UNIVERSIDAD DE
CASTILLA-LA MANCHA

OPTIMIZACIÓN DE CLUSTERS COMO PLATAFORMAS MULTIMEDIA UTILIZANDO CLIENTES PREDICTIVOS MULTIHILO

TESIS DOCTORAL

TERESA OLIVARES MONTES

ALBACETE, FEBRERO DE 2003

OPTIMIZACIÓN DE CLUSTERS COMO PLATAFORMAS MULTIMEDIA UTILIZANDO CLIENTES PREDICTIVOS MULTIHILO

MEMORIA QUE PRESENTA:
TERESA OLIVARES MONTES
PARA OPTAR AL GRADO DE
DOCTOR EN INGENIERÍA INFORMÁTICA
FEBRERO 2003

DIRECTORES
FRANCISCO JOSÉ QUILES FLOR
ANTONIO GARRIDO DEL SÓLO

DEPARTAMENTO DE INFORMÁTICA
EPSA UCLM

AUTORIZACIÓN PARA LA PRESENTACIÓN DE LA TESIS DOCTORAL DE TERESA OLIVARES MONTES

En cumplimiento de lo previsto en el artículo 8.1 del R.D. 778/98 de 30 de Abril sobre procedimiento para la presentación y defensa de una tesis doctoral, así como del artículo 33 de la sección 4 de las normas reguladoras de los estudios de tercer ciclo de la Universidad de Castilla-La Mancha, D. Francisco José Quiles Flor, y D. Antonio Garrido del Sólo, como directores de la Tesis Doctoral *Optimización de Clusters como Plataformas Multimedia utilizando Clientes Predictivos Multihilo*, realizada por Dña. Teresa Olivares Montes, emitimos el siguiente informe autorizando su presentación y defensa.

En esta Tesis se plantearon, como objetivos básicos a realizar, la evaluación y optimización de clusters soportando tráfico multimedia. Del análisis inicial realizado, se detectaron una serie de problemas relacionados con la entrada/salida de datos, y en concreto, con la forma en la que se leen los datos multimedia compartidos mediante NFS (Network File System). Esto llevó a plantear una posible mejora en la operación de lectura que implicase un adelantamiento de datos a la cache de los clientes y así evitar enviar peticiones al servidor. Así, se planteó una técnica de Prefetching Multihilo a través de ficheros. Esta técnica se ha implementado en el núcleo del sistema operativo y se ha evaluado en un cluster multimedia real, obteniendo importantes reducciones en los tiempos de lectura, para distintos patrones de carga.

Por lo tanto, los objetivos de la tesis se han cumplido perfectamente y, emitimos el presente informe de conformidad para la presentación y defensa de la mencionada Tesis Doctoral.

En Albacete, a 25 de Febrero de 2003

Los Directores de la Tesis

Fdo. Dr. D. Francisco José Quiles Flor Fdo. Dr. D. Antonio Garrido del Sólo

SR. PRESIDENTE DE LA COMISIÓN DE DOCTORADO DE LA UNIVERSIDAD DE CASTILLA-LA MANCHA

A Tomás

A mi familia

AGRADECIMIENTOS

A mis directores, Paco y Antonio por hacer posible que este trabajo sea una realidad, y por apoyarme y escucharme en tantos momentos buenos y malos

A Vicente, por estar siempre dispuesto a ayudar, por su interés, por tantas tardes, por ser como es, y claro, por su insustituible apoyo técnico y moral.

A Toni, por participar tan activamente en este trabajo, por sus ideas siempre acertadas, por su ayuda, por su espíritu investigador y por darme la fuerza que necesitaba para terminar este trabajo.

A Valentín, por ser una importantísima fuente de conocimientos de todos los aspectos relacionados con el sistema operativo, y por estar siempre dispuesto a ayudarme a saltar alguno de los muchos obstáculos que me he encontrado en el camino.

A Leo Yiu-Hong, por enseñarme a comer con palillos y por ser la única cara alegre en aquel mar de chinos.

A Blanca, por escuchar siempre mis penas y alegrías, y por compartir tantas cosas. A Pedro Cuenca por estar siempre tan cerca y por las risas que nos echamos. A Pedro García, al Franman y al resto de mazmorreros por compartir tantos momentos. Y, también, a todos los demás compañeros y amigos del departamento.

A mis padres, por apoyarme siempre, y por sufrir tan cerca, aunque estén tan lejos a orillas del mar, el desarrollo de esta tesis. También a mis hermanas, y a José Miguel por sufrir en silencio los chaparrones que le he soltado en más de una ocasión. Y también a las Mari Cruces (madre e hija) y a Tomás (padre), por hacer posible que me sintiera de su familia y por ayudarme siempre con los problemillas de la vida diaria.

A mis sobrinillos, Ana, Alejandro y Pablillo, por abrirme los ojos ante el milagro de la vida y por hacerme pasar, siempre, tan buenos momentos.

A mis amigos de Almuñécar, de Albacete y del Provencio, por hacer más fáciles estos años tan complicados.

A las myrinetes por haberme ayudado a desarrollar una paciencia infinita, tan útil en la vida diaria.

Y sobre todo a Tomás, que apareció en un momento fundamental en mi vida. Gracias por ayudarme, gracias por verlo todo de otra forma, y por conseguir alejarme de tantas preocupaciones. Espero que ahora pueda pasar más horas contigo que con las myris.

Índice General

Índice de figuras

Índice de tablas

INTRODUCCIÓN

De todos es conocido el importante crecimiento, desarrollo y evolución de los clusters de computadoras. No es un concepto nuevo, no es algo que haya surgido de las necesidades actuales de comunicación o procesamiento. Es un concepto que ya tiene sus años, ya es adulto, y por tanto, está en su mejor momento.

Actualmente, la utilización de los cluster y la integración de los mismos tanto en empresas como en centro de investigación, es enorme. Se les llama supercomputadores, porque realmente lo son. Llevan la etiqueta de alto rendimiento a bajo costo y se perfilan como la plataforma paralela ideal, ofreciendo sus múltiples ventajas de escalabilidad, disponibilidad, etc.

Existen múltiples aspectos de los clusters que están en continua investigación, mejora e innovación, como pueden ser la propia arquitectura, los entornos de aplicación, los sistemas operativos, las herramientas de evaluación de rendimiento, etc., y todas las que además aparecen en las distintas áreas técnicas del *IEEE Task Force on Cluster Computing*. Por lo tanto es el objeto de deseo de muchos investigadores actuales.

Por otro lado, si hablamos de aplicaciones, o mejor, de “grandes” aplicaciones, por fin han encontrado su plataforma ideal. Aplicaciones científicas que implicaban modelado, simulación o análisis han visto satisfechas sus elevadísimas necesidades de cálculo. Pero, además, últimamente también van emergiendo los clusters de servidores que van a servir de motor de arranque y funcionamiento de las famosas aplicaciones multimedia. Y éstas son las que más nos interesan precisamente a nosotros. El auge que han experimentado últimamente ha sido realmente espectacular. Se han desarrollado todo tipo de aplicaciones orientadas al entretenimiento (TV, juegos, etc.), a dar nuevos servicios al usuario a través de Internet (e-comercio, e-banco, e-museos, e-cualquier cosa), vídeo bajo demanda, etc.

Para este tipo de aplicaciones, que llevan asociado un elevado coste computacional, la opción de paralelizarlas se ha convertido en la forma natural de procesamiento de las mismas. El procesamiento paralelo es algo que ya está muy maduro y consolidado y que ahora puede ser explotado comercialmente. También ha ayudado en la construcción de los modernos y potentes clusters, por supuesto, el gran desarrollo en tecnología de redes y comunicación, que ha hecho posible la transmisión de información, a más velocidad, entre todo tipo de sistemas tanto homogéneos como heterogéneos.

En nuestro trabajo pretendemos hacer un estudio de los problemas principales de los clusters de computadoras con las aplicaciones multimedia. Enfrentamos los dos mundos y, además, lo hacemos de una forma “real”, sin simulaciones. Se va a partir de una plataforma real, de un cluster de alta velocidad, y de una aplicación multimedia que va a ofrecer la suficiente flexibilidad como para poder imitar el comportamiento de otras, por lo que se pretenderá enfocar el estudio de la forma más general posible.

Hay tres aspectos en los cluster que nos interesan en extremo y que van a ser a los que dediquemos toda nuestra atención:

- Paralelismo
- Red
- La Entrada/Salida

El estudio se va a centrar en ellos y mediante las aplicaciones multimedia los explotará y analizará. Por tanto, se paralelizará la aplicación multimedia y se realizarán estudios experimentales tanto para ver el comportamiento de la red como el de la entrada/salida. Se utilizará una red de alta velocidad, por lo que los requisitos de gran ancho de banda que exigen las aplicaciones multimedia se podrán ver satisfechos, con lo que el estudio va a estar centrado, por tanto, en la entrada/salida. Expondremos los principales problemas y daremos una solución que mejore los resultados obtenidos inicialmente.

En entrada/salida existen numerosos e interesantes trabajos que veremos resumidamente en los capítulos correspondientes. Todos han intentado aportar alguna novedad con el propósito de mejorar algún punto negro de la entrada/salida. En nuestro trabajo se ha hecho un estudio exhaustivo de todos esos trabajos con el fin de plantear un nuevo método híbrido de adelantamiento de datos para arquitecturas cliente-servidor, en uno de los sistemas de ficheros en red más utilizado actualmente, NFS. Pero no sólo va a ser un planteamiento algorítmico y teórico, sino que se va a implementar en el mismo núcleo del sistema operativo donde NFS aparece como módulo, y se van a realizar los experimentos que van a demostrar las mejoras conseguidas.

Debe quedar claro que éste no es un trabajo de investigación en sistemas operativos, sino que es un trabajo de investigación en cluster de computadoras con aplicaciones multimedia, pero que el eje de rotación en torno al cual gira nuestra aportación principal va a ser la *forma de compartir la información multimedia*, y para su implementación hemos bajado hasta las profundidades del núcleo de Linux. Esto es algo muy importante que queremos destacar, ya que la mayoría de los trabajos van orientados al desarrollo de interfaces a nivel de aplicación para mejorar la comunicación entre cliente y servidores. Pero sin acercarse al foco principal que provoca el problema.

Objetivos

Con todo esto, en esta memoria pretendemos exponer un trabajo interesante y novedoso en el que existen aportaciones serias, y que sirva para demostrar, que se ha abierto una puerta en la investigación orientada a la mejora de estándares de E/S

tradicionales, para que sean capaces de soportar de la forma más óptima posible los novedosos y revolucionarios tráficos multimedia.

Los objetivos que se marcaron con este trabajo han sido varios, ya que, en primer lugar, queríamos estudiar los clusters y su comportamiento en la paralelización de aplicaciones multimedia. De este estudio surgió todo un conjunto de resultados. Realizando un análisis de estos resultados, llegamos a la conclusión de que existía un importante problema en la entrada/salida que tuvimos que descifrar. Esto nos llevó hasta el propio sistema de ficheros NFS, a estudiar todos sus detalles y particularidades, para poder entender los resultados obtenidos. Así, nos planteamos el mejorar el propio sistema de ficheros NFS, en lugar de implementar otras soluciones intermedias posibles.

Por lo tanto, tras muchos pequeños objetivos consecutivos apareció el objetivo cinco estrellas, que consistía en *mejorar la forma en que los clientes NFS leen la información compartida*. Queríamos mejorar la forma en la que los clientes del cluster acceden a la información multimedia, reduciendo, por tanto, los tiempos de lectura de la información.

En el capítulo correspondiente se expondrán más detalles de implementación, y los inicios del trabajo, pero lo que quedó como objetivo definido a conseguir fue *incorporar una técnica de prefetching a través de ficheros que consiguiese traer a la cache de los clientes la información multimedia antes de que se necesitase, y pudiese evitar así peticiones al servidor a través de la red.*

Este objetivo se ha conseguido, con lo que nos sentimos satisfechos y a la vez ilusionados por la gran cantidad de cosas que se podrían hacer. Se encontrarán todos los detalles en los capítulos que a continuación hemos elaborado.

Contenido de la tesis

Empezamos esta memoria con un capítulo general sobre clusters de computadoras (capítulo dos), en él pretendemos hacer un resumen del estado del arte actual de los mismos, tocando los aspectos que más nos interesan, como son la computación paralela, las tecnologías de clusters y las aplicaciones que sobre los mismos han conseguido implementarse y desarrollarse, en concreto nos centraremos en las aplicaciones multimedia.

El capítulo tres es una descripción de la arquitectura de la plataforma real utilizada tanto software como hardware. En este capítulo describimos en detalle la aplicación multimedia utilizada, describimos también las formas típicas de compartir la información en un cluster, centrándonos, sobre todo, en el estándar más utilizado actualmente en todo tipo de sistemas, como es NFS. Exponemos, a continuación, brevemente, los fundamentos del sistema operativo utilizado y de la red que interconecta las computadoras del cluster.

En el capítulo cuatro ya nos metemos de lleno en la evaluación de la arquitectura de cluster detallada en el capítulo anterior. Contamos cuáles fueron los primeros pasos de este estudio y el análisis que se llevó a cabo. Pasamos por el ajuste, primero, de los propios parámetros NFS, y, después, de otros ajenos al propio sistema NFS, como son

la topología de red utilizada y los distintos patrones de carga que nos parecen representativos de distintas aplicaciones multimedia. Todo este análisis de resultados nos acercó a dos potentes y útiles técnicas existentes en NFS, como son las técnicas de *caching* y *prefetching*. Con su utilización se pretende reducir el tiempo de acceso de los clientes a la información almacenada físicamente en el servidor. Del estudio de estas técnicas llegamos a la conclusión de que existían ciertos aspectos que se podrían mejorar para acceder a la información de forma, todavía, más eficiente.

Esto es lo que contamos en el capítulo cinco, las propuestas reales de mejora, los posibles cambios que podríamos incorporar al cliente NFS y que nos podrían llevar a una mejora mucho más significativa de los resultados finales. En este capítulo exponemos lo que nos planteamos inicialmente, que fue la posibilidad de cambiar la forma de trabajar del servidor, y cómo ante nuestra impotencia nos trasladamos al cliente, mucho más manejable y abierto a mejoras de este tipo. Plantemos el diseño de la técnica de prefetching a implementar, una técnica basada en grafos de acceso, con el propósito de adelantar datos no sólo del fichero actual que se está leyendo (que eso ya se hacía) sino a través de distintos ficheros.

Una vez planteada la técnica de mejora, llega el momento de experimentar con ella, y de obtener resultados. Un resumen organizado de los principales resultados (que no los únicos) los podemos encontrar en el capítulo seis de la memoria. En este capítulo se puede comprobar el éxito de la implementación de la técnica de prefetching, con la consecución de una reducción en los tiempos de lectura de los clientes NFS para dos patrones de carga significativos en la mayoría de las aplicaciones multimedia.

Por último en el capítulo siete se exponen las conclusiones principales a las que hemos llegado después de todo el trabajo realizado, y las tareas que podrían seguir al mismo, es decir, los posibles trabajos futuros que nos podríamos plantear para continuar nuestro trabajo investigador. A continuación aparecen las referencias bibliográficas que hemos tenido en cuenta en este estudio y que nos sirven para concluir el mismo.

CAPÍTULO 2

CLUSTERS DE COMPUTADORAS

2.1. INTRODUCCIÓN

Vamos a tratar de dar una descripción clara y concisa, en este primer capítulo, de lo que son los cluster de computadoras, sus ventajas frente a otras plataformas, sus características, prncipios de diseño, y en definitiva sus aspectos de mayor interés. En un principio trataremos los cluster de forma general y después los propondremos candidatos ideales para el procesamiento multimedia. Esto nos llevará a resaltar algunos de los problemas con este tipo de aplicaciones. En este capítulo se mencionarán de forma general y en el transcurso de la tesis quedará demostrada su existencia y se plantearán soluciones.

Como ya se ha comentado en los objetivos de esta tesis doctoral, nuestro eje principal son las aplicaciones multimedia sobre cluster de computadoras. Pretendemos valorar los problemas principales que conlleva este enfrentamiento y para ello los aspectos que queremos analizar son:

- Paralelismo: la utilidad de los cluster como plataformas paralelas capaces de permitir la divisón de la carga computacional entre las distintas máquinas, lo que va a reducir los tiempos de ejecución de aplicaciones que suelen conllevar una gran carga computacional.
- La tecnología de red utilizada, ya que este tipo de aplicaciones tiene como requisito exigir un gran ancho de banda en la red por la que se van a transmitir. Además, como veremos en el tema siguiente, la forma de compartir la información será a través de la red, por lo que habrá que contar con redes de gran ancho de banda. Bien es verdad, que ésto ya no es problema, la evolución de la tecnología de red corre igual de rápido que lo hacen las nuevas aplicaciones multimedia, por lo que no lo plantearemos como problema sino como exposición de la situación actual y futura.
- Aplicaciones multimedia: por último haremos un estudio general de las aplicacioes que nos interesan y de su instalación, ejecución y transmisión a través de los clusters.

En este capítulo analizaremos todos estos aspectos, y algunos más, de forma totalmente teórica. Después en el capítulo 2, volveremos a ellos, pero especificando los

elementos software y hardware utilizados en nuestra plataforma real, en nuestro cluster, para realizar los experimentos que, en los siguientes capítulos, expondremos y que nos llevarán a conocer, más detalladamente, cúales son los problemas y aspectos importantes a tener en cuenta cuando queremos dedicar un cluster de computadoras al procesamiento de aplicaciones multimedia.

