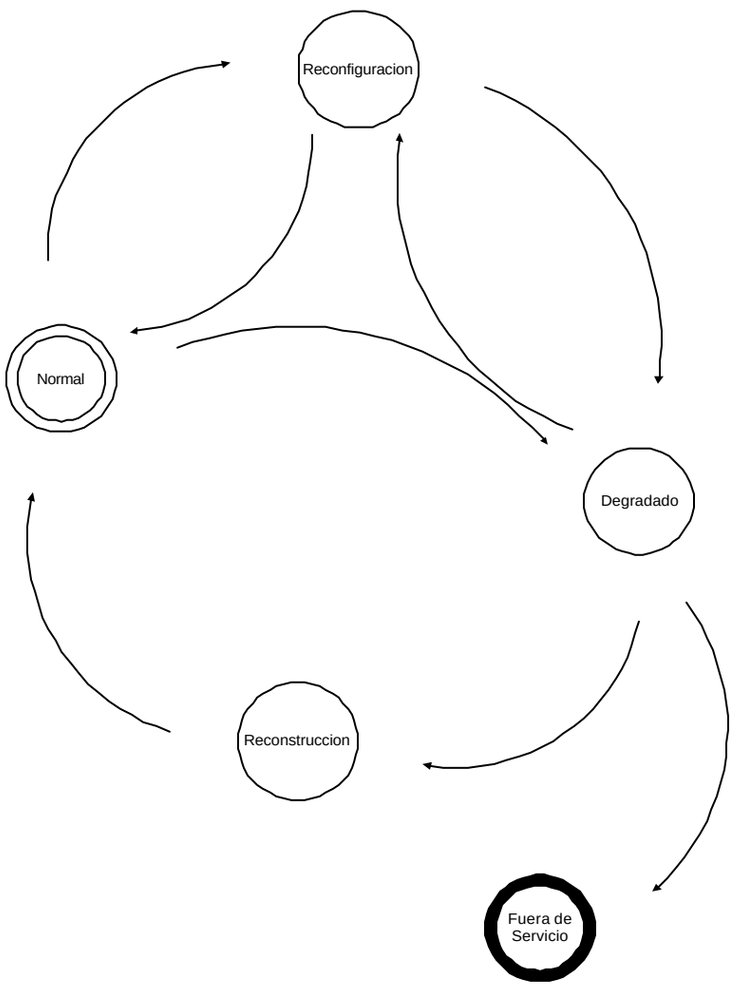

Figura 18.2 Estados de un sistema de archivos distribuido y paralelo con tolerancia a fallos y reconfiguración dinámica.

Este diagrama muestra en la parte superior el nuevo estado que tendrá el sistema denominado “Reconfiguración” cuyo propósito es indicar a cada uno de los componentes que el sistema se esta reconfigurando con la finalidad de incorporar los nuevos nodos. A su vez, este estado de reconfiguración podrá estar trabajando tanto en modo normal como en modo degradado, sin embargo en esta tesis solo se ha trabajado con el modo Normal.

Independiente del modelo a implementar, existe un problema que debe solucionarse que tiene que ver con el bloqueo de los bloques. Existen dos alternativas:

1) **Bloquear el acceso a bloques específicos**:

   Esta opción significa que solo se bloquearan los bloques que estén siendo relocalizados en ese momento lo cual trae como ventaja la minimización de la posibilidad que el bloque que se está solicitando desde los trabajadores se encuentre no disponible en ese instante, sin embargo esto conlleva a un aumento en el número de operaciones que debe realizarse y con esto el tiempo que demora la reconfiguración. Por ejemplo, para mostrar esto de una manera mas sencilla se puede llevar esta situación al extremo, es decir, se puede pensar que el proceso reconfigurador solo irá tomando un bloque y lo llevará a su nueva posición, luego tomará el siguiente y así sucesivamente. De lo anterior se puede ver que cada vez que se relocalize un bloque será necesario calcular nuevamente la paridad tanto de la franja de origen como de la franja de destino, lo que implica un considerable aumento en el número de cálculos necesarios.

2) **Bloquear el acceso a franjas específicas**.

   Esta opción propone que el bloqueo de bloques abarque un número determinado de franjas completas, produciéndose así un aumento en la probabilidad de

encontrar en un determinado momento bloqueado el bloque que se está solictando desde los trabajadores ya sea para lectura o escritura. Por otro lado, la ventaja de este método se encuentra en que se reduce la cantidad de veces que debe calcularse la paridad a solo una vez por franja, lo que en comparación con la opción anterior en la cual era necesario calcular la paridad una vez por cada bloque de la franja lleva a un menor tiempo de procesamiento.

En base a lo explicado en los puntos anteriores, se decidió utilizar la segunda alternativa puesto que aunque se aumenta la probabilidad de que encontrar bloqueado un bloque solicitado por un trabajador, esta se puede seguir considerando despreciable debido que el número total de bloques dentro del sistema de archivos es considerablemente mayor que el número de bloques dentro de una franja de paridad.

Finalmente, para agrupar todos lo conceptos anteriormente descritos, es decir, las franjas de paridad, tamaños de las unidades de paridad, operaciones de bloqueo/desbloqueo, cálculos de paridad, etc., es necesario generar un algoritmo que permita de una manera ordenada y metódica aplicar cada una de estas ideas de una forma conjunta. Para el sistema de archivos distribuido y paralelo con redundancia de datos desarrollado en [8] se tiene el siguiente algoritmo:

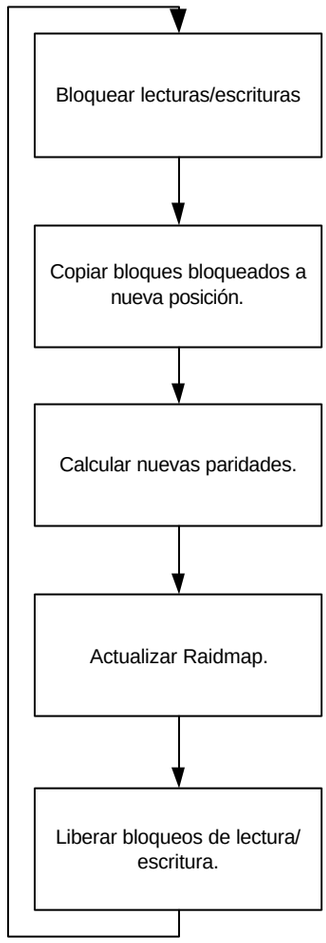

Figura 18.3 Diagrama de flujos del algoritmo de reconfiguración.

Bloquear lecturas/escrituras:

El primer paso bloquea todas las franjas de paridad que serán trasladadas a su nueva posición, así también como las franjas de paridad que se encuentran en las posiciones de destino de los bloques a trasladar.

Copiar bloques bloqueados a nueva posición.

Como segundo paso, se copian todos los bloques que pertenecen a la franja de paridad de origen hacia la franja de paridad destino.

Calcular nuevas paridades[JPG2] .

El tercer paso es calcular y escribir las unidades de paridad de las franjas de origen y de destino.

Actualizar Traductor de Bloques.

El cuarto paso es indicar en el traductor de bloques la nueva posición del puntero de reconfiguración con el fin de que este pueda redireccionar las acciones solicitadas a los bloques de manera correcta.

Liberar bloqueos de lectura/escritura.

El quinto paso dentro del proceso de reconfiguración es liberar las franjas de paridad que se encontraban bloqueadas puesto que estas ya se encuentran actualizadas y entregando un normal servicio para los trabajadores que las soliciten.

Capítulo IV

19. Evaluación

Para la evaluación del modelo propuesto, en primer lugar se generó un módulo prototipo en lenguaje de programación C que simula el trabajo realizado por un reconfigurador dinámico, es decir, recorre la matriz completa de nodos y posiciona a cada uno de ellos en su nueva ubicación. Además, se crearon los módulos que interactúan directamente con el. La descripción de cada uno de los módulos es la siguiente:

- **Módulo de inicialización de matriz inicial**.