2.2. DEFINICIÓN DE GRUPOS O *CLUSTERS* DE COMPUTADORAS

Un *cluster* es una colección de computadoras completas (nodos), que están físicamente interconectadas mediante una red de altas prestaciones o mediante una red de área local (LAN). Cada nodo es un servidor, una estación de trabajo o una computadora personal. Los más importante es que todos los nodos del *cluster* deben ser capaces de trabajar juntos colectivamente como un recurso de computación integrado y único; además, manteniendo el papel convencional de la utilización de cada nodo como usuarios interactivo individualmente [37].

También se puede definir como un tipo de sistema de procesamiento distribuido, que consta de una colección de computadoras interconectadas que trabajan juntas como un único recurso de computación integrado. Un nodo puede ser un sistema uni o multiprocesador, ya sean PCs, estaciones de trabajo o SMPs (*Symmetric Multiprocessors*) con memoria, facilidades de entrada/salida, y un sistema operativo. Un *cluster* se refiere, por lo general, a dos o más computadoras conectadas por una LAN. Tal sistema puede proporcionar una forma muy rentable de conseguir beneficios y posibilidades que históricamente se habrían encontrado solo en los caros sistemas de memoria compartida [7].

Hay cinco conceptos en relación con la arquitectura que se mezclan en un *cluster*, como conjunto de computadoras completas (*nodos*) *interconectadas* que trabajan colectivamente como un *sistema único* para proporcionar servicios ininterrumpidos (*disponibilidad*) y eficientes (*rendimiento*). En la siguiente figura mostramos la arquitectura conceptual de un *cluster* (ver figura 2.1).

- *Grupo de Nodos*: Cada nodo es una computadora completa. Esto implica que cada nodo tiene su procesador(es), cache, memoria, disco, y algunos adaptadores de entrada/salida. Además, reside en cada nodo un sistema operativo estándar y completo. Un nodo puede tener más de un procesador, pero sólo una copia del sistema operativo.

- *Imagen de Sistema Único*: Un *cluster* es un recurso de computación único. Esta es una diferencia con un sistema distribuido (una red de computadoras local), donde los nodos se usan sólo como recursos individuales. Un *cluster* entiende el concepto de recurso único por un número de técnicas de imagen de sistema único (SSI, *Single-System Image*). SSI hace un *cluster* más fácil de usar y manejar. Los *cluster* más disponibles comercialmente no han alcanzado el estado de operaciones SSI total.

- *Conexión entre los nodos*: los nodos de un *cluster* están conectados usualmente por una red, tal como Ethernet, FDDI, Canal de fibra o un conmutador ATM. Sin embargo, se usan los protocolos estándares para suavizar la comunicación entre los nodos.

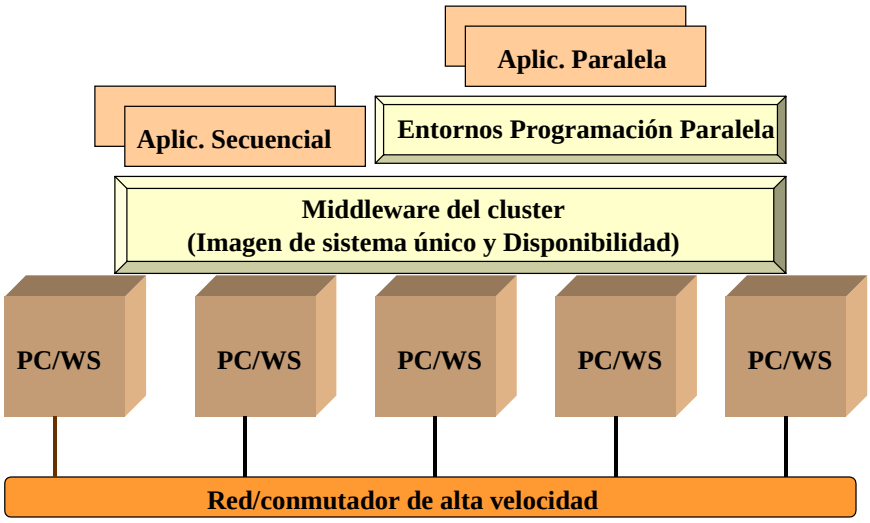


Figura 2.1. Arquitectura típica de un *cluster* de múltiples computadoras

• *Disponibilidad realzada*: los *cluster*s ofrecen una forma de realzar la disponibilidad de un sistema a un costo muy razonable, es decir el porcentaje de tiempo que un sistema está disponible al usuario.

• *Mejor rendimiento:* un *cluster* debe ofrecer alto rendimiento en un número determinado de áreas de servicio. Un área es tratar un *cluster* como un superservidor. Si cada nodo de un *cluster* de n nodos puede servir a m clientes, el *cluster* como conjunto puede servir a mxn clientes simultáneamente. Otra área es usar un *cluster* para minimizar el tiempo de ejecución de un único, pero gran trabajo, mediante procesamiento paralelo distribuido [37].

Por lo tanto, los componentes principales en un *cluster de computadoras son:*

- Computadoras de alto rendimiento
- Sistemas operativos
- Redes/conmutadores de alto rendimiento (tales como Gigabit Ethernet y Myrinet)
- Tarjetas de interfaz de red (NICs, N*etwork Interface Card*)
- Protocolos de comunicación rápidos y servicios (como mensajes activos)
- El *middleware* del *cluster* (Imagen de sistema único y disponibilidad): elementos hardware, kernel del sistema operativo, aplicaciones y subsistemas.
- Entornos de programación paralela y herramientas
- Aplicaciones: secuenciales, y paralelas o distribuidas [7].

2.3. COMPUTACIÓN PARALELA DE BAJO COSTO

Los *clusters* se conocen por varios nombres, tales como *clusters* de computadoras, *clusters* de estaciones de trabajo (COW, *Cluster of Workstations*), red de estaciones de trabajo (NOW, *Network of Workstations*), etc. Los *cluster*s están creando una tendencia en desarrollo de computadoras paralelas escalables y con un costo razonable.

El uso de *clusters* para probar, depurar y lanzar aplicaciones paralelas se está convirtiendo en una alternativa muy popular al uso de plataformas de computación

paralela, típicamente caras y especializadas. Un factor importante que ha hecho el uso de los *clusters* una opción práctica de trabajo es la estandarización de muchas utilidades y herramientas utilizadas por aplicaciones paralelas, como la librería de paso de mensajes MPI [25], y el lenguaje paralelo de datos HPF [43]. En este contexto la estandarización ayuda al desarrollo, test y a la ejecución de aplicaciones incluso en NOWs, para después llevarlas con pequeñas modificaciones a plataformas paralelas dedicadas. La siguiente lista nos muestra las razones por las que se prefieren las NOWs en lugar de computadoras paralelas especializadas:

- Las estaciones de trabajo individuales tienen cada vez mayor poder de procesamiento. El rendimiento ha crecido vertiginosamente.
- El ancho de banda para comunicaciones entre estaciones de trabajo se ha incrementado y la latencia decrece gracias al desarrollo de nuevas tecnologías y protocolos.
- Los *clusters* son más fáciles de integrar en las redes existentes que los computadoras paralelas de propósito especial
- Las herramientas de desarrollo para estaciones de trabajo, más maduras que muchas de las dadas, para sistemas paralelos no estándares.
- Los *clusters* son baratos y una alternativa realmente disponible a las plataformas especializadas de computación de alto rendimiento.
- Los *clusters* pueden crecer fácilmente, la capacidad de los nodos puede incrementarse fácilmente añadiendo procesadores o memoria adicional.

Ha habido, también, una rápida convergencia en el rendimiento de los procesadores y funcionalidad a nivel de kernel de estaciones de trabajo UNIX y PCs en los últimos años (introducción de máquinas basadas en Pentiums de alto rendimiento y los sistemas operativos Linux y Windows NT). Esta convergencia ha llevado a incrementar el interés por utilizar sistemas basados en PCs como un recurso computacional de bajo costo para computación paralela, y a iniciar múltiples proyectos software cuyo objetivo principal es utilizar estos recursos de forma cooperativa [7].

La computación paralela requiere múltiples procesadore***s***, una red para conectarlos y coordinarlos y un entorno para crear y manejar procesamiento paralelo. Intervienen los siguientes elementos:

- Un sistema operativo adecuado para coordinar las tareas entre los procesadores.
- Un paradigma de programación paralela, que podemos dividir en dos clases
    - Mediante paso de mensajes: MPI, MPL, PVM
    - Mediante memoria compartida
- Un algoritmo paralelo y un programa paralelo

En nuestro estudio el sistema operativo anfitrión será Linux, que es la mejor opción para diseñar supercomputadores paralelos de bajo coste. Utilizaremos MPI para el paso de mensajes y una aplicación multimedia paralela para realizar el estudio experimental como se describe en el siguiente capítulo. Donde, además, se especificarán algunas de las implicaciones de la programación paralela, como son:

- Descomposición de un algoritmo o datos en partes.
- Distribución de las partes como tareas que son realizadas sobre múltiples procesadores simultáneamente.

- Coordinación del trabajo y la comunicación de los procesadores.

Veamos ahora una breve descripción de MPI y sus posibles implementaciones

2.3.1. MPI (*Message Passing Interface*)

La madurez de un compilador o de una librería de paso de mensajes es un factor fundamental cuando se elige un paradigma y una implementación particular. El paso de mensajes es un método muy utilizado desde hace ya muchos años, y su funcionalidad e implementación son sencillos. Estas librerías son críticas en todos los sistemas paralelos de memoria distribuida, por lo que se tratan de hacer lo más robustas y potentes posibles.

Los compiladores de información paralela son relativamente nuevos y poco experimentados, pues no soportan todas las características del lenguaje. Los compiladores y sistemas en tiempo de ejecución poseen normalmente errores, además son muy complejos y difíciles de implementar. Por estas y otras razones se pensó en MPI como soporte para la comunicación.

Algunos de los factores más importantes que podemos tener presentes a la hora de escoger un paradigma son la facilidad de programación, la flexibilidad, la eficiencia, la escalabilidad, la portabilidad, la entrada y salida y el coste.

El paso de mensajes se define como un conjunto de procesos que usan únicamente memoria local y que se comunican enviándose mensajes. La transferencia de información entre dos procesos requiere una operación de cooperación en ambos para que pueda llevarse a cabo, cada envío debe estar ligado a una recepción de información. Esto se consigue realizando llamadas a librerías del sistema que permiten esta comunicación interprocesador. Las librerías más utilizadas son MPI (*Message Passing Interface)* [58] y PVM *(Parallel Virtual Machine)* [70].

De MPI se creó una versión estándar en 1993, entre varios fabricantes de computadoras paralelas, programadores expertos de software y científicos de aplicaciones. Está disponible para C y Fortran y para una amplia gama de máquinas paralelas. La plataforma destino es una arquitectura de memoria distribuida, y toda la comunicación es mediante paso de mensajes. Aquí, el paralelismo es explícito, sólo habrá paralelismo donde lo decida el programador. Usa una arquitectura SPMD, un solo programa con varios flujos de información [66].

PVM posibilita a una colección de sistemas de computación diferentes ser vistos como una única máquina paralela. Fue inicialmente desarrollada en 1989 por Oak Ridge del National Laboratory como una herramienta de investigación para explorar redes de ordenadores homogéneas. En la actualidad es un paquete de dominio público. Se trata de una herramienta que permite ejecutar tanto programas paralelos como concurrentes. Trabaja sobre una red heterogénea de máquinas Unix conectadas por red. Aquí toda la comunicación se realiza a través de mensajes. Consta de dos partes principales: un proceso demonio PVM que se ejecuta en cada procesador y una librería de rutinas que proporciona control del procesador y el paso de mensajes [70].

Al diseñar MPI se ha buscado hacer uso de las más atractivas características de un número de sistemas existentes de librerías de paso de mensajes. MPI se suele definir en la mayoría de las documentaciones como la primera biblioteca de Paso de Mensajes *estándar* y *portable*, y como una colección de rutinas que facilitan la comunicación entre procesadores en programas paralelos.

Las características más destacables del MPI son:

- Contiene, por una parte, rutinas básicas de comunicaciones a bajo nivel y, por otra, un conjunto de rutinas de alto nivel que permiten realizar operaciones de comunicación especiales.
- Permite trabajar con tipos de datos básicos y nuevos tipos que pueden definirse con algunas de sus operaciones.
- Tiene interfaz para los lenguajes de programación C y FORTRAN.
- Los procesos involucrados en la ejecución paralela de un programa se identifican mediante un entero que se denomina *rango*; de tal modo que si tenemos n procesos, éstos se identificarán mediante los *rangos* 0, 1, …., n-1.
- Para facilitar las comunicaciones en MPI, se definen los denominados *comunicadores*. Un comunicador está formado por un grupo de procesos que van a realizar una determinada operación e identificado mediante una etiqueta. Esto permite la realización de diferentes operaciones sobre grupos de procesos distintos.

MPI proporciona muchas características encaminadas a mejorar el rendimiento de nuestros programas. Podemos mencionar como las más importantes la portabilidad, la estandarización, la ejecución paralela e implementaciones eficientes.

Los datos utilizados para la comunicación entre procesadores deben ser del mismo tipo, es decir, tanto la operación de envío como la de recepción deben utilizar el mismo tipo de dato. En [66] podemos encontrar la descripción detallada de los tipos de datos básicos con los que cuenta MPI y su equivalencia con los tipos más usado en los lenguajes de programación

Las funciones de comunicación de MPI pueden ser punto a punto o colectivas. Se dice que una comunicación es punto a punto cuando existe un procesador que reciba y otro procesador que envíe mensajes, es decir, la transmisión de datos se realiza mediante una comunicación uno a uno. La comunicación punto a punto requiere de dos funciones que permitan la comunicación dentro de un grupo de procesadores. Las funciones de envío y de recepción. Dentro de la comunicación punto a punto se debe declarar un grupo de procesadores que conformarán un universo, cada procesador debe ser fácilmente identificable dentro del grupo. El comunicador *MPI_COMM_WORLD* identifica a todos los *n* procesadores que conforman el grupo creado por el usuario.

La comunicación punto a punto puede ser a su vez bloqueante o no bloqueante. Una comunicación bloqueante se refiere al hecho de que un procesador (emisor) tiene que esperar hasta que el procesador receptor complete cierta etapa de su programa para que pueda continuar su ejecución. Un programa con comunicación bloqueante produce una gran cantidad de tiempos muertos, lo que repercute en el aprovechamiento de la

velocidad y de la utilización al máximo de los procesadores. En la comunicación no bloqueante, el proceso recupera el control inmediatamente después de realizar la operación. Clasificaremos el modo de comunicación bloqueante de acuerdo a la manera en que los procesadores hacen las operaciones de recepción y de envío. Los modos de comunicación bloqueante para el envío (*send)* que proporciona la librería de MPI son: envío estándar, *MPI_Send()*; envío síncrono, *MPI_Ssend()*; envío con buffers, *MPI_Bsend()* y envío preparado, *MPI_Rsend()*.

Sólo existe un modo de recibir cuando existe una comunicación bloqueante pero ésta puede ser utilizada para todos los tipos de envío. Esta función es una operación básica que acepta los mensajes enviados desde otro procesador. La función para recibir es el estándar *MPI_Recv().*

En la comunicación colectiva, ésta se produce entre todos los procesos de un grupo, todos ellos deben ejecutar la operación colectiva. Lo primero que se debe tener en mente es que no empleará ninguna operación *send* o *rec**v*** como ocurrió en la comunicación punto a punto. Lo que distingue una comunicación colectiva de una punto a punto, es la distribución de un dato a muchos procesadores que pertenecen al comunicador indicado.

Brevemente, las funciones que MPI proporciona para la distribución de datos son [66].

- Difusión de datos *(Broadcast**): MPI_Bcast ( )***
- Dispersión de datos *(Scatter): MPI_Scatter ( )*
- Reunión de datos *(Gather): MPI_Gather ( )*
- Reunión en todos *(Allgather**: MPI_Allgather( )***
- Todo a todos *(all to all: MPI_Alltoall ( )*
- Reducción *(Reduce* y *Allreduce). MPI_Reduce( )*

Un programa escrito en MPI incluirá tres pasos generales, el de inicializar la comunicación, el de la comunición propiamente dicha para compartir datos entre procesos, y, por último, la de finalizar el ambiente paralelo. En la parte de comunicación se invocarán funciones MPI con el siguiente formato:

rc = MPI_Xxxxx(parámetros)

Donde *rc* es una variable tipo entero que retorna el valor de cero o uno según el éxito/fracaso de la función a su termino. La llamada a alguna rutina de MPI deber ser en mayúsculas, como debe ser también el primer carácter después del guión abajo, a continuación todo debe estar en minúsculas. MPI tiene 125 funciones. Sin embargo, es posible crear un programa paralelo MPI utilizando poco mas de media docena de ellas. Las clasificaremos por grupos según su función según el orden de aparición [58].

A principios de Marzo de 1995 el Forum de MPI comienza a considerar correcciones y extensiones al estándar de MPI el cual genera MPI-2, esta extensión

contiene nuevos tipos de funciones para comunicaciones con un solo procesador, paralelismo en dispositivos de entrada/salida, programación *multithreading*, etc. y además contiene correcciones de la versión MPI-1. Para el uso de las rutinas de MPI-2 se hace uso del lenguaje C++ [57].

En la actualidad existen diversas implementaciones de MPI, algunas de las cuales son de dominio público [15], entre las que podemos destacar las implementaciones MPICH y LAM.

MPICH [55] se desarrolló en el Argonne national Laboratory de la Universidad del estado de Mississippi. Existen versiones para Linux y Windows-NT, un ejemplo de uso de mpich para Linux usado en clusters, es el supercluster *RoadRunner* de la universidad de Nuevo Méjico. Es un estándar que sigue en continua evolución y periódicamente se publican nuevas versioens que mejroan los posibles problemas de las anteriores. Actualmente se ofrece la posibilidad de cooperación en trabajos relacionados con MPICH en redes de área amplia y en dispositivos VIA (*Virtual Interface Architecture*) para MPICH, que es una especificación de una arquitectura estándar de la industria para comunicaciones escalables en clusters.

LAM (*Local Area Multicomputer*) [45] es un entorno de programación y desarrollo MPI para computadoras heterogéneas en una red. Con LAM, un cluster dedicado puede actuar como un computador paralelo. Actualmente hay varias versiones disponibles. LAM es portable a muchas máquinas UNIX e incluye soporte estándar para SUN (SunOS y Solaris), SGI IRIX, IBM AIX, DEC OSF/1, HP-UX, y Linux.

MPI-FM (*Fast messages*) [56] es una versión de MPICH construida sobre mensajes rápidos. Requiere el entorno de ejecución HPVM (*High Performance Virtual Machine*) que está disponible tanto para Linux como para Windows-NT. Se utiliza en el Supercluster NCSA NT [61]

MPI/Pro [59] es una iplementación comercial de MPI. Existen versiones tanto para Windows-NT y procesadores Alpha, como para Linux

WMPI (Windows message Passing Interface) [60] fue una de las primeras implementaciones para sistemas Win32. Se desarrolló en la Universidad de Coimbra, Portugal. Ha seguido evolucionando hacia el entorno dinámico del estándar MPI-2.

2.4. POSIBLES CLASIFICACIONES DE LOS CLUSTERS

Veamos una clasificación de los *cluster*s. Se han dado varias en la literatura existente, además con sinónimos que confunden. En primer lugar clasificamos los *cluster*s usando cuatro atributos ortogonales: *empaquetamiento*, *control*, *homogeneidad* y *seguridad*. Por simplicidad, permitimos sólo dos valores para cada atributo (Ver tabla 2.1). Actualmente, solo se usan dos combinaciones en los *cluster*s reales, llamadas

clusters dedicados y *clusters de empresa*. Pero conceptualmente, las 16 combinaciones son posibles.

Tabla 2.1. Atributos usados en la clasificación de los *cluster*s.

ATRIBUTOS	VALORES	VALORES
Empaquetamiento	Compacto	Ligero
Control	Centralizado	Descentralizado
Homogeneidad	Homogéneo	Heterogéneo
Seguridad	Cerrado	Abierto
EJEMPLO	***Cluster dedicado***	***Cluster de empresa***

- *Empaquetamiento*: los nodos de un *cluster* pueden estar empaquetados de forma compacta o ligera. En un *cluster* compacto, los nodos están estrechamente empaquetados en un estante de una habitación, y los nodos no están acompañados de periféricos (monitores, teclados, ratones, etc.). En un *cluster* ligero, los nodos van unidos a sus periféricos usuales (son completos PCs o estaciones de trabajo), y pueden estar situados en diferentes habitaciones, edificios, o regiones geográficamente remotas.

El empaquetamiento afecta directamente a la longitud del cable de comunicación, y de este modo a la selección de la tecnología de interconexión. Mientras que un *cluster* compacto puede utilizar un ancho de banda alto y red de comunicación de baja latencia que a menudo está patentada, los nodos de un *cluster* ligero, están conectados normalmente a través de LANs o WANs estándares.

- *Control:* Un *cluster* puede estar controlado o gestionado de forma centralizada o descentralizada. Un *cluster* compacto normalmente tiene control centralizado, mientras que un *cluster* ligero suele estar controlado de la otra forma. En un *cluster* centralizado, todos los nodos son propiedad, están controlados, gestionados y administrados por un operador central. En un *cluster* descentralizado, los nodos tienen propietarios individuales. Esta falta de un único punto de control hace la administración de estos *cluster*s muy difícil, así como las llamadas a técnicas especiales de planificación de procesos, migración de la carga de trabajo, puntos de chequeo, etc.

- *Homogeneidad*: El significado de un *cluster* homogéneo es que los nodos adoptan la misma plataforma. Es decir, usan la misma arquitectura de procesador y el mismo sistema operativo. Además, los nodos son del mismo vendedor. Un *cluster* heterogéneo usa nodos de diferentes plataformas. La interoperatibilidad es un tema importante en estos *cluster*s.

- *Seguridad*: La comunicación entre los nodos del *cluster* puede ser abierta o cerrada. En un *cluster* abierto, los caminos de comunicación entre los nodos se abren al mundo exterior. Una máquina externa al *cluster* puede acceder a las vías de comunicación, y de este modo a los nodos individuales, usando protocolos estándares (como TCP/IP). Tales *cluster*s abiertos son fáciles de implementar, pero tienen algunos inconvenientes:
  - La comunicación dentro del *cluster* no es segura, a menos que los subsistemas de comunicación realicen algún trabajo adicional para asegurar la privacidad y seguridad

- La comunicación exterior puede crear problemas a la comunicación de dentro del *cluster* de forma no previsible
- Los protocolos de comunicación estándares tienden a tener una alta sobrecarga

En los *cluster*s cerrados, la comunicación dentro del *cluster* es ajena al mundo exterior, lo que alivia los problemas anteriores. Un inconveniente es que no hay actualmente estándares para una comunicación eficiente dentro del *cluster*.

• *Clusters de empresa frente a clusters dedicados:* Un *cluster* dedicado tiene las siguientes características, entre otras:
- Por lo general está configurado de forma homogénea con el mismo tipo de nodos
- Están instalados en un estante en una habitación de computadoras central
- Están gestionados por un único administrador

Los *cluster*s dedicados es usan como sustitutos de los supercomputadoras tradicionales. Un *cluster* dedicado se usa, instala y administra como una única máquina. Muchos usuarios pueden entrar y ejecutar tanto trabajos interactivos como por lotes. Los *cluster*s ofrecen buen rendimiento así como un tiempo de respuesta reducido.

Un *cluster* de empresa se usa principalmente para utilizar recursos desocupados en los nodos. Tienen las siguientes características:
- Cada nodo es una estación de trabajo o PC con todos los periféricos necesarios
- Los nodos están por lo general geográficamente distribuidos, no necesariamente en la misma habitación o el mismo edificio.
- Los nodos pertenecen a múltiples propietarios. El administrador tiene un control limitado sobre los nodos. Los trabajos locales de cada propietario tienen una prioridad más alta que los trabajos de empresa.
- Los *cluster*s se configuran a menudo con nodos heterogéneos, y los nodos se conectan a través de una Ethernet de bajo costo [37].

Otra posible clasificación teniendo en cuenta varias categorías, podría ser la siguiente [7]:

1. **Según la aplicación:** de misión crítica o para la ciencia computacional:
- *Clusters* de alto rendimiento (HP, *High Performance Clusters*)
- *Clusters* de alta disponibilidad (HA, *High Availability Clusters*)

2. **Según el propietario de los nodos**
- *Clusters* dedicados
- C*lusters* no dedicados

3. **Según el hardware de los nodos**
- *Cluster* de PCs
- *Cluster* de estaciones de trabajo
- *Cluster* de SMPs

4. **Según el sistema operativo de los nodos**
- *Clusters* de Linux
- *Clusters* de Solaris
- *Clusters* NT

- *Clusters* AIX
- *Clusters* de VMS de Digital
- *Clusters* de HP-UX
- *Cluster* Wolfpack de Microsoft

5. Según la configuración de los nodos
- *Clusters* homogéneos
- *Clusters* heterogéneos

6. Según los niveles en el *cluster*
- *Cluster* de grupo (de 2 a 99 nodos)
- *Clusters* de departamento (de 10 a 100 nodos)
- *Clusters* de organización (algunos cientos de nodos)
- *Metacomputadoras* nacionales (basados en Internet y Wan)
- *Metacomputadoras* internacionales (basados en Internet)

Los c*lusters* individuales pueden interconectarse para formar un sistema mayor (*cluster* de *clusters*) y, en efecto, Internet misma puede usarse como *cluster* de computación. El uso de redes de área amplia para computación de alto rendimiento ha llevado a la emergencia de un nuevo campo llamado *Metacomputación*.

2.5. DISEÑO

Hay que tener en cuenta varios asuntos en el desarrollo y uso de un *cluster*, como son la disponibilidad, imagen de sistema único, gestión de trabajos y comunicación eficiente. Aunque se ha trabajado mucho en estos temas, hay actualmente varias áreas de investigación y desarrollo activas.

1. *Disponibilidad*

Los *cluster*s pueden proporcionar una alta disponibilidad a un costo razonable con mucha redundancia en procesadores, memorias, discos, dispositivos de entrada/salida, redes, etc.

En el diseño de sistemas robustos, altamente disponibles, se utilizan tres términos que a menudo van juntos: fiabilidad (mide cuanto tiempo un sistema puede operar sin caídas), disponibilidad (indica el porcentaje de tiempo que un sistema está disponible al usuario) y mantenimiento (se refiere a cómo de fácil es mantener el sistema, incluyendo mantenimiento hardware y software, reparaciones, actualizaciones, etc.). La disponibilidad es la más interesante ya que combina los otros conceptos.

Sin embargo para explotar este potencial se necesitan técnicas de disponibilidad, como las siguientes:

- Redundancia aislada: una técnica clave para mejorar la disponibilidad en cualquier sistema es usar componentes redundantes. Cuando un componente (componente primario) falla, el servicio que se proporciona es que otro componente (componente de reserva) se hace cargo. Además, los componentes primario y de reserva deben estar aislados uno del otro, lo que indica que no deberían estar sujetos a la misma causa de fallos.

• Sobre fallo (*failover*): esta es probablemente la característica más importante en los *cluster*s actuales para aplicaciones comerciales. Cuando un componente falla, esta técnica permite al resto del sistema hacerse cargo de los servicios que proporcionaba originalmente el componente que ha fallado. Este mecanismo debe proporcionar algunas funciones como diagnóstico, notificación y recuperación de fallos.

• Esquemas de recuperación: la recuperación de fallos se refiere a las acciones necesarias para hacerse cargo de la carga de trabajo de los componentes que han fallado. Hay dos tipos de técnicas de recuperación, *hacia atrás*, donde los procesos en ejecución salvan periódicamente un estado consistente (llamado punto de chequeo) en un almacenamiento estable. Es relativamente fácil de implementar de forma portable e independiente de la aplicación, y se usa bastante. Y, *hacia delante*, técnica que se utiliza en aquellos sistemas donde el tiempo de ejecución es crucial, tales como sistema en tiempo real; el sistema no vuelve atrás al último punto de chequeo bajo un fallo. En lugar de esto, el sistema utiliza la información de diagnóstico del fallo para reconstruir un estado de sistema válido y continuar la ejecución. La recuperación hacia delante depende de la aplicación y puede necesitar hardware extra.

2. *Imagen de sistema único*

Un conjunto de estaciones de trabajo conectadas mediante una Ethernet no es necesariamente un *cluster*. Un *cluster* es un sistema único. Tomemos un simple ejemplo. Supongamos que una estación de trabajo tiene un procesador de 300 Mflop/s, 512 MB de memoria, un disco de 4 GB y que puede soportar 50 usuarios activos y 1000 procesos. Mediante el agrupamiento de 100 de estas estaciones de trabajo, ¿Podemos tener un sistema único equivalente a una enorme estación de trabajo, o megaestación, que tiene un procesador a 30 Gflop/s, 50 GB de memoria, y 400 GB de disco, y puede soportar 5000 usuarios activos y 100000 procesos? Este es un objetivo atractivo, pero difícil de alcanzar. Las técnicas de Imagen de Sistema Único (SSI, *Single System Image*) se proponen alcanzarlo.

SSI no significa tener una única copia de imagen de sistema operativo residiendo en memoria, como en una estación de trabajo. Si no que significa la ilusión de un único sistema caracterizado por lo siguiente:

- *Sistema único*: el *cluster* completo es visto por los usuarios como un sistema, que tiene múltiples procesadores. El usuario puede decir: “ejecuta mi aplicación usando cinco procesadores”. Esta es la diferencia con un sistema distribuido.
- *Control único*: lógicamente, un usuario final o usuario del sistema utiliza servicios de un lugar con una interfaz única. Por ejemplo, un usuario lanza trabajos por lotes a un conjunto de colas; un administrador del sistema configura todos los componentes software y hardware del *cluster* desde un punto de control.
- *Simetría*: un usuario puede usar un servicio de *cluster* de cualquier nodo. En otras palabras, todos los servicios y funcionalidades de *cluster* son simétricas a todos los nodos y a todos los usuarios, excepto a aquellos protegidos por los usuales permisos de acceso.

- *Posición transparente*: el usuario no es consciente del paradero de los dispositivos físicos que eventualmente proporcionan servicios.

La motivación principal para tener un SSI es que permite que un *cluster* sea usado, controlado y mantenido como una estación de trabajo tan familiar para nosotros. La palabra *único* es a veces llamada como sinónimo de global o central. Por ejemplo, un sistema de ficheros global significa lo mismo que una jerarquía de ficheros única, al cual un usuario puede acceder desde cualquier nodo. Un punto único de control permite a un operador monitorizar y configurar el sistema del *cluster*.

Aunque existe la ilusión de sistema único, un servicio o funcionalidad de *cluster* se realiza a menudo de forma distribuida, por la cooperación de múltiples componentes. Un requisito principal (y ventaja) de las técnicas de SSI es que proporcionan tanto los beneficios de una implementación distribuida como los beneficios de uso de un sistema único. La ilusión de un SSI se puede obtener en algunas capas, como las siguientes. Señalar que estas capas pueden solaparse unas con otras.

- Capa de software de aplicación: dos ejemplos son los *Web browsers* y varias bases de datos paralelas. El usuario ve una imagen de sistema único gracias a la aplicación y no es ni siquiera consciente de que está usando un *cluster*. Esta aproximación implica la modificación de aplicaciones de estaciones de trabajo para *cluster*s.

- Capa hardware o kernel: idealmente, el sistema operativo o el hardware deberían proporcionar el SSI. Desafortunadamente, esto no es una realidad todavía. Además, es extremadamente difícil proporcionar una imagen de sistema único sobre *cluster*s heterogéneos.

- Capa por encima del kernel: la aproximación más viable es construir una capa SSI justo por encima del kernel. Esta aproximación es prometedora ya que esw independiente de la plataforma y no requiere la modificación de las aplicaciones. Muchos sistemas de gestión del trabajo del *cluster* han adoptado ya esta aproximación.

La imagen de sistema único es un concepto muy rico, que consiste de punto de entrada único, jerarquía de ficheros única, espacio de entrada/salida único, red única, punto de control único, sistema de gestión de trabajo único, espacio de memoria único, y espacio de procesos único.

- *Punto de entrada único*: que habilite a los usuarios a entrar en el *cluster* (como por ejemplo, mediante telnet, rlogin, o http) como un host virtual, aunque el *cluster* puede telnet múltiples nodos host físicos para servir las sesiones login. El sistema distribuye de forma transparente los login de los usuarios y las peticiones de conexión a los diferentes hots físicos para equilibrar la carga.

- *Jerarquía de ficheros única*: usaremos este término para indicar la ilusión de un sistema de ficheros enorme y único que integra de forma transparente los discos locales y globales y otros dispositivos de ficheros. En otras palabras, todos los ficheros que un usuario puede necesitar se almacenan en algunos subdirectorios del directorio raíz /, a los que se puede acceder a través de llamadas Unix ordinarias. Esto no lo

debemos confundir con el hecho de que pueden existir múltiples sistemas de ficheros en una estación de trabajo como subdirectorios del directorio root.

Se han proporcionado parcialmente las funcionalidades de una jerarquía de ficheros única por sistemas de ficheros distribuidos existentes, tales como los Sistemas de Ficheros en Red (NFS, *Network File System*) y Sistemas de Ficheros de Andrew (AFS, *Andrew File System*). Desde el punto de vista de cualquier proceso, los ficheros pueden residir en tres tipos de posiciones en un *cluster*. *Almacenamiento local*, que es el disco en el nodo local de un proceso. Los discos en nodos remotos se conocen como a*lmacenamiento remoto.* Y *almacenamiento estable*, que será persistente y tolerante a fallos, se muestra como una unidad centralizada lógicamente (global). Los ficheros en el almacenamiento estable se llamarán ficheros globales, los de almacenamiento local, ficheros locales, y los de almacenamiento remoto, ficheros remotos.

Para mejorar el rendimiento, se usa ampliamente el manejo de la cache en muchos sistemas de ficheros. Una aproximación muy popular es la cache en el cliente: el nodo que realiza peticiones de ficheros utiliza la cache con los segmentos de fichero más usados. Sin embargo, la cache conlleva problemas de coherencia (o consistencia). Aunque se pueden utilizar las técnicas de coherencia de cache desarrolladas para DSMs (*Distributed Shared Memory Multiprocessors*). Hay algunas formas de implementar una imagen de jerarquía de ficheros única:

1. **NFS**: El Sistema de Ficheros en Red es un estándar abierto desarrollado por Sun Microsystems, y tiene mucha aceptación en plataformas Unix. Los sistemas de ficheros en nodos remotos se adjuntan (montan) a una jerarquía única. Cuando un proceso realiza una petición de operación sobre un fichero, NFS la convierte en una llamada de procedimiento remoto (*RPC, Remote Procedure Call*) y la envía al nodo remoto a través del estándar TCP/IP. La operación sobre el fichero se ejecuta por un nodo remoto y el resultado se devuelve por una RPC. Esto se hace de forma transparente. El usuario puede acceder a ficheros remotos tan fácil como lo hace a los ficheros locales. El principal problema de NFS es su pobre escalabilidad. Lo veremos en detalle en el siguiente capítulo.