  Este módulo se encarga de inicializar la matriz en la cual se encuentran los nodos y sus respectivos bloques. Las dimensiones de esta matriz corresponden al tamaño que tiene el sistema de archivos antes de que se ingresen los nuevos nodos.

- **Módulo de inicialización de matriz final**.

  Este módulo realiza la inicialización en cero de la matriz final en la cual se ubicarán de manera definitiva los bloques de datos. Las dimensiones de esta matriz corresponden al tamaño del sistema de archivos después de que se han ingresado los nuevos nodos.

- **Módulo generador de trabajadores**.

  Este módulo se encarga de generar de manera aleatoria un determinado número de trabajadores, quienes solicitarán lecturas y escrituras dentro del sistema de archivos en el mismo instante en que se este generando la

reconfiguración. La distribución de probabilidad utilizada en este módulo de la simulación es la distribución uniforme tanto para seleccionar el tipo de acción a realizar (lectura o escritura), como para determinar el número del bloque en el cual se debe realizar dicha acción.

- **Módulo de reconfiguración**.

  Este módulo es el encargado de realizar las acciones que redistribuyan a cada uno de los bloques en el sistema. Para cumplir con esto, el módulo debe recorrer la matriz completa desde el primer hasta el último nodo. Al realizar este recorrido, el módulo averigua en primer lugar si corresponde o no a un bloque de paridad. De no ser cierta esta afirmación, se traslada el bloque a su nueva posición, de lo contrario, se salta al siguiente bloque. Por otro lado, existe un agente que se encarga de averiguar si el bloque que corresponde ingresar en la nueva ubicación corresponde o no a un bloque de paridad, lo cual de ser cierto desencadena el cálculo de la paridad o de lo contrario continúa con la lectura de la matriz original. Por otro lado, existe el problema de que ocurriría si mientras se están trasladando los bloques de un franja de paridad, esta recibe una petición de lectura o escritura. Para resolver este problema, se emplea una técnica conocida como semáforos, la cual permite el funcionamiento exclusivo de un determinado número de operaciones, que en este caso corresponderán al traslado completo de una determinada franja de paridad correspondiente a la nueva distribución de la matriz.

- **Módulo de Traducción**.

  Este módulo se encarga de indicar a los procesos trabajadores el lugar físico exacto en el cual deben realizar la acción. En primer lugar, es necesario

hacer notar que en un instante dado dentro de la reconfiguración, la matriz en la cual se encuentran los bloques se puede considerar dividida en tres zonas. La primera zona es aquella que se encuentra en la parte superior de la matriz y que contiene los nodos que ya han sido relocalizados en su posición final. La segunda zona es la que contiene los bloques que en este momento están siendo reubicados, y por último, la tercera zona corresponde a aquella en la que se encuentran los bloques que aún se mantienen en su posición original puesto que el proceso de reconfiguración aun no pasa por ellos. En definitiva, este módulo se encarga de identificar en cual zona se encuentra el bloque que está siendo solicitado por el proceso trabajador y con esta información generar la acción en la posición exacta.

- **Módulo de semáforos**.

  Este módulo se encarga de implementar el modelo de semáforos con las operaciones P(Wait) y V(Signal) definidas por Dijkstra para secciones críticas, junto con las operaciones de inicialización y destrucción del semáforo. El semáforo tendrá siempre un valor mayor o igual a cero, de forma que si 0, cualquier operación V sobre el semáforo suspenderá el hilo que realiza la operación, hasta que el semáforo adquiera un valor superior.

- **Módulo principal**.

  El módulo principal del prototipo de simulación se encarga en primer lugar de inicializar las distintas matrices que contienen los bloques sobre los cuales se actuará, luego las imprime en pantalla y comienza la ejecución del módulo que reconfigura estas matrices, además concurrentemente inicia la ejecución de los procesos trabajadores que realizan las distintas

operaciones. Por último se despliega la matriz final en la cual se encuentran todos los bloques redistribuidos en su posición definitiva.

El propósito de este prototipo era demostrar el correcto funcionamiento del algoritmo propuesto en un ambiente no tan detallado como el simulador creado en [8]. Este objetivo se logró con éxito puesto que se puede apreciar el buen funcionamiento del proceso reconfigurador trabajando en armonía con los procesos trabajadores que en ningún momento interponen su ejecución dentro de una misma franja de paridad.

Para la evaluación definitiva del modelo visto con anterioridad, se procedió a modificar el sistema simulador de un sistema de archivos distribuido y paralelo con redundancia de datos desarrollado en [8] para incorporar los módulos generados en el prototipo. En primer lugar se detectaron cuales son las librerías sensibles a las nuevas funcionalidades para su posterior modificación. Principalmente, debieron modificarse aquellas secciones que tienen que ver con el funcionamiento de los procesos trabajadores que en este sistema se encuentran aleatoriamente repartidos con tiempo entre llegadas correspondientes a una distribución uniforme.

Las modificaciones realizadas a estos módulos corresponden a la adición de operaciones que les indiquen a los trabajadores que deben detener su ejecución hasta que el bloque sobre el cual deben actuar se encuentre liberado del proceso de reconfiguración. Además fue necesario agregar una funcionalidad, correspondiente al traductor de bloques, que le indique a los nodos cual es el verdadero bloque sobre el cual debe trabajar. Por otro lado, estas nuevas funcionalidades provocan un retardo que es detectado por el sistema y permite generar las estadísticas finales que se muestran en los gráficos de los próximos capítulos.

20. Selección del tamaño de la Unidad de Reparto.

Como se vio con anterioridad, el reparto de los datos aumenta las prestaciones de los accesos de entrada/salida en dos aspectos: en primer lugar, los accesos dirigidos a discos distintos pueden ser atendidos en paralelo; en segundo lugar, las peticiones cuya dimensión traspase el límite de la unidad de reparto de información entre discos, pueden ser atendidas coordinadamente por múltiples discos.

Un factor que determina si el aumento de las prestaciones se obtiene en mayor medida por el primer o el segundo de los aspectos citados en el párrafo anterior es el tamaño de la unidad de reparto. Cuando esta unidad es tan pequeña que cualquier petición, independientemente de su tamaño, debe acceder a todos los discos de la matriz, se dice que se trata de un reparto de grano fino. Esto permite una tasa de transferencia muy alta, pero tiene los inconvenientes de que solo se puede servir una petición de entrada/salida a la vez y que los tiempos de posicionamiento y rotación aumentan respecto a los de un disco aislado. Sin embargo, el equilibrio de la carga entre los discos se consigue de forma natural.

Un reparto de grano grueso permite que los discos cooperen para servir las peticiones grandes, mientras que las peticiones pequeñas se sirven independientemente. El equilibrado de la carga se consigue ahora por la fragmentación de los datos entre varios discos y porque el cálculo del primer disco que debe ser accedido en cada acceso tiene un componente aleatorio. En las matrices de discos con un reparto de grano grueso, el tamaño de la unidad de reparto tiene una gran influencia sobre las prestaciones. Una unidad de reparto pequeña hará que los datos que mantienen una relación lógica estén físicamente esparcidos entre muchos discos. Una unidad de reparto grande permite que estos datos estén agrupados en uno o en unos pocos discos. Por tanto, el tamaño de la

unidad de reparto determina cuantos discos se acceden en cada petición de entrada/salida. La elección del tamaño adecuado es un problema complejo, ya que debe satisfacer distintos requerimientos, algunos de ellos incompatibles como son:

- Maximizar la cantidad de información que se transfiere en cada operación de entrada/salida lógica. Dado que cada operación de entrada/salida incluye tiempos de posicionamiento y latencia rotacional de carácter mecánico en los que no se realiza trabajo útil, para contrarrestar estos tiempos muertos deben transferirse la mayor cantidad de información posible.

- Utilizar todos los discos de forma equilibrada, para evitar que mientras unos discos estén cargados, otros estén ociosos.

El conflicto entre estos objetivos se debe a que para conseguir una utilización uniforme de todos los discos debe utilizarse una unidad de reparto pequeña (de manera que cada acceso lógico implique un acceso físico a cada uno de los discos), lo cual implica transferir poca cantidad de información en cada acceso físico. Por el contrario, si se utiliza una unidad de reparto grande se maximiza la cantidad de información transferida por disco, pero se pueden dejar discos sin acceder. Chen y Patterson [47] estudiaron como resolver este conflicto en un RAID nivel 0 con 16 discos. Según sus conclusiones, la selección de la unidad de reparto adecuada tiene una fuerte dependencia del nivel de concurrencia C de la carga y una dependencia mínima del tamaño medio de las peticiones. Si se conoce la concurrencia, se puede obtener una ecuación empírica (ecuación 19.1) del tamaño de la unidad de reparto que permite alcanzar el 95% de la máxima productividad posible para una gran diversidad de tamaños medios de las peticiones. La expresión del tamaño en sectores es:

$$SxPx(C-1)+1$$

Ecuación 20.1 Tamaño de las unidades de reparto.

Donde P es el producto de prestaciones del disco que se calcula como:

$$P= (t_p+t_{LR})\lambda_T$$

Ecuación 20.2 Producto de prestaciones del disco.

Es decir, la suma de los tiempos de posicionamiento medio t_p más la latencia rotacional t_{LR}, multiplicada por la tasa de transferencia media λ_T. S es un coeficiente cuyo valor es, aproximadamente, ¼ para una matriz de 16 discos.

Si no se especifica o no se conoce la concurrencia, una buena elección para el tamaño de la unidad de reparto es:

$$ZxP$$

Ecuación 20.3 Tamaño de la unidad de reparto.

Donde Z es el coeficiente de conocimiento cero, ya que se aplica a los casos en los que no tenemos ninguna información sobre la carga. El valor de este coeficiente para una matriz de 16 discos es, aproximadamente, de 2/3.

Para el caso del RAID nivel 5, Lee y Katz [48] obtienen la siguiente ecuación para elegir el tamaño óptimo de la unidad de reparto:

$$\sqrt{\frac{P(C-1)T}{N}}$$

Ecuación 20.4 Tamaño optimo de la unidad de reparto de un RAID nivel 5.

donde T es el tamaño de las peticiones y N el número de discos. Chen y Lee [49] estudiaron el problema y llegaron a las siguientes conclusiones:

- Las lecturas en un RAID de nivel 5 se comportan igual que las lecturas y escrituras en un RAID nivel 0. Por tanto, para una carga compuesta predominantemente por lecturas, los resultados anteriores también son aplicables.

- Para cargas con un porcentaje elevado de escrituras, es mejor una unidad de reparto mas pequeña. Esto se debe a que las operaciones de escritura grandes tienen mejores prestaciones que las operaciones de lectura-modificación-escritura y reconstrucción-escritura. Una unidad de reparto más pequeña provoca más escrituras grandes y, por tanto, mejores prestaciones. La relación mostrada en los párrafos anteriores con un valor de S=1/120 es una buena elección para una matriz con 17 discos.

- Tanto para lecturas como para escrituras, el nivel de concurrencia es la característica más importante de la carga. El conocimiento de esta característica permite realizar una selección de la unidad de reparto casi óptima.

- En ausencia de información sobre las características de la carga soportada por un RAID de nivel 5, Chen y Lee recomiendan una unidad de reparto de tamaño P/2, independientemente del número de discos.

En definitiva, se ha optado trabajar con los siguientes tamaños de unidades de reparto:

Tamaño Gsuperstripe:

Este es el tamaño de la unidad de reparto para el gestor de vraids, el cual se ha fijado en 4KB para las cargas OLPT y de 256KB para las cargas Científicas.

Tamaño Supertripe:

Esto corresponde al tamaño de la unidad de reparto de las unidades VRAID y se ha fijado en 4KB para las cargas OLPT y de 64KB para las cargas Científicas.

Tamaño Stripe:

Esta es unidad de reparto de los RAID que pertenecen a cada uno de los nodos del sistema. Los valores que se han fijado en este caso son de 4KB para las cargas OLPT y de 4KB para las cargas de tipo Científica.

El criterio utilizado para seleccionar los distintos tamaños de las unidades de reparto fue el hecho que es necesario comparar los resultados de este estudio con otros experimentos anteriores realizados en [8] y en [9], estudios en los que se utilizaron los valores anteriormente mencionados. Sin embargo es bueno mencionar que los valores

seleccionados en esos estudios garantizaban que la división de la petición del cliente permitía el máximo del paralelismo dado en el sistema[9].

21. Selección del tipo de Carga de Trabajo.

Una correcta caracterización de la carga es fundamental en cualquier estudio de prestaciones. La carga de entrada/salida presenta características muy diversas dependiendo del tipo de sistema y su aplicación. Los principales parámetros para diferenciar cada tipo de carga son:

- Datos accedidos (GB)
- Porcentaje de datos accedidos solo para lectura
- Tamaño medio de las lecturas (KB)
- Tamaño medio de las escrituras (KB)
- Ratio Lecturas/Escrituras
- Tasa de accesos máxima (accesos/segundo)
- Tasa de accesos máxima (MB/segundo)
- Número de archivos en el sistema.

Los tipos de carga de entrada/salidas mas frecuentes son:

- Las cargas de los sistemas transaccionales se caracteriza por un gran número de procesos independientes que leen y escriben datos de forma concurrente y en pequeñas cantidades. En muchos de estos sistemas, además la carga provoca un pequeño porcentaje de accesos de mayor tamaño. Normalmente, en los sistemas transaccionales, el índice de prestaciones transacciones por segundo es preferible al tiempo de servicio de una transacción particular.

- Sobre la carga de los discos en las estaciones de trabajo de propósito general (oficina, ingeniería u otros) se han realizado diversos estudios. En [25][26] se mide la carga de varias estaciones Unix aisladas Baker en [27] estudia un sistema Unix distribuido (Sprite). Los autores de [28] analizan los patrones de acceso a disco en entornos de computación comerciales sobre un sistema VAX/VMS.

  Estos trabajos identifican ciertas características comunes a la carga que soportan los sistemas de archivos monoprocesadores en entornos de propósito general:

  - Los archivos son pequeños (unos pocos Kilobytes).
  - Se acceden en peticiones pequeñas.
  - Se acceden completa y secuencialmente, es decir, los bytes se leen en orden desde el principio hasta el final del archivo.

Los estudios citados fueron realizados desde el nivel del sistema de archivos del sistema operativo. La existencia de un caché de archivos impide que muchos accesos de entrada/salida alcancen los discos, por lo que estos estudios no caracterizan correctamente lo que ocurre a nivel de disco. Asimismo ignoran el tráfico generado por el propio sistema de archivos y por los mecanismos de memoria virtual.

Ruemmler y Wilkins [29] realizan un estudio de caracterización de la carga soportada por los discos bajo un entorno Unix. Los sistemas analizados durante un periodo de dos meses estaban dedicados a funciones diferentes. Uno era un sistema interactivo dedicado a tareas de investigación y oficina. Otro actuaba

como servidor de archivos. El tercero era una estación de trabajo de uso personal. Las trazas fueron obtenidas mediante una instrumentación instalada en el nivel de núcleo del sistema operativo, completamente transparente a los usuarios. Las trazas se registraron en un disco dedicado para evitar la perturbación del sistema medido.