2. **AFS**: El Sistema de ficheros de Andrew quiere realzar la escalabilidad del sistema de ficheros. Divide un gran sistema distribuido en una jerarquía de segmentos, cada uno de los cuales con sus propios servidores de ficheros. Para reducir el tráfico en la red, AFS realiza un uso extensivo de la cache y emplea un *protocolo de estado*, en lugar del protocolo que se utiliza en NFS. Reduce la carga del servidor repartiendo las funciones usuales del servidor de ficheros en NFS a los clientes. Sin embargo el acceso a los ficheros locales a través de AFS es mucho más lento que con operaciones de ficheros Unix ordinarias. (Esto es similar a un sistema paralelo que muestra un alto *speed up* pero un pobre rendimiento de nodo único).

3. **Solaris MC**: Tanto NFS como AFS son sistemas comerciales maduros. Si están disponibles los recursos, una tercera aproximación es modificar el sistema de ficheros de la estación de trabajo para crear una jerarquía de ficheros única. Afortunadamente, muchos sistemas proporcionan una interfaz que permite añadir un nuevo sistema de ficheros al kernel sin necesidad del código fuente del kernel. Esta aproximación tiene la ventaja de que se puede ajustar el sistema de

ficheros a las necesidades particulares. AFS se implementó originalmente con esta aproximación. El lado negativo es que el desarrollo de un sistema de ficheros requiere muchos meses de trabajo. El Solaris MC ofrece un ejemplo de esta aproximación.

- *Espacio de memoria, Red y Entrada/Salida únicos*: Para alcanzar SSI, se requiere un *punto de control único,* el administrador del sistema debe ser capaz de configurar, testear, revisar y controlar el *cluster* completo y cada nodo individual desde un único punto de control, este es uno de los asuntos de mayor interés en la construcción de un sistema de *cluster*.

E*spacio de direcciones único,* que da al usuario la ilusión de una gran memoria principal y centralizada, que puede ser en realidad un conjunto de memorias locales distribuidas, PVPs (*Parallel Vector Processors*), SMPs (*Symmetric Multiprocessors*) y DSMs (*Distributed Shared Memory Multiprocessors*), que tienen ventaja sobre MPPs (*Massively Parallel* Processors) y sobre los *cluster*s en este aspecto, ya que permiten que un programa utilice todas las memorias locales o globales.

Trabajo en red único, un *cluster* bien diseñado debería comportarse como un sistema. En otras palabras, sería como una gran estación de trabajo, cualquier proceso en cualquier nodo puede usar cualquier red y dispositivo de entrada/salida como si estuvieran unidos al nodo local, cualquier nodo puede acceder a cualquier conexión de red. El último objetivo de un SSI es que un *cluster* sea tan fácil de usar como una estación de trabajo. Algunos de los tipos de imagen de sistema único que estan presentes en una estación de trabajo son , *el sistema de gestión de trabajos único,* todos los trabajos del *cluster* se pueden someter de un nodo a un sistema de gestión de trabajos único, hay muchos de tales sistemas disponibles actualmente; *interfaz de usuario única,* los usuarios deben ser capaces de usar el *cluster* mediante una única interfaz gráfica, esta interfaz está disponible para estaciones de trabajo y PCs; y *espacio de procesos único,* todos los procesos de usuario, no importa en que nodo residan, pertenecen a un espacio de procesos único y comparten un esquema de identificación de procesos uniforme.

3. *Gestión del trabajo*

Los *cluster*s intentan alcanzar una alta utilización del sistema. Es necesario un software de gestión del trabajo para proporcionar equilibrio de la carga, procesamiento paralelo, trabajo por lotes, y otras funcionalidades. La gestión del trabajo también se conoce como gestión de la carga de trabajo, división de la carga, o gestión de la carga.

Un Sistema de gestión de la carga (*JMS, Job Management System*) debería tener tres partes:

- Un servidor de usuario que permita al usuario meter los trabajos en una o más colas, especificar requisitos de recursos para cada trabajo, borrar un trabajo de una cola, y pedir información sobre el estado de un trabajo o una cola.
- Un planificador de trabajos que realice la planificación de trabajos y mantenga las colas de acuerdo a los tipos de trabajos, requisitos de los recursos, disponibilidad de recursos y políticas de planificación.

- Un director de recursos que reparta y controle los recursos, haga cumplir las políticas de planificación, y recoja información.

Funcionalidades adicionales incluirían la migración de trabajos y los puntos de chequeo.

Las funciones de un JMS se realizan, a menudo, de forma distribuida. Aunque, la administración debe ser centralizada, y debería existir una interfaz única para usarlo. El JMS debería reconfigura dinámicamente el *cluster* con un impacto mínimo en los trabajos en ejecución.

Los distintos tipos de trabajos que se ejecutan en un *cluster* son: *trabajos en serie*, que se ejecutan en un único nodo; *trabajos en paralelo*, que usan múltiples nodos; *trabajos interactivos*, que son aquellos que requieren un tiempo rápido de operación, y su entrada/salida se dirige al terminal, no necesitan grandes recursos y los usuarios esperan ejecutarlos inmediatamente, sin esperas en una cola; *trabajos por lotes*, que normalmente necesitan más recursos, tales como un gran espacio en memoria y gran tiempo de CPU, pero no necesitan una respuesta inmediata, los trabajos se meten en una cola para ejecutarlos cuando existan recursos disponibles. Además de estos trabajos, llamados trabajos de *cluster*, hay otros, llamados, *trabajos locales*, que no se presentan al JMS y que requieren un tiempo de respuesta rápido, el propietario querrá que todos los recursos de las estaciones de trabajo estén dedicados a su trabajo, como si los trabajos de *cluster* no existieran.

El tema de planificación de trabajos es similar al de planificación de procesos. Además, los trabajos se deben planificar para ejecutarse en un tiempo específico (planificación de calendario) o cuando ocurra un evento particular (planificación de eventos). Ya que los requisitos de los recursos de usuario son complejos, un JMS debe tener una interfaz de planificador flexible para implementar las complejas reglas de planificación.

Actualmente están en uso más de veinte paquetes JMS, tanto comerciales como de dominio público, algunos ejemplos son:

- *DQS (Distributed Queueing System)*: es el paquete de dominio público más maduro y usado, se creó y se mantiene actualmente por la Universidad del estado de Florida. Tiene una versión comercial llamada *Codine*, que se ofrece por GENIAS en Alemania. GENIAS intenta convertir a *Codine* en el estándar europeo.
- *LSF (Load Sharing Facility)*: de *Computing Corporation* en Canadá, es probablemente el más usado, con más de 20.000 licencias en todo el mundo

Estos paquetes tienen las siguientes características, entre otras:

- Pueden soportar *cluster*s heterogéneos
- No requieren hardware o software adicional
- Proporcionan distintos tipos de colas de trabajos
- Todos los paquetes ofrecen alguna clase de mecanismo de equilibrio de carga para utilizar de forma eficiente los recursos del *cluster*
- Algunos soportan puntos de chequeo

4. *Comunicación eficiente*

Es más desafiante desarrollar un sub-sistema de comunicación eficiente para un *cluster* que para un MPP, por las siguientes razones:

- Debido a la más alta complejidad de los nodos. Los nodos de un *cluster* no se pueden empaquetar de forma compacta como los de un MPP
- Las longitudes de los cables físicos entre los nodos son más largas en un *cluster* que en un MPP. La longitud del cable implica mayores latencias de la red de interconexión. Pero, lo más importante, los cables más largos tienen más problemas de fiabilidad, lo que implicará, por ejemplo, usar protocolos que permitan una comunicación segura y fiable, pero que añaden una sobrecara.
- Los *cluster*s usan a menudo las redes comerciales como Ethernet o ATM, con protocolos de comunicación estándares tales como TCP/IP. Pero implican una alta sobrecarga. Otros protocolos de mas bajo nivel, aunque más eficientes, no son actualmente aceptados [37].

2.6. *CLUSTER*S DE SERVIDORES Y ESTACIONES DE TRABAJO

2.6.1. Aparición y evolución

En los últimos años, hemos visto un aumento en el número de productos y sus aplicaciones en empresas, negocios, e instituciones de investigación. Algunos de los sistemas en *cluster* actuales son:

- Wolfpack de Microsoft
- POWER CHALLENGEarray de SGI
- El *cluster* tolerante a fallos Marathon
- SP de IBM
- Tru*Cluster* de Digital
- El proyecto NOW de Berkeley
- El *cluster* DSM de Rice TreadMarks.

Actualmente hay sobre unos 100.000 *cluster*s de computadoras en uso en todo el mundo, incluyendo tanto productos comerciales como de diseño específico. Los nodos en estos *cluster*s son en su mayoría computadoras personales, estaciones de trabajo y servidores SMP. Los tamaños de los *cluster*s suelen estar en el orden de diez, y sólo unos poco exceden de 100 nodos. Muchos *cluster*s usan redes como Fast o Gigabit Ethernet, anillos FDDI, conmutadores ATM o Myrinet para interconectar los nodos aparte de las conexiones LAN regulares entre ellos. Solo unos pocos, como el SP2 de IBM y el NT de Compaq, usan redes de alto ancho de banda patentadas. Hay disponibles algunos paquetes software comerciales y de dominio público para soportar imagen de sistema único, mayor disponibilidad, y gestión de trabajos distribuidos. La tecnología de *cluster* básico existe, y empieza a crecer el volumen de producción, para aplicaciones de negocios. La tendencia en los *cluster*s se mueve de los grandes mainframes a las estaciones de trabajo Unix y a los sistemas con un elevado número de computadoras personales. Algunos de los *cluster*s comerciales se listan en la siguiente tabla (Tabla 2.2) [37].

Buenos ejemplos de *cluster*s de estaciones de trabajo son los SPARC*cluster*s de Sun, los *cluster*s computacionales Apollo 9000 de HP, Sequent Symmetry 5000, las de estaciones de trabajo Alpha de Digital, etc. Las tendencias más actuales van dirigidas

hacia los *cluster*s de computadoras personales basadas en procesadores Intel. Esta tendencia se dispara por el uso generalizado de estas computadoras y de las estaciones de trabajo de bajo costo. Desde IBM y Digital hasta Sun, SGI, HP, Compaq, y Microsoft, la industria completa está entusiasmada con los huecos de los *cluster*s.

Los productos de los *cluster*s aparecen como sistemas integrados, herramientas software, infraestructura de disponibilidad, extensiones de sistemas operativos, etc. Los *cluster*s de alto rendimiento se construyen con sub-sistemas de comunicación especialmente diseñados, incluyendo redes de alto rendimiento, interfaces de red, y software de comunicaciones. Dos ejemplos son el Tru*Cluster* de DEC que utiliza técnicas de canal de memoria y el SP2 de IBM que emplea un conmutador de alto rendimiento. Algunas redes de estaciones de trabajo ya en uso de producción son:

- **RTCPP** (*Regional Training Center for Parallel Processing*, Universidad del estado de Carolina del Norte)
- **NAS** (McDonnell Douglas, Pratt & Whitney, Boeing)
- **PEIRS** (*Parallel Integrated Environment for Reliability of Structures of existing industrial plants*)

Tabla 2.2. Una muestra de los productos de *cluster*s comerciales

COMPAÑÍA	NOMBRE	BREVE DESCRIPCIÓN
DEC	VMS-clusters	*Cluster*s altamente disponibles para VMS
	TruClusters	*Cluster* Unix de servidores SMP
	NT Clusters	*Cluster*s basados en Alpha para Windows NT
HP	Apollo 9000 Cluster	*Cluster* computacional
	MC/ServiceGuard	NetServer de HP para soluciones de *cluster* NT
IBM	Sysplex	*Cluster* de mainframe de disco compartido
	HACMP	*High-Availability Cluster Multiprocessor*
	Scalable POWER Parallel (SP)	*Cluster* con nodos POWER2 y conmutador Omega
MICROSOFT	Wolfpack	Estándar abierto para *cluster*s de servidores NT
SGI	POWER CHALLENGEarray	Escalable de servidores SMP y conmutador HiPPI
SUN	Solaris MC	Extensión del *cluster* de Sun Solaris
	SPARCcluster 1000/2000 PDB	Alta disponibilidad, para OLTP y bases de datos
TANDEM	Himalaya	Altamente escalable y tolerante a fallos
MARATHON	MIAL2	Alta disponibilidad con redundancia completa

Los *cluster*s son un campo activo de investigación. Algunos de esos proyectos se muestran en la siguiente tabla (Tabla 2.3), en la podrían aparecer muchos más, que están actualmente en desarrollo. Cada uno de los proyectos ha desarrollado alguna característica única digna de mencionar. Las características cubren innovaciones en la arquitectura así como soporte software para imagen de sistema único, disponibilidad, y un banco de pruebas de aplicaciones distribuidas con recursos Internet o/y Intranet.

Otros proyectos son [62]:

- **ATOMIC-2** (Donde nació Myrinet, y la siguiente etapa...)
- **AVALANCHE** (HP, Myrinet, unidades de comunicación y cache)
- **DMRC** (*Distributed Multimedia Research Center*)
- **GAMMA** (12 Pentium con linux, mensajes activos y Fast Ethernet)
- **HPVM** (*Cluster* de Pentium con Myrinet, Servernet, etc.)
- **LANL**, *cluster* de estaciones de trabajo (8 RS/6000 más 1 SP2 (2 nodos) intercoenctados con un Giga-conmutador)
- **MPC (**una red francesa de PCs con interconexión a medida)
- **NAS Davinci**, 8 multiprocesadores Power Challenge de SGI (alguno de ellos monoprocesador) conectados por Ethernet, HIPPI, FDDI y ATM
- **PAPERS** (Red de estaciones de trabajo más un hardware específico para operaciones globales rápidas)
- **RWCP PC *Cluster*** (36 PC Pentium 166 más Myrinet)
- **RWCP WCS** (36 SuperSparc 20 más Myrinet)
- **StarT-Voyager** (*cluster*s de PPC 604, NES : *Network Endpoint Subsystem* y router Arctic)

Tabla 2.3. Proyectos de investigación representativos de *cluster*s [37].

PROYECTO/REFERENCIA	CARACTERÍSTICAS
Berkeley Proyecto **NOW**	Una red de estaciones de trabajo caracterizada con mensajes activos, sistemas de ficheros cooperativos, y extensiones de Unix globales
Princeton **SHRIMP**	Componentes comerciales, comunicación eficiente y memoria virtual compartida mediante interfaces de red especiales
Karsruhe **Parastation**	Red de comunicación eficiente y desarrollo de software para procesamiento paralelo distribuido
Rice **Treadmarks**	*Cluster* de estaciones de trabajo de memoria compartida distribuida implementada en software
Wisconsin **Wind Tunnel**	Memoria compartida distribuida en un *cluster* de estaciones de trabajo interconectadas con una red comercial
El **NSCP** http://ncsp.upenn.edu	*National Scalable Cluster Project*, para metacomputación sobre tres *cluster*s locales enlazados por Internet
Argonne **Globus**	Desarrollan una plataforma de metacomputación y software mediante una WAN conectada por ATM de 17 puestos en Norte América
Syracusa **WWVM**	Una máquina web virtual para alto rendimiento para computación de alto rendimiento con tecnologías Internet y HPCC
Universidad de Hong-Kong **PearlCluster** http://pearl.cs.hku.hk	Un *cluster* de investigación para aplicaciones de libraría digital financiera y multimedia
Virginia **Legion**	Desarrollo de software de metacomputación para un servicio de computadora virtual nacional

El Proyecto **NOW** (*Network Of Workstations*) [3] cubre casi todo el espectro de temas sobre computación con *cluster*s, incluyendo arquitectura, soporte software para servidores Web, imagen de sistema único, sistema de ficheros y entrada/salida,

comunicación eficiente, y disponibilidad realzada. Lo veremos un poco más en detalle en la siguiente sección.

En la siguiente tabla (tabla 2.4) enfrentamos algunos clusters para comparar sus características:

Tabla 2.4. Comparación de *Cluster*s [7]

Proyecto	**Plataforma**	**Comunicacones**	**S.O.**	**Otros**
Beowulf	PCs	Ethernet múltiple con TCP/IP	Linux y Grendel	Sockets MPI/PVM y HPF
NOW	PCs basados en Solaris y estaciones de trabajo	Myrinet y mensajes activos	Solaris +GLUunix+ XFS	AM, PVM, MPI, HPF, Split-C
HPVM	PCs	Myrinet y mensajes rápidos	Linux o NT + LSF	FM, Sockets, Arrays globales, SHMEN y MPI
Solaris MC	PCs basados en Solaris y estaciones de trabajo	Soportadas por Solaris	Solaris + capa de globalización	C++ y Corba

Los supercomputadores clase Beowulf son muy conocidos por su potencia computacional. Un ejemplo claro fue la demostración en Linux Expo donde se comparó un Cluster beowulf con un Cray T3t 900-AC64 de 5.5 millones de dóalres. Esta máquina tenía el record mundial de potencia computacional ejecutando PovRay, una aplicación de trazado de rayos que emplea gran cantidad de cálculos en coma flotante y de operaciones con grandes bloques de memoria. Un supercomputador clase Beowulf con un costo hardware de ciento cincuenta mil dólares, es decir, un tercio del costo del Cray, montado por IBM, generó la escena tres veces más rápido. El Beowulf era tres veces más potente y al mismo tiempo tres veces más barato, por lo que la relación costo/potencia de cálculo era nueve veces mejor en el beowulf que en el Cray T3 [74].