A grandes rasgos, las conclusiones de su trabajo fueron las siguientes:

- Entre el 25% y el 50% de los accesos son asíncronos, es decir, no existe un proceso asociado que espere la finalización de la operación sino que esta se ejecuta en el fondo.

- Solo entre el 13 y el 41% de los accesos corresponden a datos de usuario, mientras que del 67 al 78% de las escrituras corresponden a metadatos metadatos, intercambio (del inglés swapping), ejecución de programas, etc.

- Una proporción elevada de la actividad de entrada/salida se produce en ráfagas y no de manera aislada.

- El tiempo medio de espera en cola observado por un acceso que llega al subsistema de entrada/salida está entre 1.7 y 8.9 ms (de 1.2 a 1.9ms para las lecturas y de 2.0 a 14.8ms para las escrituras). El percentil 95 de la longitud de las colas es de 89 accesos.

- El 57% de los accesos son escrituras.

- Aunque estudios realizados al nivel del sistema de archivos detectaban una secuencialidad alta [26], esta solo llega parcialmente al nivel de disco, como lo

demuestra el hecho de que solo entre el 8 y el 12% de las escrituras y entre el 10 y el 18% de las lecturas son lógicamente secuenciales.

- Del 10 al 18% de las escrituras son reescrituras del último bloque escrito (generalmente metadatos).

- Las características de la entrada/salida en aplicaciones científicas han sido analizadas en [30] (caracterización informal de aplicaciones grand-challenge) y en [31] (estudia la tasa de entrada/salida de aplicaciones intensivas sobre Cray). Destacan las siguientes características comunes a la carga que soportan los sistemas de archivos de computadores vectoriales:

  - Los archivos son grandes (muchos Megabytes o Gigabytes)
  - Se acceden en peticiones grandes
  - Se acceden completa y secuencialmente.

- Las características de los accesos a disco de las aplicaciones científicas paralelas han sido objeto de gran interés en los últimos años. La entrada/salida representa el mayor obstáculo que impide el uso eficiente de los sistemas de computación en la escala de los gigaFLOPS actuales y los teraFLOPS futuros. Entre los proyectos puestos en marcha para afrontar esta caracterización destacaremos los siguientes. Dentro de los trabajos desarrollados en el proyecto CHARISMA (del inglés CHARacterize I/O in Scientific Multiprocessor Applications) del Darthmouth Collage [32] se caracteriza la carga soportada por el sistema de archivos de dos computadores paralelos con memoria distribuida, INTEL iPSC/860 con el sistema de archivos CFS (del inglés Concurrent File System) y CM-5 de

Thinking Machines con el sistema de archivos SFS (del inglés Scalable File System). Se obtiene el siguiente perfil:

- Los archivos son grandes
- Se acceden en peticiones pequeñas (64 – 256 bytes)
- Se realizan accesos secuenciales no consecutivos.
- La mayoría de las aplicaciones paralelas utilizan muchos ficheros diferentes en una misma ejecución.
- Hay mucha compartición de archivos entre distintos procesadores pero poca entre diferentes tareas.

En esta línea es igualmente destacable la Scalable I/O Initiative (SIO), un proyecto en el que participan diversas universidades, laboratorios de investigación y fabricantes de sistemas paralelos. El objetivo básico de este proyecto es caracterizar el patrón de accesos de entrada/salida en sistemas paralelos y aprovechar este conocimiento para guiar el desarrollo del software del sistema relacionado con la entrada/salida: compiladores, librerías dinámicas, sistemas de archivos paralelos, interfaces de red de altas prestaciones y servicios del sistema operativo. Todos estos componentes deben ofrecer la velocidad suficiente para evitar convertirse en cuellos de botella del sistema. Por último, cabe destacar el proyecto ARPA Input/Output Characterization que se está desarrollando en la Universidad de Illinois en urbana-Champaign y que persigue objetivos similares a los anteriores.

Tomando en consideración los estudios realizados por empresas y universidades se ha considerado trabajar con las cargas mas representativos de sistemas de archivos

potencialmente usuarios directos de un sistema de archivos distribuido y paralelo con redundancia de datos que serían las cargas de tipo transaccional y las de tipo científica.

Carga de Trabajo OLPT

Las principales características de las cargas transaccionales con las que se trabajaron en este estudio son:

Distribución Probabilística	: Uniforme
Lecturas de 4 KB	: 8%:
Escrituras de 4KB	: 88%
Lecturas de 24KB	: 2%
Escrituras de 24KB	: 2%

Tabla 21.1 Características de una carga de trabajo OLPT.

Carga de Trabajo Científica

Para la carga de tipo científica se tiene el siguiente perfil:

Distribución Probabilística	: Uniforme
Accesos	: 50% a 10 archivos de 5MB 50% acceso secuencial a 1 archivo de 100MB
Lecturas a los archivos de 5MB	: 10% lecturas de 5MB
Escrituras a los archivos de 5MB	: 90% escrituras de 5MB
Lecturas al archivo de 100MB	: 90% lecturas de 1MB
Lecturas al archivo de 100MB	: 10% escrituras de 1MB

Tabla 21.2 Características de una carga de trabajo científica.

Al igual que en la selección del tamaño de la unidad de reparto y con el fin de poder realizar una comparación posterior, se ha decidido continuar con la línea planteada en los estudios realizados en [8] y en [9] y se ha optado por generar las pruebas en bases a las cargas de tipo OLPT y las cargas de tipo Científica. Además, se han realizado las pruebas utilizando tiempos entre peticiones generadas por las cargas de trabajo distribuidos de manera uniforme con tiempo de 250, 500, 750, 1000 t 1250 ms. en promedio.

22.Selección de la configuración del VRAID.

En cuanto a la configuración del VRAID, las pruebas se han realizado tomando en cuenta que se debe trabajar al igual que en [9], por lo que para las primeras pruebas que tienen que ver con la medición de la productividad en el sistema, se han utilizados configuraciones de dos sistemas VRAID con 3, 5 y 9 nodos tanto para las cargas OLPT como para las de tipo científica. Luego, para las pruebas que tienen que ver con la medición de los tiempos de lectura y escritura, se utilizaron configuraciones de dos VRAID de 5 nodos cada uno.

Conviene decir que para todas las configuraciones generadas para las distintas pruebas se han distribuido los nodos de los VRAID en una red de topología bus de 100Mbps y además cada uno de estos contenía una matriz de discos RAID.

23. Selección de la configuración de los RAID.

Para la selección de las configuraciones de las matrices de discos RAID contenidas en cada uno de los nodos, se utilizó el mismo criterio que para las configuraciones anteriores, es decir aquellas que permitiesen ser comparadas con los trabajos realizados en [8] y en [9].

En definitiva, cada una de las matrices RAID estaba compuesta por 5 discos y con una configuración de tipo RAID5 con simetría izquierda, además los discos estaban conectados con un bus de tipo SCSI2 con un ancho de banda de 20 MB/s y concurrencia activada. Resta decir que los discos simulaban discos de marca IBM, modelo ULTRA2 con 3534 cilindros, 21 pistas por cilindro, 110 sectores por pista, 10000 revoluciones por minuto, tiempo mínimo de búsqueda de 2.4 ms, tiempo promedio de búsqueda de 9 ms, tiempo máximo de búsqueda de 19 ms, caché de 2048 KB y 4GB de tamaño.

24. Resultados Carga OLPT

Después de ejecutar el sistema simulador con las modificaciones necesarias, se midieron los tiempos de respuesta del sistema para peticiones de lecturas y escrituras encontrándose lo siguiente:

- Tiempos de Lectura (figura 24.1):

  En este caso, se graficaron los valores obtenidos para peticiones cuyos tiempos entre llegadas se distribuyen de manera uniforme entre 0 y 250, 500, 750, 1000 y 1250 milisegundos. Se puede apreciar que los valores obtenidos tanto para el modo normal (puntos marcados con cuadrados) como para el modo de reconfiguración (puntos marcados con triángulos), no existe una gran diferencia. Esto se puede explicar debido a que el proceso de reconfiguración adiciona tan solo una pequeña cantidad de proceso que es despreciable si se compara con los tiempos totales del sistema.

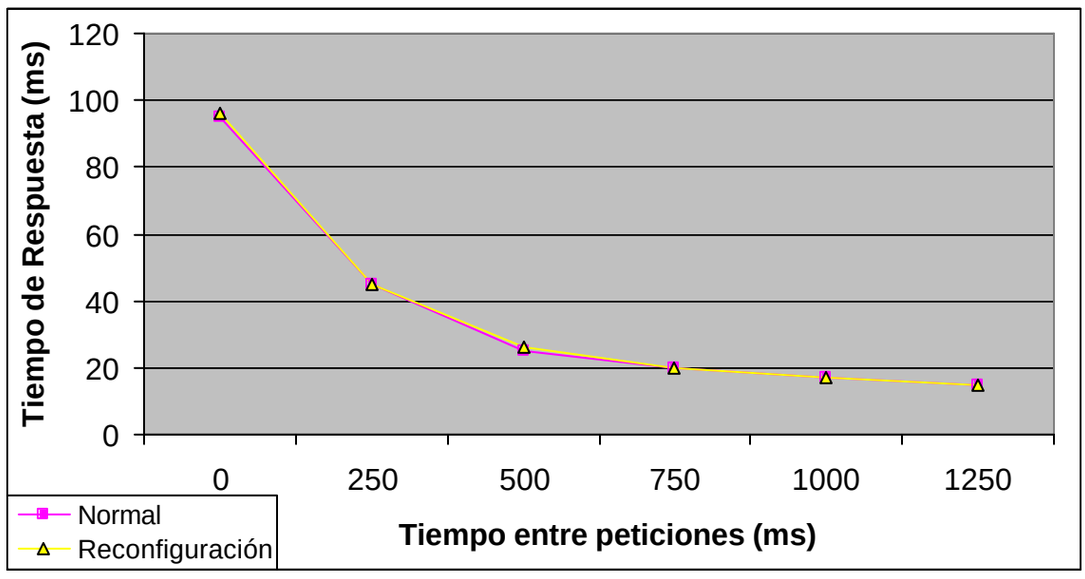

Figura 24.1 Gráfico de tiempos de lecturas del sistema para el modo normal y en reconfiguración con cargas de tipo OLPT.

- Tiempos de Escritura (figura 24.2):

  Al igual que el gráfico anterior, este presenta los valores obtenidos para el sistema en los estados normal y en reconfiguración para los tiempos entre llegadas de las peticiones de lectura/escritura distribuidas de manera uniforme entre 0, 250, 500, 750, 1000 y 1250 milisegundos. De la misma manera, se aprecia que el tiempo de respuesta del modo reconfiguración cuando los tiempos entre llegadas de las peticiones es pequeño es levemente superior a los tiempos del estado normal, tendiendo estos tiempos de respuesta a igualarse a medida que se incrementan los tiempos entre llegadas. Si se toma en cuenta que todo experimento considera un rango de incertidumbre, puede considerarse que la diferencia de los primeros valores es prácticamente nula.

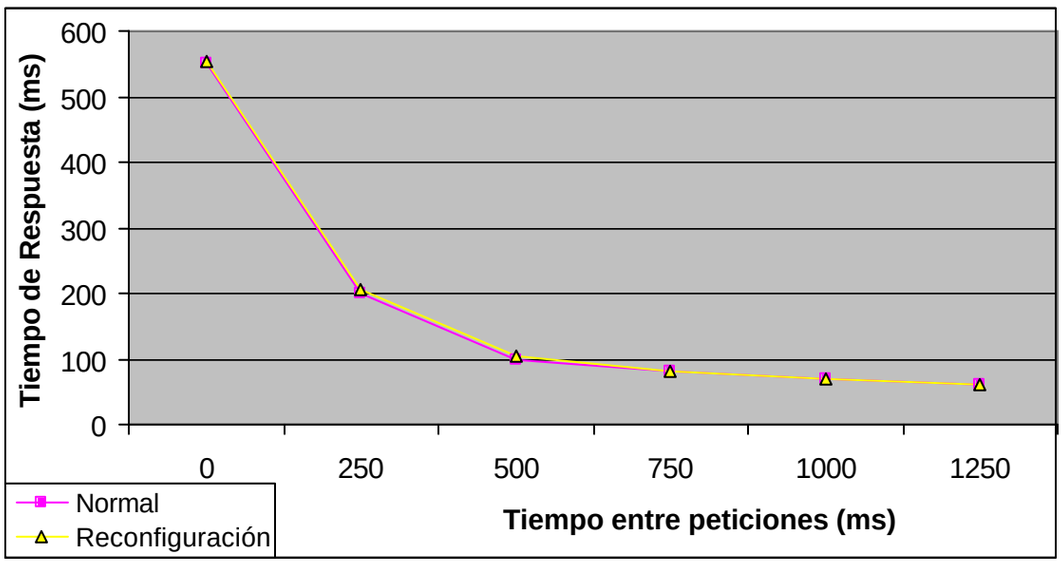

Figura 24.2 Gráfico de tiempos de escrituras del sistema para el modo normal y en reconfiguración con cargas de tipo OLPT.

25. Resultados Carga Científica

Para el caso de las cargas de tipo científica, se realizo el mismo tipo de pruebas que para las cargas de tipo OLPT. Los resultados obtenidos para los distintos tipos de peticiones son:

- Tiempos de Lectura (figura 25.1):

  Los tiempos de lectura de las cargas de tipo científica se realizaron con los mismos valores que para las carga de tipo OLPT. En este gráfico se puede apreciar que los tiempo de lectura para el tipo de cargas de tipo científica es mayor que los tiempos de lectura para cargos de tipo OLPT tendiendo a igualarse a medida que aumentan los tiempos entre peticiones. Sin embargo, al igual que en las cargas de tipo OLPT, la diferencia de tiempos entre las modalidades normal y reconfiguración son prácticamente los mismos.

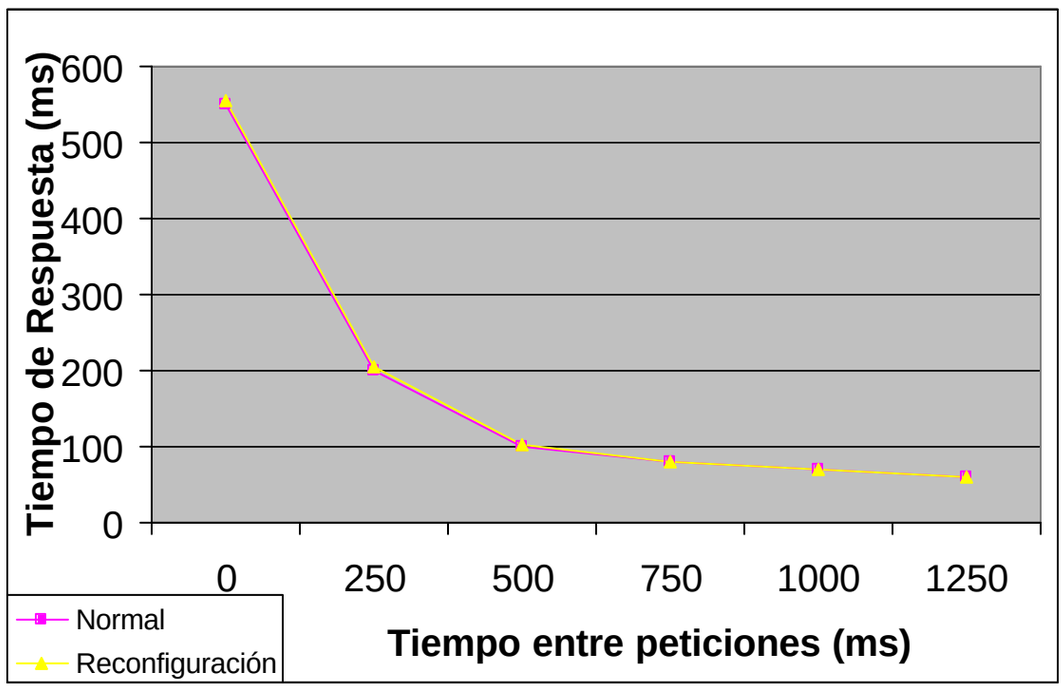

Figura 25.1 Gráfico de tiempos de lecturas del sistema para el modo normal y en reconfiguración con cargas de tipo científica.