2.6.2. El proyecto NOW de Berkeley

El proyecto de redes de estaciones de trabajo de la Universidad de California en Berkeley, busca aprovechar el poder de maquinas agrupadas y conectadas mediante redes conmutadas de alta velocidad. Influenciado por las estaciones de trabajo y los sistemas operativos comerciales, NOW puede conseguir un incremento en el rendimiento de la industria. El asunto clave para NOW es la llegada de la agresiva red de alto ancho de banda basada en conmutador. Esta evolución tecnológica permitirá soportar una variedad de cargas de trabajo distintas, incluyendo trabajos paralelos, secuenciales e interactivos así como servicios web escalables, que incluyen el motor de búsqueda web más rápido del mundo, y trabajos comerciales como NOW-Sort. NOW alcanzó sobre los 10 Gflops en el banco de pruebas LINPACK, impulsando la NOW al tope de las 200 supercomputadoras más rápidas del mundo [63].

Con este proyecto se quieren desarrollar técnicas para *cluster*s de empresas, aplicables a *cluster*s dedicados. La motivación y objetivo son que el sistema debe proporcionar al usuario individual el tiempo predecible y rápido de una estación de

trabajo dedicada, y permitiendo que las tareas demasiado largas para un puesto recluten los recursos del *cluster*.

El proyecto NOW trata la mayoría de los temas de interés del agrupamiento, discutidos en la sección anterior. Se consigue comunicación eficiente utilizando redes del orden de gigabits comerciales y el protocolo de comunicación de mensajes activos. La imagen de sistema único, la gestión de recursos, y la disponibilidad se consiguen con el software GLUnix a nivel de usuario. Se desarrolló un nuevo sistema de ficheros en red *sin servidor*, llamado *xFS*, para soportar la jerarquía de fichero única, escalable y altamente disponible. También han construido un marco de trabajo software, llamado *WebOS*, para servicios web, incrementalmente escalables y altamente disponibles.

- *Mensajes activos para comunicaciones rápidas:*

Los mensajes activos son un mecanismo de comunicación asíncrono para realizar comunicaciones con baja sobrecarga. Su objetivo es exponer al usuario la capacidad en bruto del hardware de comunicación subyacente. La idea básica es usar la información de control en la cabecera de un mensaje como un puntero a una subrutina a nivel de usuario llamada manipulador de mensajes (*message handler*). Cuando la cabecera del mensaje llega al nodo destino, se llama al manipulador de mensajes para extraer el resto del mensaje de la red e integrarlo en los cálculos en curso.

El mecanismo de mensajes activos tiene una especificación genérica y diferentes implementaciones. Hay algunas sugerencias para los mensajes activos originales para proporcionar funcionalidad adicional. La versión 1 de la especificación hace uso de un pequeño número de funciones (primitivas) basadas en las siguientes suposiciones:

- Los mensajes activos son una capa software usada para soportar comunicaciones entre los procesos de una aplicación SPMD
- El programa consiste de n procesos, cada uno de ellos numerado por un *número de nodo virtual* (VNN) de 0 a n-1. La creación y gestión de procesos se proporciona por alguna facilidad del sistema (como por ejemplo, GLUnix) externa a la capa de mensajes activos.

La capa de mensajes activos que cumple la especificación de mensajes activos genéricos se denomina capa GAM. Esta capa proporciona cinco primitivas básicas, y un número de funciones de utilidades. Hay dos tipos de mensajes en GAM: *request* y *reply*.

Son un mecanismo general, que no requiere ninguna plataforma hardware o software. Han sido implementados en MPPs, *cluster*s de estaciones de trabajo, y en PVM. Sin embargo, el significado de los mensajes activos es que tienen que ser usados como una capa de comunicación de bajo nivel que puede dar un gran porcentaje del rendimiento puro del hardware de comunicaciones.

El buen rendimiento de los mensajes activos se atribuye a las siguientes razones:

- Se realizan a nivel de usuario
- Simplicidad
- Solapamiento entre computación y comunicación

- *GLUnix para la gestión de recursos globales:*

Los *cluster*s necesitan funcionalidad del sistema operativo para proporcionar disponibilidad e imagen de sistema único, que están ausentes en los sistemas operativos tradicionales de las estaciones de trabajo.

Ha habido muchas innovaciones en los sistemas operativos, aunque pocas se han incorporado a los sistemas operativos comerciales. Lo que indica que un sistema operativo se adquiere no por su funcionalidad, sino por las aplicaciones que soporta, la combinación de costo/rendimiento del hardware/sistema operativo, y la robustez del sistema. Se han evaluado las aproximaciones actuales y se ha llegado a la conclusión de que todos tienen limitaciones.

La idea principal de GLUnix (*Global Layer Unix*) es que el sistema operativo de un *cluster* debería consistir de dos capas:

- La capa más baja es el sistema operativo comercial de los nodos, que se ejecuta en modo kernel (o microkernel).
- La capa más alta es el sistema operativo a nivel de usuario y que proporciona las nuevas características que necesita un *cluster*.

Estas capas proporcionan una imagen de sistema único de los nodos en un *cluster*, así que todos los procesadores, memoria, capacidad de red, y disco se pueden asignar a aplicaciones paralelas y secuenciales.

Esta aproximación hace posible que muchas características sean implementadas a nivel de usuario. La aproximación GLUnix tiene algunas ventajas: fácil de implementar, portabilidad, eficiencia y robustez.

Se ha implementado un prototipo de GLUnix que proporciona muchas de las siguientes características consideradas importantes por los *cluster*s de empresa:

- Co-planificación de programas paralelos
- Detección de recursos desocupados, migración de procesos y equilibrio de la carga
- Comunicación a nivel de usuario rápida
- Paginación remota
- Soporte para disponibilidad

El prototipo proporciona un conjunto de herramientas de utilidades similares a las que se encuentran en LSF. La idea parece muy atractiva. Sólo el tiempo dirá si la idea a nivel de usuario es suficientemente flexible para proporcionar toda la funcionalidad necesaria para los *cluster*s.

- *El Sistema de Ficheros sin servidor xFS*

Un sistema de ficheros sin servidor distribuye las funciones de un servidor de ficheros entre todos los nodos de un *cluster*. Un servidor de ficheros centralizado tradicional realiza las siguientes funciones principales:

- *Almacenamiento central*: los ficheros de datos y los metadatos (atributos de fichero como tipo, tamaño, número de inodo, propietario, etc.) se almacenan en uno o más discos unidos al servidor de ficheros.
- *Gestión centralizada de la cache*: para mejorar el rendimiento, cada cliente puede utilizar la cache localmente con algunos bloques de ficheros. Además, el

servidor de ficheros utiliza la cache con los bloques de ficheros más usados en su memoria principal para satisfacer algunas pérdidas de los clientes

- *Gestión centralizada*: el servidor realiza todas las funciones de gestión del sistema de ficheros, lo que incluye gestión para los metadatos y consistencia de cache. En xFS todas las funciones de los clientes y del servidor se realizan por todos los nodos de forma distribuida.

Una idea característica en xFS es la utilización de la cache de forma cooperativa. Cada nodo cliente de un *cluster* asigna una porción de su memoria como cache de ficheros. Un algoritmo cooperativo utiliza todas esas memorias para crear una gran cache de fichero en todo el *cluster*. Cuando un cliente se encuentra con una pérdida de la cache de fichero local, no va al disco como se haría en una estación de trabajo convencional, va a otra memoria de cliente a coger los datos.

Los ficheros se almacenan en el disco del servidor. Tradicionalmente, los ficheros se guardan en la cache de la memoria del cliente (la mayoría de los sistemas), en el disco local del cliente (como en AFS) y en la memoria del servidor (la mayoría de los sistemas). XFS permite que un fichero se guarde en la cache de un cliente remoto también [37].

2.7. TECNOLOGÍAS DE *CLUSTER* FUTURAS

2.7.1. Introducción

Las nuevas tecnologías hardware junto con los recursos software indican que los sistemas basados en *cluster* se acercan rápidamente al rendimiento ofrecido por plataformas de computación paralelas dedicadas. Los *cluster*s dedicados a aplicaciones de alto rendimiento continuarán su desarrollo conforme estén disponibles en el mercado interfaces de red y computadoras cada vez más evolucionadas y poderosas. Se desarrollará software que permita usar eficientemente las aplicaciones paralelas. Es probable que haya un uso extendido de Gigabit Ethernet, y así surgirán nuevos estándares para los *cluster*s. Para reducir las latencias del paso de mensajes, se pasará por encima del kernel del sistema operativo y se evitará la necesidad de las costosas llamadas al sistema, y se explotará el uso de tarjetas de red inteligentes.

La habilidad de proporcionar un amplio conjunto de herramientas de desarrollo y de utilidades así como de servicios seguros y rentables, determinará la elección del sistema operativo que se usará en los *cluster*s futuros. Los sistemas operativos basados en UNIX son probablemente los más populares, pero teniendo en cuenta las mejoras y aceptación de Windows NT [7].

No podemos dejar de referenciar a InfiniBand como una propuesta cada vez más real de bus de alta velocidad que hará posible la creación de clusters y SANs cada vez más rápidos. InfiniBand proporciona una solución escalable a las siempre crecientes demandas de los sistemas servidor. Las capacidades de ancho de banda de InfiniBand son suficientes para las tecnologías de comunicación más rápidas. Y, además este ancho de banda puede escalarse añadiendo enlaces adicionales entre los dispositivos, sin tener que recurrir a una solución más cara o a una implementación completamente diferente [77].

La arquitectura InfiniBand está siendo desarrollada por la *Infiniband™ Trade Association* (IBTA) [39] que publicó la primera especificación de la misma en octubre del 2000 [38]. La alternativa que se propone, con respecto a los buses estándares, es permitir la interconexión de procesadores entre sí y con dispositivos de entrada/salida. La arquitectura de Infiniban incluye mecanismos que pueden servir para proveer de calidad de servicio a las aplicaciones multimedia [2]. Este es un campo de gran interés actualmente. Un capítulo completo sobre esta arquitectura se puede encontrar en [42]

2.7.2. Tecnologías emergentes

La industria de las computadoras/comunicaciones está sufriendo uno de los cambios más intensos y drásticos de su corta historia. En los últimos años, los avances logrados en el desarrollo de computadoras de alta velocidad y bajo costo ha hecho surgir muchas posibilidades de crear nuevas y potentes aplicaciones.

Las respuestas positivas de los usuarios a tales sistemas y la demanda de más funciones en sus aplicaciones está preparando el camino para aplicaciones aún más potentes y productivas. Lo que ha sido suficiente en el pasado no bastará en el futuro. El creciente uso de, y experimentación con, aplicaciones que integran voz, vídeo y gráficos puso de manifiesto que las aplicaciones del futuro necesitaran redes que operarán con velocidades del orden de multimegabits por segundo.

La revolución en la industria de las computadoras personales dio origen a una amplia variedad de nuevas aplicaciones que funcionaban bien con la tecnología LAN, la cual se expandió rápidamente. Al tiempo que estas LAN se interconectaban para compartir recursos e intercambiar datos, las compañías telefónicas estaban mejorando también su tecnología de área extensa instalando fibras ópticas en sus líneas troncales. Instalaron también equipos más rápidos y suministraron mejores servicios a los usuarios. Desafortunadamente, parte de las labores de interconexión de LAN se efectuaron sin pensar en la necesidad de utilizar protocolos estandarizados entre el hardware y el software de los diferentes fabricantes. La consecuencia fue que muchos sistemas que se venden actualmente no pueden comunicarse fácilmente entre sí, aunque proporcionen servicios similares. Para un número creciente de usuarios y proveedores de telecomunicaciones, los sistemas instalados en los años ochenta parecían ya insuficientes para satisfacer las necesidades de las aplicaciones tanto de los usuarios como de los fabricantes. Por ello, a mediados de la década de los ochenta se iniciaron esfuerzos por atender las necesidades de aplicaciones futuras.

En unos cuantos años, las aplicaciones futuras se han convertido en las aplicaciones presentes. La industria de las comunicaciones se está desplazando hacia las potentes redes de gigabits que soportan cualquier tipo de aplicaciones. Es más, por primera vez en la industria de las computadoras/comunicaciones, las administraciones de redes de todo el mundo están adoptando un conjunto de estándares a nivel mundial para usarlos en redes multimedia de alta capacidad. Las tecnologías emergentes de comunicaciones apoyan redes integradas de alta capacidad que involucran computadoras de distintos fabricantes y paquetes software que operan con diferentes proveedores de servicios de comunicaciones.

Un objetivo importante es ofrecer más capacidad de rendimiento (en bits/s) para aplicaciones de usuario y realizar las operaciones de red con mayor rapidez a favor de

las aplicaciones. Esto implica que las tecnologías emergentes se han diseñado para ofrecer alto rendimiento, con velocidades de transmisión muy altas y con retardos muy bajos. Otra meta importante es apoyar cualquier tipo de aplicación como voz, vídeo, audio, etc. La implementación de las redes multimedia está resultando ser uno de los mayores retos que enfrenta la industria.

En la siguiente tabla se listan varios tipos de aplicaciones que requieren un gran ancho de banda. En general, se trata de aplicaciones de transmisión de vídeo o imágenes fijas de alta calidad a todo color. Hasta hace poco, parecía poco probable encontrar soluciones de red económicas para apoyar estos tipos de aplicaciones, pero se han logrado avances importantes en las técnicas de compresión que permiten a algunas tecnologías actuales manejar ese tipo de aplicaciones.

Tabla 2.5. Aplicaciones y ancho de banda requerido

Descripción del medio	**Velocidades sin compresión (Mbits/s)**	**Velocidades con compresión (Mbits/s)**
Vídeo NTSC con calidad de difusión	120	3-6
Vídeo NTSC con calidad de estudio	216	10-30
Vídeo de alta definición con calidad de difusión	1500	20-30
Visualización de imágenes con pleno movimiento, 60 cps, 1000x1000 pixeles	1500-2500	50-200
Imágenes fijas binarias monocromáticas 600x600 puntos/pulgada,135 pp/min.	120	5-50
Imágenes fijas a todo color 400x400 pixeles/pulgada, 60 pp/min.	500	45

Uno de los problemas que ha acaparado gran atención en los últimos años es la necesidad de interconectar las redes de área local y de área extensa. Se han propuesto varias soluciones:

1. La tecnología de *retransmisión de tramas* (*Frame Relay*) se propone resolver el problema del cuello de botella de las WAN ofreciendo un servicio de paquetes rápido (en realidad un servicio de tramas rápido).

2. Los términos *Ethernet rápida* (*Fast Ethernet*) y *Ethernet conmutada* (*Switched Ethernet*) a menudo se utilizan de manera indistinta. Juntos, estos términos describen una tecnología de LAN diseñada para apoyar individualmente estaciones de trabajo de usuario cuyos requisitos de capacidad varían entre 10 Mbits/s y 100 Mbits/s, en contraste con el esquema de compartir estos anchos de banda con múltiples estaciones de trabajo. También ya la *Gigabit Ethernet* se aprovecha del éxito de Ethernet y Fast Ethernet para ofrecer una LAN con sus mismas características, pero a la velocidad de 1 Gbit/s [24] o ya a 10 Gbit/s [30]

3. La *red de área metropolitana* (*MAN, Metropolitan Area Network*) es un estándar del IEEE (802.6) que ofrece un protocolo de bus dual con cola distribuida (DQDB,

Distributed Queue Dual Bus) muy rápido que apoya redes integradas para aplicaciones multimedia.

4. El *Servicio de Conmutación de Datos Multimedia* (*SMDS, Switched Multimegabit Data Service*) se basa en la tecnología MAN y proporciona un sistema de transporte público de alta velocidad. Las compañías telefónicas en Estados Unidos y varios proveedores de portadoras públicas europeas están ofreciendo SMDS como un servicio para aplicaciones de datos de alta velocidad que requieren ráfagas de transmisión con alto ancho de banda.

5. El *Modo de Transferencia Asíncrono* (*ATM, Asynchronous Transfer Mode*) forma parte de la solución B-ISDN. Se trata de una tecnología de relevo de celdas que incluye servicios de multiplexación y conmutación de alta velocidad para aplicaciones de voz, datos y vídeo.

6. La *Red Óptica Síncrona* (*SONET, Synchronous Optical Network*)/*Jerarquía Digital Síncrona* (*SDH, Synchronous Digital Hierarchy*) es un sistema de portadora de alta velocidad basado en temporización síncrona, en el cual los dispositivos de red se sincronizan con uno o más relojes maestros. Este sistema utiliza tecnología de fibras ópticas y está diseñado para apoyar sistemas de portadora actuales y también tráfico de LAN y de relevo de tramas y celdas.

7. Las *Tecnologías inalámbricas y móvil* son sistemas que suministran recursos de comunicación utilizando ondas de radio de alta frecuencia. En la actualidad, estas tecnologías representan una mejora de la infraestructura de comunicaciones existentes, como los circuitos locales y los servicios telefónicos locales. Obviamente, estas tecnologías ofrecen un mecanismo muy flexible para manejar usuarios móviles, y se están convirtiendo en competidores fuertes de algunas de las otras tecnologías.

8. El término *banda ancha residencial* se acuñó para describir una amplia variedad de tecnologías que llevan más ancho de banda al circuito del suscriptor local (al hogar) para apoyar aplicaciones como vídeo bajo demanda, transferencia de archivos grandes y navegación interactiva en Internet.

9. La *Señalización de Banda Ancha* es una tecnología utilizada por una red para establecer los recursos (conexiones) que apoyan a las necesidades del usuario final. Las diversas organizaciones de normas y *cluster*s comerciales definen una manera distinta el término *banda ancha*. Por lo general se suele usar como el término para describir cualquier sistema de alta capacidad (un sistema de megabits/s) que ofrece un sistema de transporte para multimedia.

10. La *Red Inteligente* (*IN, Intelligent Network*) es una extensión y una evolución de la arquitectura SS7 diseñada para proporcionar un conjunto estandarizado de protocolos (que operan encima de SS7) para ayudar al suministro rápido de servicios (funciones de llamada, por ejemplo) al usuario.

La tendencia en las redes públicas y privadas ha sido tratar de proporcionar servicios más rápidos al usuario. El cimiento de buena parte de las tecnologías emergentes es la fibra óptica. Una serie de avances realmente extraordinarios en los

últimos quince años han abaratado los costos y aumentado la capacidad de la fibra óptica. Hace varios años, un sistema de 40 Mbits/s se consideraba de muy alta velocidad. Hoy día son muy comunes los sistemas comerciales de 2.5 Gbits/s, y ya se están instalando sistemas de 10 Gbits/s.

En ningún lugar es más evidente el avance de la tecnología que en la creciente potencia de la unidad central de proceso (CPU) y en la velocidad de la memoria de las computadoras. La industria ha logrado también avances notables en la reducción de los costos de las computadoras, lo que ha propiciado el uso de tecnología más avanzada y aplicaciones más potentes.

Parece razonable llegar a las siguientes conclusiones debido al ímpetu y a la fuerza impulsora para las nuevas tecnologías:

- Las aplicaciones más potentes tienen un efecto directo y positivo sobre la productividad humana
- Está comenzando a haber un suficiente ancho de banda para apoyar estas aplicaciones.
- La reducción de costo de este ancho de banda permitirá expandir el uso de circuitos y conmutadores de alta velocidad para apoyar las aplicaciones.
- La creciente potencia de las computadoras apoyará nuevas aplicaciones
- La reducción en los costos de cómputo hará económica la implementación de aplicaciones basadas en gigabits.

Por tanto, podemos decir que las redes más potentes son indispensables para continuar nuestro progreso tecnológico y para nuestra productividad [6].

Queda claro, por tanto, que la evolución de las redes actuales de telecomunicación va orientada hacia una red integrada de comunicación de banda ancha conocida normalmente como la Red Digital de Servicios Integrados de Banda Ancha (B-ISDN, *Broadband Integrated Services Digital Network*) por considerarse una extensión lógica de la actual Red Digital de Servicios Integrados (ISDN, *Integrated Services Digital Network*). La dirección actual que está tomando esta red está influenciada por numerosos parámetros, el más importante de los cuales es la aparición de un gran número de servicios con diferentes exigencias. Esta gran cantidad de exigencias introduce la necesidad de una red universal suficientemente flexible, influenciada además por la evolución de los semiconductores, de la tecnología óptica y del concepto de sistema. Todo ello llevó a la definición de ATM, aceptado como solución para la B-ISDN por la ITU-T y por el ATM Forum [69].

¿Qué aplicaciones se supone que van a apoyar estas nuevas tecnologías? En la tabla siguiente (tabla 2.6) presentamos un resumen de las principales aplicaciones objetivo.

Tabla 2.6. Aplicaciones

• *Relevo de tramas y SMDS* Redes de área local y redes de área extensa “sin costuras” Gráficos, CAD/COM y rayos X, transferencia de grandes bases de datos Interconexiones de grandes redes de área local, Vídeo de calidad media, Transferencia de datos Enfocada hacia aplicaciones de datos de alto ancho de banda
• *ATM y MAN* Todas las anteriores y además voz y vídeo de alta calidad
• *Ethernet rápida y conmutada* Reemplazo de las LAN de bus compartido Enfocadas hacia datos (algo de voz, pero limitado)
• *SONET/SDH* Cualquier cosa; no es más que una tecnología de portadora física, un servicio de transporte para ATM, SMDS, relevo de tramas, T1, E1, etc.
• *Inalámbrico y móvil* Cualquier aplicación de datos de baja velocidad, voz y vídeo de baja a mediana calidad
• *Banda de ancha residencial* Vídeo bajo demanda, aplicaciones cliente-servidor, navegación de Internet en el circuito local
• *Señalización de banda ancha* Apoya todas las aplicaciones estableciendo conexiones de alta capacidad en una red
• *Red Inteligente* Apoya funciones avanzadas para aplicaciones de voz, vídeo y datos

2.7.3. Redes de alta velocidad

La implementación de una red de alta velocidad implica mirar más de cerca aquellos detalles que van a influir en el rendimiento global. Pequeñas modificaciones en el hardware pueden mejorar considerablemente las prestaciones de nuestra red. En general la norma a seguir es: *mantener el caso más frecuente lo más simple y rápido posible* [22].

Las redes de alta velocidad son de suma importancia para trabajar eficazmente en un entorno distribuido. La importancia cada vez mayor de las LANs y la creciente complejidad de las aplicaciones requieren redes de alta valocidad. El ancho de banda proporcionado por una conexión Ethernet no es adecuado para la carga de un sistema distribuido ya sea en el mundo científico como en el empresarial. Se necesita la aproximación de red más especializada, de menor costo, con intercoenxiones de alto rendimiento que realcen tanto el rendimiento paralelo como la alta disponibilidad, de un *cluster*. Algunas tecnologías que persiguen este objetivo son Fast/Giga Ethernet, HiPPI (*High Performance Parallel Interface*), ATM, SCI (*Scalable Coherent Interface*), ServerNet, Myrinet, *Memory Channel*, y Sinfinity, que veremos más adelante.

La red es la parte más crítica de un *cluster*. Su capacidad y rendimiento afecta directamente al sistema global. Veamos algunos temas de diseño general para redes de alta velocidad y después la interconexión en algunos *cluster*s conocidos.

Vamos a describir, a continuación, las tendencias principales para interconectar el hardware, ya que hay que tomar algunas decisiones cuando se realiza el diseño de la interconexión de un *cluster*. Lo más importante es indudablemente el equilibrio entre el precio y el rendimiento, y también importantes son la escalabilidad y la fiabilidad.

- *Protocolo de enlace*: este término se usa para los formatos de los mensajes que se transmiten sobre la capa física y la interacción entre puntos finales del enlace de comunicación.

- *Capa física*: en la elección del medio físico correcto de un canal hay que tener en cuenta el caudal de datos y el costo del cable. Los medios de serie ofrecen un ancho de banda moderado pero pueden confiar en una capa de transporte a nivel de enlace estándar tal como la Gigabit Ethernet o el canal de fibra. Una red HiPPI puede usar un cable de 64 bits que habilita caudales de datos muy altos. Pero la cantidad de pines es la limitación para la implementación de los conmutadores. Se puede utilizar la técnica LVDS (*Low Voltage Data Signaling*) para transmitir señales a 100 MHz o más, con un consumo razonable. Un conmutador unidireccional 8x8 con líneas de señales diferenciales de 32 bits, necesitaría 1024 pines solo para los enlaces, lo que es demasiado para la fabricación actual. Los enlaces de un byte de ancho son un buen compromiso, como los usados en Myrinet y ServerNet. Pueden construirse conmutadores de tamaños moderados, aunque los caudales de datos exceden todavía mucho a los medios de serie. Ya que la longitud de transmisión eléctrica es muy restrictiva, las capas ópticas pueden ser una buena opción de futuro, aunque los costos actuales impiden su uso en muchos casos.

- *Conmutación*: es la conexión entre los puertos de entrada-salida y la transferencia de datos entre ellos dentro de una red. Hay dos técnicas principales que se usan en las redes actuales que son la conmutación de paquetes y wormhole.

La conmutación de paquetes almacena un paquete completo en un conmutador de la red antes de que se envíen los datos a la siguiente etapa. Este mecanismo implica un límite superior en el tamaño del paquete y espacio de buffer para almacenar uno o varios paquetes de forma temporal. Las redes LAN/WAN más tradicionales (Fast Ethenet, ATM) usan conmutación de paquetes. Otras más actuales como Myrinet usa conmutación wormhole donde los datos se envían inmediatamente a la siguiente etapa tan pronto como la cabecera de dirección se decodifica. Las ventajas de esta técnica son la baja latencia y la necesidad de poca cantidad de espacio de buffer, aunque la corrección de errores es más difícil.

- *Encaminamiento*: la cabecera del mensaje lleva la información de dirección necesaria para el hardware de enrutamiento interior del conmutador para determinar el canal de salida correcto. Se han propuesto muchos algoritmos de enrutamiento adaptativos y deterministas. Los adaptativos tratan de encontrar caminos alternativos por la red de forma dinámica en caso de caminos sobrecargados o enlaces rotos, aunque

no han encontrado aún su forma en hardware real. Dos mecanismos utilizados en la interconexión actual son el basado en tablas y el encaminamiento fuente.

- *Control de flujo:* ya lo hemos comentado en una sección anterior. Se utiliza para evitar inundar los buffers interiores del conmutador, que puede implicar pérdidas de datos. Antes de que el emisor comience la transmisión, el receptor debe señalar la habilidad para recibir los datos. Una posible solución es el esquema basado en créditos. Por ejemplo, Myrinet inserta bytes de control STOP & GO para parar y reanudar la transmisión de datos por el emisor. La siguiente tabla (Tabla 2.7) resume los diferentes parámetros para algunas interconexiones.

Tabla 2.7. Parámetros de interconexión

Interconexión	**Caudal de datos unidireccional**	**Conmutación**	**Encaminamiento**
Fast Ethernet	100 Mbits/s	Paquetes	Basado en tabla
Gigabit Ethernet	1 Gbits/s	Paquetes	Basado en tabla
10Giga Ethernet			
Myrinet	2.28 Gbits/s	Wormhole	Encaminamiento fuente
ServerNet II	125 Mbyte/s	Wormhole	Basado en tabla
Memory Channel	100 Mbyte/s	Paquetes	Basado en tabla
Synfinity	1.6 Gbyte/s	Wormhole	Encaminamiento fuente
SCI	400 Mbyte/s	Paquetes	Basado en tabla
ATM (OC-12)	155(622) Mbit/s	Paquetes	Basado en tabla
HiPPI	800 Mbit/s	Paquetes	Basado en tabla

- *Detección y corrección de errores:* aunque las capas físicas actuales tienen caudales de error muy bajos, la red debe ofrecer algunos mecanismos para detección y posible corrección de errores en hardware.

- *Transferencia de datos:* la transferencia de datos eficiente entre la memoria principal y el interfaz de red es un factor crítico a la hora de lograr que las aplicaciones se aproximen al ancho de banda real. Para lograrlo, el software de los interfaces de red actuales hacen intervenir al sistema operativo solo cuando las aplicaciones abren o cierran el dispositivo de red. Las transferencias de datos reales se realizan completamente en modo usuario mediante bibliotecas de rutinas que evitan el coste de las llamadas al sistema operativo. El objetivo es un mecanismo sin copia, donde los datos se transfieren directamente entre el espacio de usuario en memoria principal y la red.

- *Entrada/salida programada frente a los accesos directos a memoria:* los mensajes se pueden transferir de dos formas: con entrada/salida programada, donde el procesador copia datos entre la memoria y el interfaz de red, y mediante acceso directos a memoria (DMA, *Direct Memory Access*) , donde el dispositivo de red mismo inicia la transferencia. Existen factores que influyen en el rendimiento de ambos mecanismos.

- *Polling frente a las interrupciones:* Si se usa DMA, otra elección de diseño crítica es el mecanismo para señalar el procesador de la recepción completa de un mensaje

entero. En modo *polling* el procesador lee continuamente un registro de estado del interfaz de red. Otra solución es interrumpir el procesador. Y una solución híbrida podría habilitar el interfaz de red para mandar un interrupción cuando se presente un mensaje en un instante de tiempo específico sin transferir datos.

- *Operaciones colectivas:* El software para computación en *cluster* a menudo necesita técnicas de comunicación colectiva tales como barreras de sincronización o *multicast*, en especial, redes para memoria virtual compartida, donde la actualización de datos se debe distribuir a todos los nodos. Las redes actuales dejan esta tarea al software, y solo unas pocas tienen hardware para operaciones colectivas. El registro de barrera de la interconexión Synfinity es un ejemplo, otras redes con bus compartido como Fast Ethernet pueden difundir los datos fácilmente, mientras que la integración en redes punto a punto como Myrinet o ServerNet es más complicada. Este es un asunto que se debe mejorar en los *cluster* actuales.

Veamos algunas de las redes actuales de alta velocidad (ATM, Fast/Giga Ethernet y Myrinet) que nos han servido como plataforma paralela para nuestros experimentos y que por ello se merecen mención especial y un desarrollo más exhaustivo de sus características principales.

ATM: ***Assinchronous Transfer Mode***

Es el modo de transferencia para implementar la RDSI de Banda Ancha. Es una tecnología de conmutación, de multiplexado, incluso de transmisión, que es una variante de la conmutación de paquetes en cuanto recurre a paquetes cortos de tamaño fijo, llamados células. El tratamiento de esas células en los conmutadores está limitado al análisis de la cabecera, para permitir su encaminamiento hacia las colas apropiadas.

Las funciones de control de flujo o de tratamiento de errores no se efectúan en la red ATM, sino que se dejan a cargo de las aplicaciones de los usuarios o de los equipos de acceso. Estas características permiten a ATM responder razonablemente a las exigencias de tráficos tan diferentes como la voz, las imágenes animadas o los datos. Este modo de transferencia universal hace posible la integración de toda clase de servicios sobre un acceso único a la red [17].

Los conceptos básicos sobre los que se sustenta la tecnología ATM se pueden resumir brevemente en los siguientes [79]:

- *Células:* ATM se basa en el concepto universal de célula (un paquete muy pequeño de tamaño fijo) de 53 bytes de longitud, de los cuales los primeros cinco bytes se utilizan de cabecera para su encaminamiento y control, y los cuarenta y ocho restantes se utilizan para transportar datos. Cada célula es autocontenida y por sí misma se puede encaminar individualmente a través de cada conmutador, desde el emisor hasta el destino. Esta célula puede transportar cualquier tipo de datos. El término asíncrono es utilizado para indicar que no existe ninguna relación temporal entre células ATM. Las recomendaciones de la ITU hasta ahora definidas para ATM proponen que estas células sean transmitidas en caudales de 155 Mbits/s, 622 Mbits/s y 2.4 Gbits/, encajadas en marcos bidimensionales de transmisión síncrona hacia el próximo conmutador.

- *Conexiones ATM:* cualquier usuario que quiera acceder a la red ATM, debe establecer primero una conexión con el conmutador local. Durante el tiempo de establecimiento de la llamada, el usuario *negocia* con la red las características deseadas para la calidad de servicio que requiere su llamada, como el ancho de banda, el destino y la calidad de servicio requerida. Esto es muy importante ya que los diferentes tipos de tráfico requieren diferentes características y es el usuario el encargado de establecerlas. El conmutador local negociará entonces con todos los conmutadores necesarios para conectar al usuario con su destino. En caso de que la conexión sea posible, y que la calidad de servicio y ancho de banda requeridos por el usuario pueden ser soportados por la red, el conmutador local confirma la conexión al usuario y asigna la cabecera de encaminamiento a las células ATM que componen la conexión pedida. Si los requisitos pedidos por el usuario no pueden ser atendidos en su totalidad, es el usuario el que elige aceptar lo que le puede ofrecer la red o cancelar la conexión.

- *Control de la cabecera de la célula*: Se propone, controlar el valor instantáneo y medio del ancho de banda (u otros parámetros) utilizados para cada conexión sobre la red, y se realiza célula a célula. Dependiendo del tipo de servicio negociado, a los usuarios que lo infringen se les proporcionan más servicio si la red lo permite o se le comienzan a descartar células si amenazan la calidad de servicio a otros usuarios. Otra opción para estas células que incumplen lo negociado es asignarles baja prioridad, de manera que, si es necesario, se descarten posteriormente.

- *Traducción de la cabecera de la célula:* La ruta que una célula ATM toma a través de la red viene determinada por el contenido del campo de encaminamiento en la cabecera de la célula. El espacio está muy limitado, así que el campo de encaminamiento se reutiliza en cada conmutador.

La arquitectura ATM consta de tres capas [64], según se observa en la siguiente figura (Figura 2.2). Son las siguientes:

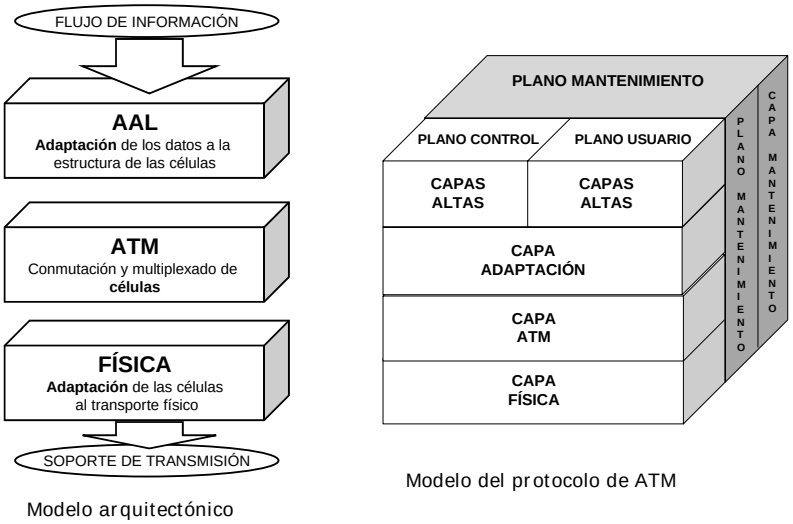


Figura 2.2. Modelo arquitectónico y de protocolos para ATM

- *Capa de adaptación (AAL):* adapta los flujos de información independientemente del tipo que sean, a la estructura de las células. La información se inserta/extrae de la carga útil de la célula.
- *Capa ATM*: realiza la conmutación y multiplexado de las células. Añade a las células que provienen de la capa de adaptación, la cabecera para enviarlas a la dirección correcta.
- *Capa física*: asegura la adaptación al entorno de transmisión. ATM no está sujeto a ningún tipo específico de transporte físico.