- Tiempos de escritura (figura 25.2):

  Se puede apreciar al igual que el gráfico anterior que los tiempos de respuesta son mejores para las cargas de tipo OLPT que para las de tipo científica sobre todo cuando los tiempos entre llegadas de las peticiones son bajos. Además también se puede ver que de la misma manera que los gráficos anteriores, el tiempo de respuesta del sistema para los modos normal y reconfiguración es bastante similar.

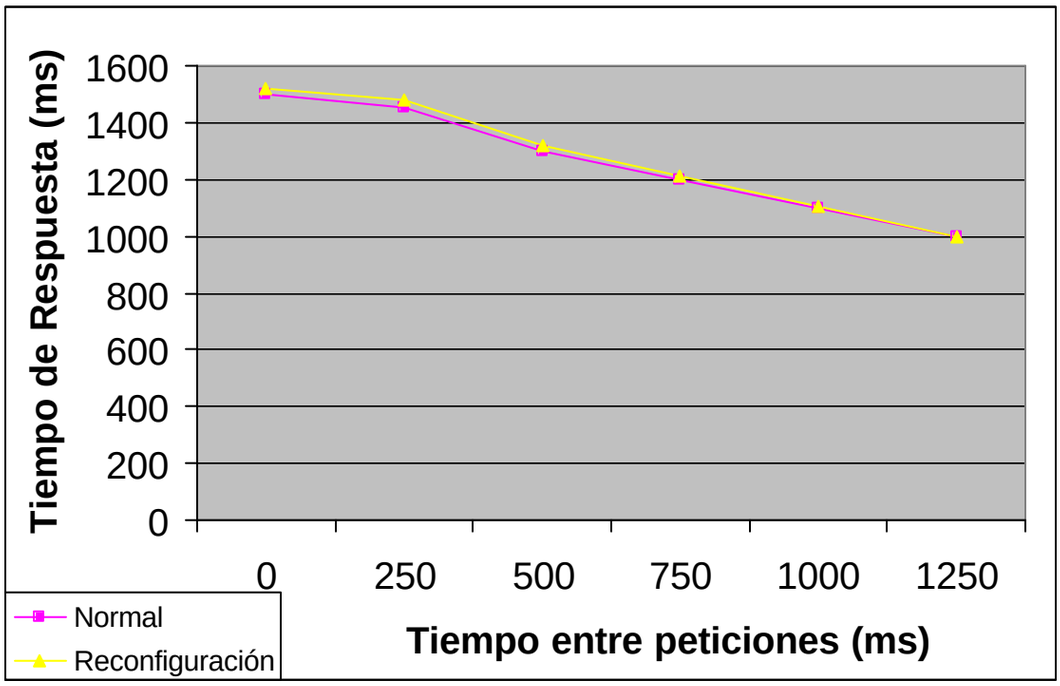

Figura 25.2 Gráfico de tiempos de escrituras del sistema para el modo normal y en reconfiguración con cargas de tipo científica.

Capítulo V

26. Conclusiones

Del trabajo desarrollado en esta tesis se puede concluir lo siguiente:

1. A medida que el tiempo entre peticiones aumenta, el sistema tiende a comportarse de la misma manera en modalidades normal y de reconfiguración

2. El modelo seleccionado para realizar las pruebas es una alternativa viable para ser implementada en la realidad puesto que cumple con el objetivo de reconfigurar el sistema desde un estado inicial de desbalance a uno final en el que los datos se encuentran repartidos de manera homogénea.

3. Cuando el sistema se encuentra en modo de reconfiguración existe una carga de trabajo adicional que se puede considerar despreciable respecto a la carga de trabajo total cuando el sistema se encuentra en modo normal.

   Esto se explica por medio del siguiente análisis: En las pruebas realizadas se utilizó un sistema compuesto por 50 discos de 4GB cada uno por lo tanto el tamaño total del sistema es de 208,98 GB. El tamaño de cada franja de paridad es de 640KB, existiendo por lo tanto 326541 franjas. El proceso de reconfiguración bloquea dos franjas, una para la lectura de la franja que se esta trasladando y otra que es la franja que se esta escribiendo. Esto implica que la probabilidad de que una petición se efectúe sobre una franja bloqueada es de :

$$P = \frac{2}{326541} = 6{,}12 * 10^{6}$$

Ecuación 26.1 Probabilidad de que una petición se realice sobre una franja bloqueada.

De aquí se puede apreciar que el valor de P es extremadamente pequeño, sin embargo si se considera que el tiempo entre peticiones es de 0, y el número de peticiones que se ejecutan es de 150 como en el caso de las pruebas realizadas, entonces suponiendo que todas las peticiones han bloqueado una franja la ultima petición tiene la siguiente probabilidad de encontrar una franja ocupada:

$$P = \frac{152}{326541} = 0{,}0004593$$

Ecuación 26.2 Probabilidad de que la ultima petición se realice sobre una franja bloqueada.

De lo anterior se aprecia que las probabilidades de que una petición encuentre ocupada la franja sobre la que realizará la operación es baja. De todas maneras es bueno hacer notar que estos cálculos suponen que las peticiones son independientes lo que en la realidad no es así puesto que las peticiones tanto de lectura y escritura tienden a agruparse sobre franjas contiguas.

En cuanto a las mejoras que se puede realizar en futuros trabajos en cuanto a la misma nueva funcionalidad desarrollada en esta tesis se encuentran:

- Implementar el segundo modelo de reconfiguración dinámica presentado en la figura 18.1 para comparar resultados.
- Implementar el modelo desarrollado en esta tesis para los estados degradado y reconstrucción, puesto que aquí solo se estudió el estado normal. Para esto, a continuación se presenta un algoritmo de reconfiguración dinámica que puede utilizarse (figura 26.1):

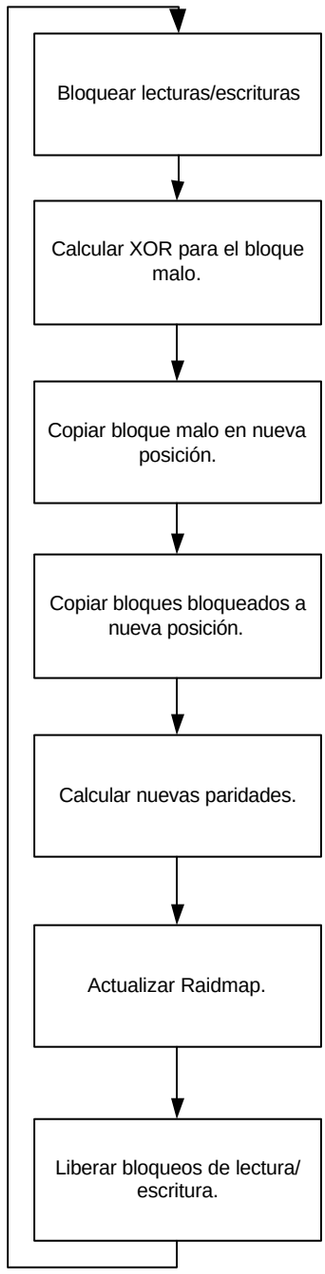

Figura 26.1 Algoritmo de reconfiguración para el sistema en modo degradado.