Además de lo anterior, la ITU, en su recomendación I.321 describe las funciones de *Mantenimiento y Control* para cada una de las capas que componen el modelo arquitectónico de ATM [79]. Estas funciones de mantenimiento y control, junto al modelo arquitectónico de ATM forman el PRM (*Protocol Reference Model*), que hace referencia a tres planos separados:

- *Plano de usuario*: permite al usuario transferir información relacionada con el control (de flujo y de errores) así como información suya propia
- *Plano de control*: realiza las funciones de control de la llamada y la conexión
- *Plano de mantenimiento*: incluye el plano de mantenimiento, que realiza las funciones de mantenimiento del sistema y coordinación entre todos los planos, y la capa de mantenimiento, que realiza las funciones de mantenimiento acerca de los recursos.

La cabecera de la célula codifica sus cinco bytes en los siguientes campos [79]:

- *Identificador de Camino Virtual e Identificador de Canal Virtual (VPI y VCI):* donde está contenida la información de encaminamiento. Se establecen dos métodos de encaminamiento, uno a través del VPI y otro a través del VCI (ver figura 2.3).

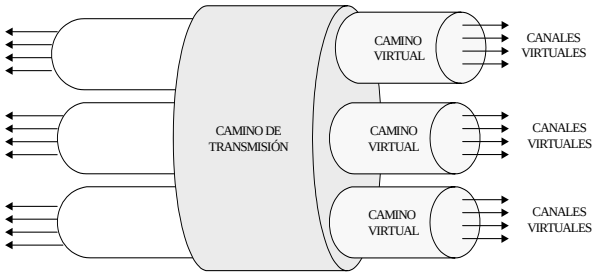


Figura 2.3. Relación entre los campos VPI y VCI

Se distinguen para establecer dos niveles jerárquicos de encaminamiento. Un VPI está formado por un conjunto de VCIs y un conjunto de VPIs formarán un camino de transmisión. El VCI y el VPI tienen sólo significado local, es decir, se utilizan por un conmutador localmente, y los utiliza para determinar la salida hacia la cola apropiada a la vez que se realiza la traducción de la cabecera de la célula. Cuando una célula pasa a través de un conmutador, el conmutador cambia el VCI-VPI, en función de unas tablas de encaminamiento, para que el siguiente conmutador lo utilice, para seguir determinando la ruta de forma progresiva.

- *Identificador del tipo de carga (PTI, Payload Type Identifier):* : 3 bits encargados a la descripción del tipo de carga útil transportada por la célula.

- *Prioridad de pérdida de células (CLP, Cell Loss Priority):* un bit que interviene en los mecanismos de protección contra la congestión. Por lo general, el control de este bit está bajo la responsabilidad de la fuente, que determina, para cada célula, la importancia relativa de la información transportada.

- *Corrección de error de la cabecera (HEC, Header Error Correction):* un byte dedicado a la detección de errores múltiples y a la corrección de un error simple en la cabecera. Es competencia de la capa física.

- *Control de flujo genérico (GFC, Generic Flow Control):* los bits de este campo no están completamente definidos. Soportará las prioridades y las contenciones de acceso entre varios terminales.

Otros temas importantes que hay que describir son:

- *Multiplexación estadística*: es una de las principales diferencias de una red ATM con otro tipo de redes. ATM a parte de ser una tecnología de conmutación, lo es también de multiplexado. Las células se generan por encargo en función del caudal de la fuente. El multiplexado de las células emitidas por fuentes diferentes que comparten el mismo enlace de acceso, se transmite a la capa física. Teniendo en cuenta la baja probabilidad de que la generación de tráfico a la máxima velocidad permitida sea simultánea en todas las conexiones admitidas, se podría aceptar un número de conexiones tal que la suma de las velocidades máximas de las mismas supere el ancho de banda disponible en el enlace. Mediante la multiplexación estadística, se puede conseguir un incremento significativo en el factor de utilización de la red a cambio de incrementar ocasionalmente los retardos y las pérdidas de células, en las situaciones instantáneas en las que el tráfico ofrecido supere la capacidad del enlace.

- *Conmutadores ATM:* la función principal de un conmutador es establecer una conexión entre un puerto de entrada y uno de salida, en función de una información de encaminamiento contenida en la cabecera de la célula. A un conmutador ATM se le piden las siguientes características, más exigentes que las que se piden a otros conmutadores:

- Caudales de acceso elevados que conduzcan a un caudal global de varios Gbits/s
- Varios millones de células conmutadas por segundo
- Una latencia de conmutación poco importante (inferior a 1 ms) y estable
- Un caudal muy bajo de pérdidas de células

La arquitectura no está definida en los estándares, por lo que el desarrollo de ellos persigue la rapidez y la eficiencia, reduciendo la latencia de transmisión y elevando la fiabilidad. El objetivo de diseño de un conmutador ATM es incrementar su capacidad de conmutación soportando elevadas velocidades de acceso. La naturaleza estadística de los flujos ATM y el transporte de varios tipos de tráfico requieren demandas adicionales sobre el conmutador. Además del análisis y modificación de la cabecera de la célula, un conmutador ATM proporciona esencialmente dos funciones:

- Sistema de rutas de las células hacia los puertos de salida apropiados
- Almacenamiento temporal de las células

Por último, tratamos concretamente las **LAN ATM**, una vez planteados los conceptos principales de esta técnica de conmutación, transferencia y multiplexado [80].

Se pueden identificar tres generaciones de redes LAN hasta el momento:

- Primera Generación: redes LAN CSMA/CD y en anillo con paso de testigo (transmisiones de datos moderadas)
- Segunda Generación: como FDDI
- Tercera Generación: donde se enmarcarían las LAN ATM, proporcionando los rendimientos conjuntos de las anteriores y garantizando el transporte de datos en tiempo real, necesario para aplicaciones multimedia.

Entre los posibles tipos de redes LAN ATM destacamos los siguientes:

1. Pasarela a ATM WAN: un conmutador ATM funciona como dispositivo de encaminamiento para conectar una red existente con una WAN ATM.
2. Conmutador ATM central: la interconexión de otras redes LAN se realiza a través de un único conmutador ATM o mediante una red local de conmutadores ATM
3. ATM de grupo de trabajo: las estaciones de trabajo y otros sistemas finales se conectan directamente con un conmutador ATM

Estas son las configuraciones puras, en la práctica, se usan combinados los tipos anteriores para crear una LAN ATM

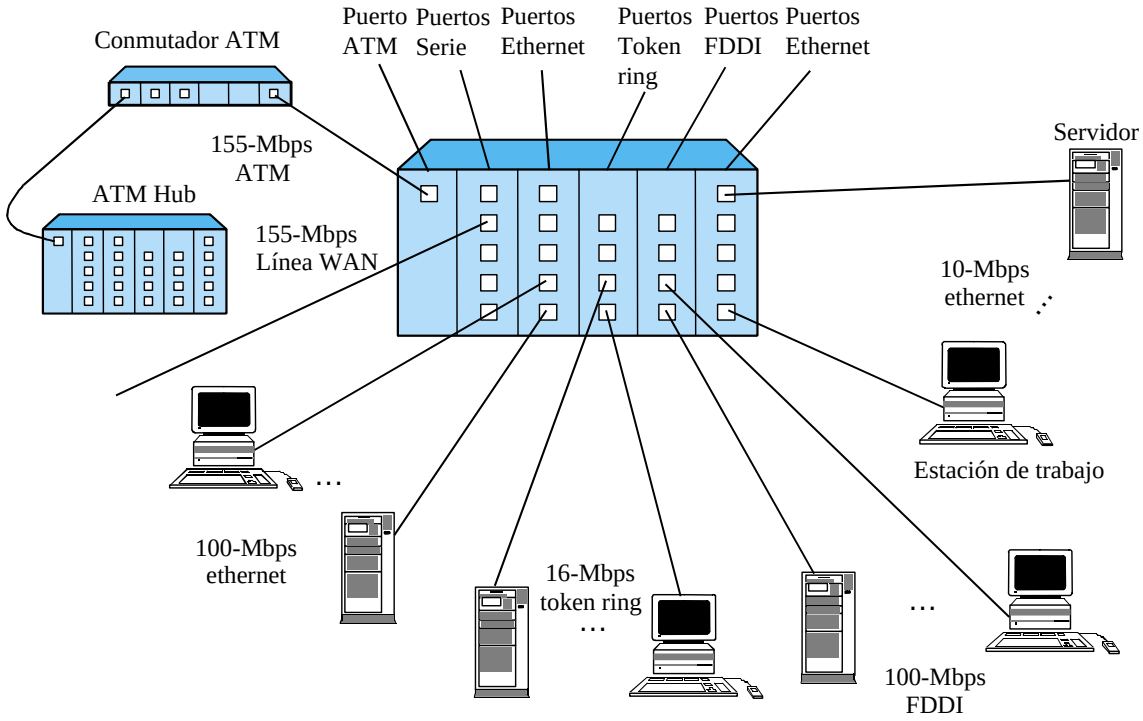


Figura 2.4. Tecnología ATM en un núcleo central

En la figura anterior (Figura 2.4), cada centro ATM incluye puertos que funcionan a distintas velocidades de transmisión y usan distintos protocolos:

- Cada sistema final tiene un enlace dedicado con el centro

- Cada sistema final incluye el hardware y el software necesarios para conectarse a un tipo específico de LAN. Pero en este caso, la LAN sólo contiene dos dispositivos: el sistema final y el centro

Por ejemplo, cada dispositivo conectado a un puerto Ethernet a 10 Mbps hace uso del protocolo CSMA/CD. Sin embargo, dado que cada sistema final tiene su propia línea dedicada, el efecto es que cada sistema final tiene en exclusiva su propia Ethernet a 10 Mbps. La ventaja principal es poder seguir usando las configuraciones de la LAN tradicional, mientras se introduce poco a poco la tecnología ATM, y el inconveniente, el usar un entorno de mezcla de protocolos implica implementar algún método de conversión de protocolos. Un enfoque más simple, pero que requiere que los sistemas finales estén equipados con capacidades ATM, es la implementación de una red ATM pura como la que se expondrá en el siguiente capítulo.

➿ Las generaciones Ethernet

Ethernet es una red de área local, muy extendida, que se ajusta al estándar IEEE 802.3, que utiliza el protocolo de acceso al medio CSMA/CD y posee una velocidad de 10 Mbit/s, aunque con nuevos modelos se alcanzan ya velocidades muy superiores. La versión original de esta técnica en banda base se desarrolló por Xerox para redes de área local Ethernet [80]. Más del 85 por ciento de las conexiones de red al final de 1997 eran Ethernet. Esto representa sobre 118 millones de PCs, estaciones de trabajo y servidores interconectados. El resto de conexiones son una combinación de Token Ring, FDDI, ATM y otros protocolos. Todos los sistemas operativos populares y aplicaciones son compatibles con Ethernet, como lo son los protocolos más superiores, como TCP/IP, IPX, NetBeui y DECnet. [29]

Ethernet ha sido, con diferencia, la LAN más extendida, pero sus bajas prestaciones han obligado a una evolución para adaptarse a las necesidades actuales. La *Ethernet* de 10 Mbps (1982), no es muy adecuada para satisfacer las demandas de ancho de banda actuales en *clusters* o en Internet. En 1994, se desarrollaron dos versiones de *Fast Ethernet* a 100 Mbps (100BaseT y 100VGAnyLAN) y el trabajo más reciente (1997) del grupo de trabajo IEEE 802.3 ha sido *Gigabit Ethernet.* Podemos ver la evolución de la tecnología Ethernet en la siguiente tabla (tabla 2.8).

La distancia máxima con Ethernet llega a los 25 kilómetros dependiendo de la tecnología de cableado utilizada. La distancia de red se reduce al rango de los 2 kilómetros para la tecnología Gigabit Ethernet, lo que nos indica que va orientada a campus o empresas que requieren un gran ancho de banda.

Ethernet asume, en su mayoría, topologías en bus y en estrella, mientras que la Fast Ethernet soporta principalmente la topología en estrella. Para proteger inversioens anteriores, Gigabit Ethernet no requiere cambiar la infraestructura de red, ni la gestión, ni las aplicaciones de las generaciones Ethernet anteriores [37].

Otra opción posible para aumentar el flujo en un entorno Ethernet es la *Ethernet conmutada.* Históricamente se empezó por cortar las redes Etehrnet en sub-redes autónomas conectadas entre sí por puentes, tratando de guardar el tráfico local. Así se multiplica el tráfico por el número de sub-redes. Los puentes no eran aquí más que conmutadores Ethernet que memorizaban las tramas y las envíaban hacia otras redes

Ethernet. La lógica extrema nos llevaría a cortar las redes hasta que cada una sólo tuviera una estación. En Ethernet conmutada la tarjeta de acoplamiento de cada estación está conectada directamente a un conmutador Ethernet que se encarga de encaminar las tramas en la buena dirección según el encaminamiento que se pondrá en marcha en los nodos de conmutación. Cada estación dispone completamente de sus 10Mbps. Sobre una red Ethernet no hay más que dos estaciones. La que se quiere conectar y el conmutador de conexión. Se dispone pues de una Ethernet por terminal (10 Mbps) y de 10 Mbps para el conmutador. Las dificultades provienen de las técnicas de control que hay que poner en marcha para hacerse cargo de los flujos que vienen simultáneamente de todas las tarjetas de acoplamiento de Ethernet. Se encuentran todos los problemas por la arquitectura de red de conmutación de paquetes. La recuperación de una colisión ya no es útil puesto que ya no las hay. Pero, no hay limitación de distancia [69].

Tabla 2.8. Desarrollo de las generaciones Ethernet [7]**.**

Generación	Ethernet 10BaseT	Fast Ethernet 100Baset	Gigabit Ethernet
Año de introducción	1982	1994	1997
Velocidad	10 Mbps	100 Mbps	1 Gbps
Par trenzado UTP	100 m	100 m	25-100 m
Cable coaxial/STP	500 m	100 m	25-100 m
Fibra multimodo	2 Km	412 m en half-duplex 2 Km en full-duplex	500 m
Fibra en modo único	25 Km	20 Km	2 Km
Aplicaciones principales	Compartir ficheros, Impresoras	Trabajo en grupo, Cliente-servidor, Acceso a bases de datos grandes	Procesamiento de imágenes, multimedia, Intranet, Internet.

La interoperatibilidad y la compatibilidad hacia atrás son dos factores importantísimos en Gigabit Ethernet. Conserva el protocolo CSMA/CD. Con cableado en fibra óptica, el precio inicial por puerto es bastante alto, pero los avances contínuos en la tecnología conseguirán reducirlo en un futuro bastante próximo [37]. Sigue utilizando el mismo formato de trama, permitiendo incluso, la misma longitud en el campo de datos. También permite funcionamiento full-duplex usando un concentrador. Se ha mejorado el método para mantener una longitud de 200 metros a velocidades de gigabit. Se ha aumentado el tiempo de portadora del CSMA/CD y el tiempo de ranura, de su valor actual de 64 bytes a uno nuevo de 512 bytes. Se ha añadido la característica de ráfagas de paquetes, que permite a los conmutadores y a otros dispositivos enviar ráfagas de pequeños paquetes para utilizar completamente el ancho de banda disponible. Su funcionamiento es principalmente conmutado [7]

Gigabit Ethernet proporciona, por tanto, un ancho de banda de 1Gbps para redes de campus con la simplicidad de Ethernet a costos más bajos que otras tecnologías de velocidad comparable. Ofrece un camino de mejora natural para las actuales instalaciones Ethernet, manteniendo las estaciones, gestión, herramientas y capacitación necesarias. En la tabla 2.9, vemos, un poco más a fondo, los requisitos que imponen las nuevas aplicaciones [29].

Tabla 2.9. Resumen de aplicaciones que dirigen el crecimiento de las redes

APLICACIÓN	TIPO/TAMAÑO DE DATOS	IMPLICACIÓN DE TRÁFICO EN LA RED	NECESIDADES
Modelado científico, ingeniería	Ficheros de datos 100's de mega-gigabytes	Los ficheros grandes incrementan el ancho de banda requerido	Ancho de banda mayor
Publicaciones, transferencia de datos médicos	Ficheros de datos 100's de mega-gigabytes	Los ficheros grandes incrementan el ancho de banda requerido	Ancho de banda mayor
Internet/Intranet	Ficheros de datos Audio y Video Caudal de transacción alto Ficheros grandes, 1-100MB	Los ficheros grandes incrementan el ancho de banda requerido Bajas latencias de transmisión Alto volumen de flujos de datos	Ancho de banda mayor Menor latencia
Almacén de datos Backup de redes	Ficheros de datos Giga-Terabytes	Los ficheros grandes incrementan el ancho de banda requerido Transmitidos durante periodos de tiempo fijo	Ancho de banda mayor Menor latencia
Vídeo-Conferencia	Flujos de datos constantes 1.5 - 3.5 Mbps	Reserva de servicios Alto volumen de flujos de datos	Ancho de banda mayor Menor latencia Latencia Predecible

Con la proliferación de estas aplicaciones, crece la demanda de mayor ancho de banda, por lo que las organizaciones necesitarán migrar porciones críticas de sus redes a tecnologías de red de un ancho de banda mayor.