Con respecto al algoritmo de la figura 18.3, este nuevo procedimiento incluye módulos que deben realizar operaciones adicionales debido a la necesidad de calcular los nodos que no se encuentran disponibles de manera directa.

Capítulo VI

27. Bibliografía

[1] E. Levy and A. Silberschatz. Distributed File Systems. *ACM Computer Surveys*, 22(4):321-374, December 1990.

[2] L. Svodoba. File Servers for Network-Based Distributed Systems. *ACM Computing Surveys*, 16(4):354-398, December 1984.

[3] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *In Proceedings of ACM SIGMOD* pages 109-116. ACM, June 1988.

[4] D. Kotz and N. Ñieuwejaar. Characterisitcs of a Production Parallel Scientific Workload. Technical Report PCS-TR94-211, Dep. of Computer Science, Darmouth College, 1994.

[5] D. Kotz and N. Ñieuwejaar. File System Workload on a Scientific Multiprocessor. *IEEE Parallel and Distributed Technology. Systems and Aplications*, pages 134-154, Spring 1995

[6] M. Ñelson, B. Welch, and J. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1):134-154, February 1988.

[7] Real Academia Española. Diccionario de la Lengua Española. Pagina 822. Junio de 1992.

[8] Raimundo Vega, Francisco Rosales. Disponibilidad de Datos en Sistemas de Ficheros Paralelos. 1998.

[9] Cristian Romo Tregear, Mejora de la disponibilidad de datos de un subsistema de e/s paralelo mediante agrupación de componentes.

[10] David Bustard, John Elder, Jim Welsh. Concurrent Program Structures. Discrete Event Simulation, paginas 154-158,171-172. Prentice Hall International, Series in Computer Science.1988.

[11] Pankaj Jalote. Fault Tolerance in Distributed Systems. Stable Storage. Paginas 105-107. Prentice Hall. 1994.

[12] Gibson, G., Redundant Disk Arrays: Reliable, Parallel Secondary Storage, MIT Press, 1992.

[13] IBM Corporation, IBM 3380 Direct Access Storage Introduction, Manual GC26-4491-0, 1987.

[14] Wood, C. and Hodges, P., "DASD Trends: Cost, Performance, and Form Factor",Proceedings of the IEEE, Vol. 81, No. 4, 1993, pp. 573-585.

[15] Seagate Corporation, Disk Drive Model 15 product information.

[16] Katzman, J. "System Architecture for Nonstop Computing," Proceedings of the Computer Society International Conference (COMPCON 77), 1977.

[17] Gray, G., Horst, B. and Walker, M., "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput," Proceedings of the Conference on Very Large Data Bases, 1990, pp. 148-160.

[18] Stodolsky, D., Gibson, G., Courtright, W.V., and Holland, M., "A Redundant Disk Array Architecture for Efficient Small Writes," Technical Report No. CMUCS- 94-170, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890, July 1994.

[19] Cao P., Lim, S.B., Venkataraman, S., and Wilkes, J., “The TickerTAIP Parallel RAID Architecture,” Proceedings of the International Symposium of Computer Architecture (ISCA), 1993, pp. 52-63.

[20] Chen, P. and Patterson, D., “Maximizing Performance in a Striped Disk Array,” Proceedings of International Symposium on Computer Architecture, 1990, pp. 322-331.

[21] RAID Advisory Board, The RAIDBook: A Source Book for RAID Technology, 5th Ed., St. Peter, Minnesota, 1996.

[22] Peterson, W. and Weldon Jr., E., Error-Correcting Codes, second edition, MIT Press, 1972.

[23] Blaum, M., Brady, J., Bruck, J., and Menon, J., “Evenodd: An Optimal Scheme for Tolerating Double Disk Failures in RAID Architectures,” Proceedings of the International Symposium of Computer Architecture (ISCA), 1994, pp. 245-54.

[24] Lee, E. and Katz, R., “Performance Consequences of Parity Placement in Disk Arrays,” Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 1991, pp. 190-199.

[25] R. Floyd, “Short-term reference patterns in a Unix environment” Tech.Rep. 177, Dpt of Computer Science. University of Rochester, Mar 1986.

[26] J. Ousterhout, H Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, “A trace driven analysis of the Unix 4.2 BSD file system” in Proceedings of the Tenth ACM Symposium on Operating Systems Principles, pp. 15-24, Dec. 1985.

[27] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout, “Measurements of a distributed file system” in Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles, pp 198-212, 1991.

[28] K. K. Ramakrishnan, P. Piswas, and R. Karedla, “Analysis of the file i/o traces in commercial computing environments”, in Proceedings of the ACM SIGMETRICS and PERFORMANCE’92, pp 78-90, 1992.

[29] C. Ruemmler and J. Wilkes, “Unix disk access patterns” in USENIX Winter 1993 Technical Conference Proceedings, Jan. 1993.

[30] J.M. del Rosario and A.N. Choudhary, “High performance I/O for massively parallel computers: Problems and prospects” IEEE Computer, vol 27 pp 59-68, Dec. 1994

[31] B.K. Pasquale and G.G. Polizos, “A case study of a scientific application I/O behavior” in Proceedings of the International Workshop on Modeling Analysis and Simulation of Computer and Telecomunication Systems, pp. 101-106, 1994.

[32] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis and M. Best, “File access characteristics of parallel scientific workloads” Tech. Rep. PCS-TR95-263, Dartmouth College, Aug. 1995.

Apéndice A

```
#include<stdio.h>

#define mat_inicial_filas 50
#define mat_inicial_columnas 15
#define mat_final_filas 55
#define mat_final_columnas 16

int matriz_inicial[mat_inicial_columnas][mat_inicial_filas];
int matriz_final[mat_final_columnas][mat_final_filas];
int main(int argc,char *argv[])
        {
        int i;
        int j;
        int suma=0;
/* declaracion de variables para el calculo del nodo */
        int i_proy;
        int j_proy;
        int bloque;
        int res1;
        int res2;
        int n=mat_final_columnas-1;

/*---------------------- LLENADO DE MATRIZ INICIAL --------------------*/
        for (j=0;j<mat_inicial_filas;j++)
        {
          for (i=0;i<mat_inicial_columnas;i++)
                {
                if (i==((mat_inicial_columnas-1)-(j%(mat_inicial_columnas))))
                        {
                        matriz_inicial[i][j]=-1;
                        printf("  P|");
                        }
                else
                        {
                        matriz_inicial[i][j]=suma;
                        printf("%*d|",3,matriz_inicial[i][j]);
                        suma++;
                        };
                };
          printf("\n");
        };

/*------------------- INICIALIZACION DE MATRIZ FINAL ------------------*/
        for (j=0;j<mat_final_filas;j++)
          for (i=0;i<mat_final_columnas;i++)
                {
                matriz_final[i][j]=-1;
                };

/* ----------- PROYECCION DE MATRIZ INICIAL EN MATRIZ FINAL ----------- */
        for (j=0;j<mat_inicial_filas;j++)
          for (i=0;i<mat_inicial_columnas;i++)
                {
                if (matriz_inicial[i][j]!=-1)
                  {
                  bloque=matriz_inicial[i][j];
```

```
                        // CALCULO DE LA FILA PROYECTADA (FRANJA DE PARIDAD)
                        j_proy=(int)(bloque/n);
                        // CALCULO DE LA COLUMNA PROYECTADA (NODO)
                        res1=(n+1) * (int)(j_proy/(n+1));
                        res2=(int)(bloque/(((n-1)*(j_proy+1))+1+res1));
                        i_proy=(bloque%n)+res2;
                        matriz_final[i_proy][j_proy]=bloque;
                        };
                    };
/* ---------------------- DESPLIEGUE DE MATRIZ FINAL ------------------- */
        for (j=0;j<mat_final_filas;j++)
          {
          for (i=0;i<mat_final_columnas;i++)
                {
                if (matriz_final[i][j]==-1)
                        printf ("  P|");
                else
                        printf ("%*d|",3,matriz_final[i][j]);
                };
          printf("\n");
          };
        return(0);
};
```

Apéndice B

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

#include <pthread.h>

#define mat_inicial_filas 500
#define mat_inicial_columnas 13
#define mat_final_filas 500
#define mat_final_columnas 16

/*************************************************************************
*******
********************  Implementacion de Semaforos
**************************
*************************************************************************
*******/
/* Estructura de datos del semáforo */
typedef struct {
    pthread_mutex_t lock;  /* Acceso exclusivo a la estructura */
    pthread_cond_t  no_cero;       /* Condición de semáforo no cero */
    unsigned int    contador; /* Valor del semáforo */
    unsigned int    en_espera;/* No. de hilos en espera */
} sem_t;

sem_t mutex;

/* Inicializa la estructura del semáforo con el valor indicado */
int sem_init(sem_t *s, unsigned int valor) {
    /* Inicializa el mútex y la variable de condición asociadas */
    pthread_mutex_init(&s->lock, NULL);
    pthread_cond_init (&s->no_cero, NULL);
    s->contador = valor;
    s->en_espera = 0;
    return (0);
}

/* Destruye un semáforo que ya no es necesario */
int sem_destroy(sem_t *s) {
    pthread_mutex_destroy(&s->lock);        /* Destruye el mútex */
    pthread_cond_destroy(&s->no_cero);     /* Destruye la var.cond.*/
    return (0);
}

/* Operación P o WAIT de Dijkstra sobre el semáforo */
int sem_wait(sem_t *s) {
    pthread_mutex_lock(&s->lock);  /* Acceso exclusivo a datos sem.*/
    while (s->contador == 0) {    /* Esperar hasta que semáforo>0 */
        s->en_espera++;                    /* contar al hilo en espera */
```

```
      pthread_cond_wait(&s->no_cero, &s->lock); /* Espera condic. */
      s->en_espera --;              /* Descontar al hilo en espera */
   }
   s->contador--;        /* Decrementa el valor del semáforo */
   pthread_mutex_unlock(&s->lock);       /* Libera el mútex */
   return (0);
}

/* Operación V o SIGNAL de Dijkstra sobre el semáforo */
int sem_signal(sem_t *s) {
   pthread_mutex_lock(&s->lock);  /* Acceso exclusivo a datos sem.*/
   if (s->en_espera)                    /* Si existen hilos en espera */
      pthread_cond_signal(&s->no_cero);      /* Se despierta un hilo */
   s->contador++;               /* Se incrementa el semáforo */
   pthread_mutex_unlock(&s->lock);       /* Libera el mútex */
   return (0);
}

/*************************************************************************
*******
******************* FIN Implementacion de Semaforos
************************
*************************************************************************
*******/

int matriz_inicial[mat_inicial_columnas][mat_inicial_filas];
int matriz_final[mat_final_columnas][mat_final_filas];

void inicializacion_matriz_inicial()
{
    int i;
    int j;
    int suma=0;
    /*---------------------- LLENADO DE MATRIZ INICIAL ------------------*/
    for (j=0;j<mat_inicial_filas;j++)
    {
      for (i=0;i<mat_inicial_columnas;i++)
            {
            if (i==((mat_inicial_columnas-1)-(j%(mat_inicial_columnas))))
                  {
                  matriz_inicial[i][j]=-1;
                  printf("   P|");
                  }
            else
                  {
                  matriz_inicial[i][j]=suma;
                  printf("%*d|",4,matriz_inicial[i][j]);
                  suma++;
                  };
            };
      printf("\n");
    };
};
```

```
void inicializacion_matriz_final()
{
    int i;
    int j;
    /*------------------- INICIALIZACION DE MATRIZ FINAL -----------------*/
    for (j=0;j<mat_final_filas;j++)
      for (i=0;i<mat_final_columnas;i++)
            {
            matriz_final[i][j]=-1;
            };
};

void despliegue_matriz_final()
{
    int i;
    int j;
/* ---------------------- DESPLIEGUE DE MATRIZ FINAL ------------------ */
    for (j=0;j<mat_final_filas;j++)
      {
      for (i=0;i<mat_final_columnas;i++)
            {
            if (matriz_final[i][j]==-1)
                  printf ("   P|");
            else
                  printf ("%*d|",4,matriz_final[i][j]);
            };
      printf("\n");
      }
}

typedef struct
{
    int accion;
    int bloque;
} argumentos;

void crea_trabajador(void *arg)
{
    argumentos param = *(argumentos *)arg;
//  printf("Espera trabajador\n");
    sem_wait(&mutex);
    if (param.accion==0)
    {
            printf("accion Leer en bloque %d\n",param.bloque);
            fflush (stdout);
    }
    else
    {
            printf("accion Escribir en bloque %d\n",param.bloque);
            fflush (stdout);
    };
//  printf("Avanza trabajador\n");
```

```
    sem_signal(&mutex);
};

void proceso_de_reconfiguracion()
{
    int i;
    int j;
    int i_nuevo=0;
    int j_nuevo=0;
    int nodo_de_paridad;

    j=0;
    while (j<mat_inicial_filas)
    {
            sem_wait(&mutex_franja);
            for (i=0;i<mat_inicial_columnas;i++)
            {
                    if (matriz_inicial[i][j]!=-1)
                    {
                            if                      ((abs(i_nuevo-mat_final_columnas)-
1)!=(j_nuevo%mat_final_columnas))
                            {

    matriz_final[i_nuevo][j_nuevo]=matriz_inicial[i][j];  // mueve el bloque al nuevo
nodo
                            }
                            else
                            {
                                    nodo_de_paridad=i_nuevo;  // almacena el nodo
donde debe ir la paridad de la franja actual
                            };

                            i_nuevo++;
                            printf("j=%d i=%d ",j,i);
                            fflush (stdout);
                            if (i_nuevo==mat_final_columnas)
                            {
                                    matriz_final[nodo_de_paridad][j_nuevo]=-1;
    // calcula la paridad en el nuevo raidmap
                                    i_nuevo=0;
                                    j_nuevo++;
                            };
                    };
            };
            printf("\n");
            sem_signal(&mutex_franja);
            j++;
    };
    printf("FIN RECONFIGURACION");
    fflush(stdout);
};
int r;
int main(int argc,char *argv[])
```

```
{
    int i,j;
    int n_trabajadores=250;
    int accion;
    int bloque;
    pthread_t agents[n_trabajadores+1];
    void* return_values[n_trabajadores+1];
    argumentos *args1;
    sem_init(&mutex_franja,1);

    inicializacion_matriz_inicial();
    inicializacion_matriz_final();
    //********** Generacion de procesos trabajadores **********
    for (i=0;i<n_trabajadores+1;i++)
    {
            accion=rand()%2;
            bloque=rand()%(mat_inicial_filas*(mat_inicial_columnas-1));
            args1[0].accion=accion;
            args1[0].bloque=bloque;
            if (i==0)

    pthread_create(&agents[i],NULL,proceso_de_reconfiguracion,NULL);
            else
                    pthread_create(&agents[i],NULL,crea_trabajador,&args1[0]);
    };

    for (i=0;i<n_trabajadores+1;i++ )
    {
            if ((r=pthread_join(agents[i],&return_values[i]))!=0)
            {
                    //printf ("Join error %d - retorno: %d\n",i,r);
            }
            else
            {
                    //printf ("Join OK %d - retorno: %d\n",i,r);
            }
    };
    sem_destroy(&mutex_franja);
    despliegue_matriz_final();
    return(0);
};
```