Gigabit Ehernet reúne algunos de los factores clave para elegir redes de alta velocidad:

- Migración sencilla a niveles de alto rendimiento sin trastornos
- Bajo costo
- Capacidad para soportar nuevas aplicaciones y tipos de datos
- Flexibilidad en el diseño de la red

Actualmente es posible mezclar datos y vídeo sobre Ethernet mediante una combinación de lo siguiente:

- Ancho de banda incrementado proporcionado por Fast y Giga Ethernet, realzado con las LAN conmutadas
- Nuevos protocolos, como RSVP (*Resource Reservation Protocol*) que proporciona reserva de ancho de banda
- Nuevos estándares como 802.1Q y 802.1p que proporcionarán una LAN virtual (VLAN) e información de prioridad explícita para los paquetes de la red.
- El uso generalizado de compresión de vídeo avanzada tal como MPEG-2

Estas tecnologías y protocolos se combinan para hacer de Gigabit Ethernet una solución muy atractiva para la distribución de vídeo y tráfico multimedia [29]. Más aún, cuando recientemente se ha publicado el nuevo estándar a 10 Gigabits por segundo [30], que no solo va orientado a incrementar el tráfico normal de datos sino también la proliferación de nuevas aplicaciones que consumen mucho más ancho de banda.

El borrador del estándar para la 10 Gigabit Ethernet es bastante distinto de los estándares Ethernet anteriores, en primer lugar es que sólo trabaja con fibra óptica y sólo opera en modo full-duplex, lo que indica que los protocolos de detección de colisión ya no son necesarios. Debido a estas características, Ethernet puede ahora llegar a los 10 Megabits por segundo, además de seguir manteniendo las características de Ethernet como el formato de paquete. También, puede interoperar con otras tecnologías de red como SONET.

La expansión de Ehernet para usarla en redes de área metropolitana puede ser ahora extendida de nuevo a redes de área amplia. Se espera que 10 Gigabit Ethernet sea el nuevo estándar que ayudará a crear una convergencia entre las redes diseñadas en principio solo para tráfico de voz y las nuevas redes de datos. El estándar se empezará a adoptar a mediados del 2002.

Por tanto, El propósito de la 10 Gigabit Ethernet es extender los protocolos 802.3 a una velocidad de 10 Gigabits por segundo y expandir el espacio de aplicaciones Ethernet para que incluyan enlaces WAN. Estos son los criterios que el nuevo estándar propuesto debería reunir [31]:

- Debe tener un amplio potencial de mercado, soportar un gran conjunto de aplicaciones, con múltiples vendedores soportándola y múltiples clases de clientes.
- Debe ser compatibles con otros protocolos 802.3 existentes, así como con la arquitectura OSI y con las especificaciones de gestión de SNMP (Simple Network Management Protocol)
- Debe ser bastante diferente de los otros estándares 802.3, haciéndolo la única solución al problema en lugar de una solución alternativa.
- Debe estar técnicamente demostrada antes de la ratificación final
- Debe ser economicamente posible para cliente que desarrollen, debe tener un costo razonable, incluyebdo toda la instalación y gestión, para que crezca el esperado rendimiento

➲ Myrinet

Es una red conmutada, del orden de los Gigabits por segundo, de Myricom, Inc. El objetivo de Myricon es hacer de la interconexión de sistemas un producto para la construcción de *cluster*s de computadoras. Myrinet está basada en la tecnología VLSI y en los multicomputadoras desarrollados en el Instituto de Tecnología de California y la tecnología ATOMIC/LAN desarrollada en la Universidad del Sur de California. Está diseñada para construir tanto *clusters* basado en SAN como en LAN. Myrinet puede asumir cualquier topología, no se restringe a una malla de conmutadores o a cualquier topología regular.

Una SAN (*System Area Network)* es una forma de combinar elementos hardware estándares para lograr sistemas verdaderamente escalables que proporcionan

prestaciones y productividad superiores, de manera fiable. Al igual que los sistemas tradicionales basados en bus, una SAN hace posible que un grupo de recursos computacionales sea compartido para poder ejecutar aplicaciones con altas prestaciones. Por el contrario, una SAN agrupa los componentes más importantes del sistema como elementos independientes. Como resultado, cualquier elemento puede relacionarse con cualquier otro sin la intervención del procesador. Esta conectividad *cualquiera-a-cualquiera* es crítica para utilizar aplicaciones intensivas en datos, como multimedia, Internet e Intranet.

Myrinet se define a nivel de enlace de datos por su formato de paquete de longitud variable, control de flujo y errores en cada enlace, el uso de conmutadores de barras cruzadas *cut-through*, de baja latencia, para encaminar los paquetes, e interfaces programables por el cliente. Las SAN Myrinet tienen un costo más bajo que las LAN Myrinet por su reducido tamaño físico [37]. Los enlaces para SAN operan a la misma velocidad que los de la LAN, soportando distancias de hasta tres metros. El formato de los paquetes y el software para una SAN Myrinet son idénticos a los de una LAN Myrinet. De hecho, dentro de cada LAN Myrinet hay una SAN Myrinet, puesto que los chips que implementan la comunicación, conmutación e interfaces para la LAN tienen puertos SAN. En los componentes para LAN, estos puertos se traducen a puertos LAN mediante otro chip VLSI.

Myrinet es una tecnología de conmutación de alto rendimiento y costo razonable que se usa ampliamente en la inetrconexión de *cluster* de estaciones de trabajo y PCs. Redes convencionales como Ethernet también se pueden usar para construir *cluster*s, pero no proporcionan el rendimiento o las características requeridas para conseguir el alto rendimiento y la alta disponibilidad del *clustering*. Las principales características que lo distinguen del resto de redes son las siguientes:

- Enlaces, puertos del conmutador y puertos de interface full-duplex de 2.28+2.28 Gbps.
- Control de flujo y control de errores en cada enlace
- Conmutadores *cut through*, de barras cruzadas, de baja latencia para aplicaciones altamente escalables
- Se permite cualquier topología. Las redes Myrinet pueden escalar a decenas de cientos de hosts y puede también proporcionar caminos de comunicación alternativos entre hosts
- Las interfaces en los hosts. Ejecutan un programa de control para la interacción directa con los procesos del host para comunicaciones de baja latencia, enviando, recibiendo y almacenando paquetes, y para mapear y monitorizar la red.

En Myrinet, por ser conmutada, los mensajes emitidos por una estación llegan a su destino a través de enlaces punto a punto bidireccionales y conmutadores, sin pasar por otras estaciones. Cada estación debe tener, por tanto, una dirección propia. Los componentes básicos de Myrinet son:

- Interfaces Myrinet, tales como interfaces Myrinet/PCI
- Conmutadores Myrinet, de hasta 16 puertos

- Cables de enlace y adaptadores de fibra. Myrinet como Ethernet tiene múltiples implementaciones de la capa física. Para Myrinet, esos enlaces físicos son SAN, LAN, y fibra.
- Soporte software: para los hosts y sistemas operativos más comunes. El software está disponible (*open source*).

Se pueden instalar las interfaces y el software en los hosts, y conectar la red con cables y conmutadores. El software mapea la red periodicamente, y usa los caminos de comunicación que están disponibles de host a host. No es necesario configurar tablas de encaminamiento ni programar los conmutadores [65].

- *Interfaces:* Las interfaces de host conectan los terminales a la red a través de un único puerto, y son los únicos puntos de inyección de mensajes en la red Myrinet. También deben consumir los mensajes dirigidos a su puerto. Un host puede incluir varias tarjetas de interfaz, pero desde el punto de vista de la red, dichos interfaces operan independientemente. Cada tarjeta posee una dirección única (UID) de 48 bits escrita en una memoria EPROM.

Las interfaces de host incluyen, en la placa, un chip VLSI llamado LANai (del que existen varias versiones), que consta de un procesador, un dispositivo DMA y el controlador de las conexiones red-host. Además, la placa incluye una memoria RAM, usada para almacenar paquetes en tránsito entre el host y la red y para almacenar el MCP (*Myrinet Control Program*).

El MCP se carga desde el host (por el controlador de dispositivo, un comando de usuario o una aplicación) a la RAM de la tarjeta de interfaz; hasta ese momento, el interfaz de host se encuentra en condición de *reset*, aceptando mensajes pero descartándolos. Una vez cargado el MCP, el procesador interno del chip Lanai comenzará a ejecutarlo. Este programa de control es fundamental para el funcionamiento de Myrinet pues controla la transferencia de paquetes entre el host y la red. Además, puede realizar otras funciones esenciales para la reconfiguración de la red.

Myrinet es muy similar a ServerNet [26]. Difieren considerablemente en el diseño de la interfaz del host. Una interfaz de host Myrinet consiste de dos componentes principales: el chip LANai y su memoria SRAM asociada. El LANai es un chip VLSI a medida que controla la transferencia de datos entre el host y la red. Su principal componente es un microcontrolador programable. Así, los datos deben escribirse primero en la SRAM de la tarjeta de red antes de que se inyecten en la red. La SRAM también almacena el MCP y algunas colas de trabajos.

Versiones anteriores de la interfaz del host estaban basados en el Sbus de SAN, pero las últimas usan la interfaz PCI de 32/64 bits. EL LANai corre a frecuencias comprendidas entre 33 y 66 MHz. Junto con el control de la transferencia de datos, el LANai también es responsable del mapeo de red automático y de monitorizar el estado de la red. El tamaño de la SRAM en placa va de los 512 Kbytes a 1 Mbyte. El LANai se comunica con los divers de los dispositivos del hosts o con las librerías a nivel de usuario mediante colas de trabajo que residen en la SRAM [7, 21].

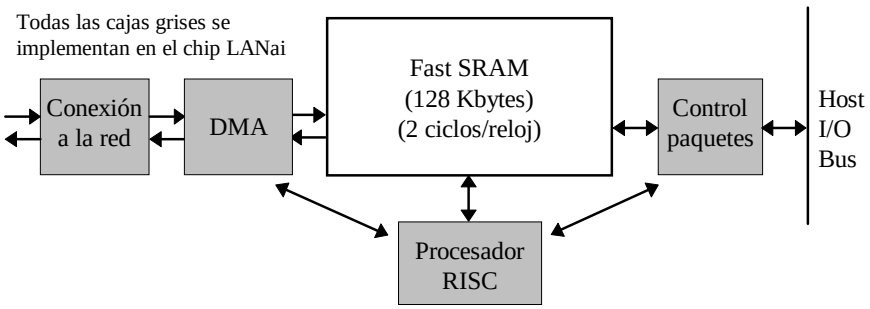


Figura 2.5. Chip LANai

• *Enlaces:* Los enlaces full-duplex Myrinet están formados por un par de canales unidireccionales. Estos canales transportan 9 bits en paralelo, para LAN, y 10 bits en paralelo para SAN. Llamaremos puerto a la conexión de un enlace a un sistema (conmutador o host). Es interesante destacar los siguientes puntos:

- El control de flujo se realiza en cada enlace. El receptor de un canal puede bloquear y desbloquear (con un protocolo *STOP/GO*) el flujo de información procedente del emisor de dicho canal. Esto sucederá cuando el *buffer* de entrada del receptor corra riesgo de desbordarse. La gestión del control de flujo se realiza haciendo que el receptor inyecte los símbolos de control *STOP* y *GO* en el canal, en la dirección opuesta al enlace. Sólo afecta a los datos, todos los símbolos de control están exentos de él y tienen mayor prioridad que los datos. El receptor utiliza un buffer organizado en forma de cola que funciona de manera conceptual, como se observa en la siguiente figura (figura 2.6).

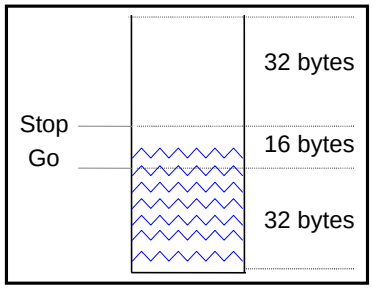


Figura 2.6. Operación del buffer

Si el flujo que sale se bloquea provocando que el buffer se llene hasta alcanzar la línea de Stop, el receptor genera un símbolo de control *STOP*, que detiene el flujo antes de que el buffer se desborde. Al reanudarse el flujo de salida, el receptor genera un símbolo de control *GO* cuando el nivel de llenado del buffer alcanza la línea Go. La parte superior del buffer evita el exceso de flujo, la inferior evita la inanición de datos. Las posiciones del buffer entre Stop y Go aseguran que los símbolos *STOP* y *GO* no consuman excesivo ancho de banda en el canal que va en sentido contrario.

- Los circuitos del puerto detectan si éste está desconectado, el enlace está desconectado o el enlace está conectado a un componente apagado. Si un puerto

se encuentra en cualquiera de estas condiciones, un paquete que intente salir por dicho puerto debe ser descartado en lugar de bloqueado.

- Un paquete no puede estar bloqueado indefinidamente debido al control de flujo (posible situación de *Deadlock*). Tampoco puede ocupar indefinidamente el canal por donde se transmite; esto puede ocurrir por un fallo en la generación del símbolo de final de paquete. Para evitar estas situaciones, existe un *timeout* (*Long-Period Timeout*), que se aplica tanto en emisión como en recepción. Un paquete es descartado (por emisor y receptor) si consume tiempo de transmisión más allá de este *timeout*.

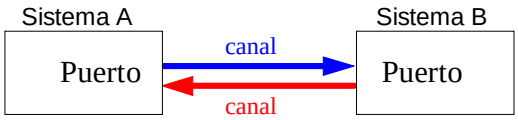


Figura 2.7. Enlaces Myrinet

- *Conmutadores:* Los conmutadores Myrinet son sumamente sencillos, pues sólo se requiere de ellos que encaminen un paquete desde un puerto de entrada hasta un puerto de salida, que viene indicado en la cabecera del paquete. No es necesario, pues, que los conmutadores tomen decisiones de encaminamiento. Los conmutadores ni siquiera tienen UID propio.

La implementación física de los conmutadores se basa en chips de conmutación *crossbar* y chips de interfaz para Myrinet. La técnica de conmutación empleada es *Wormhole* (con 108 bytes en el *buffer* de cada puerto del conmutador). Esta conocida técnica, unida a la sencillez del encaminamiento en los conmutadores, logra que la latencia media en Myrinet sea baja (en torno a 20 microsegundos) [21].

- *Topología:* Myrinet permite cualquier combinación de conmutadores e interfaces de host, incluso si la red resultante contiene ciclos. También se permite que los conmutadores tengan puertos desconectados (al aire) o puertos conectados a terminales apagados [21].

- *Interfaces software y Rendimiento:* hemos comentado ya que los paquetes pueden ser de cualquier longitud, y de este modo pueden encapsular otros tipos de paquetes, incluyendo paquetes IP, sin una capa de adaptación. Cada paquete se identifica por un tipo, por lo que una Myrinet, como una Ethernet, puede transportar paquetes de muchos tipos o protocolos, concurrentemente. Así, Myrinet soporta varias interfaces software.

Bancos de pruebas del programa de control y de la interfaz de programación de aplicciones (API) de Myricom muestran, por un lado, caudales de datos alrededor de 1.1 Bbps entre procesos de usuario Unix en diferentes hots, y latencias de mensajes cortos mas bajas de 13 microsegundos. Bancos de pruebas usando interfaces software TCP/IP o UDP/IP muestran, por un lado, caudales de datos en el rango de los 250 a los 1147 Mbps. Estos caudales están determinados en su mayoría por la pila de protocolos IP del sistema en lugar de por las interfaces de red o los enlaces, y son, por tanto, altamente variables en función de los hosts y los sistemas operativos. Este rendimiento TCP/IP es

a menudo más que suficiente para aplicaciones distribuidas construidas en esas interfaces software estándares. Implementaciones eficientes de las interfaces software, como MPI, sobre Myrinet están disponibles por Myricom [65].

En la siguiente tabla (tabla 2.10) se resumen algunos valores de rendimiento (para tamaño de mensaje desconocido)

Tabla 2.10. Rendimiento de la capa de mensajes sobre Myrinet

Maquina	API	Latencia (μs)	Ancho de banda (Mbps)
200 MHz Ppro	BIP	4.8	1009
166 MHZ Pentium	PM	7.2	941
Ultra-1	AM	10	280
200 MHz Ppro	TCP (Linux/BIP)		293
200 MHZ Ppro	UDP (Linux/BIP)		324
DEC Alpha 500/266	TCP (Digital Unix)		271
DEC Alpha 500/266	UDP (Digital Unix)		404

BIP es un protocolo muy simple, que alcanza buenos valores de rendimiento, pero, se ha desarrollado un nuevo protocolo por Myricom llamado GM que sustituye a los antiguos y a los programas MCP [7, 47] que será el que utilicemos en nuestros experimentos y que, por tanto, expondremos en el capítulo 2 más detenidamente.

- *Enlaces y componentes*: Myrinet está especificada a nivel de enlace de datos y en los niveles Físicos de SAN y LAN como un estándar ANSI. Las especificaciones de encaminamiento y enlace son públicas, están publicadas y abiertas. A nivel de enlace de datos, todos los enlaces Myrinet son iguales. Para montar un *cluster* Myrinet, sin embargo, es necesario hacer algunas elecciones de la forma física de los enlaces y de los componentes Myrinet que se van a conectar. Los puertos “nativos” de Myricom son puertos SAN. Los cables SAN y LAN operan al mismo caudal en algunos cables y en otros los LAN solo a la mitad. Se pueden acomodar automáticamente caudales de datos mezclados, gracias al control de flujo en cada enlace. Muchas interfaces y componentes del conmutador se construyen en ambas formas, para SAN y para LAN y los conmutadores Myrient también pueden proporcionar una mezcla de puertos SAN y LAN. La actual generación de Myrinet en fibra, emplea un convertidor separado de SAN o LAN a fibra [65].

- *Tecnología y Fiabilidad:* Los componentes Myrinet se implementan con la misma tecnología avanzada de las estaciones de trabajo o PCs actuales (chips CMOS VLSI). Este uso de tecnología CMOS es una razón por la que el rendimiento Myrinet continuará avanzando, en las etapas en las que avancen los hosts, sin cambios en la arquitectura de la red ni en el software. Estos componentes son también muy fiables. Myrinet exhibe un caudal de errores muy bajo y es altamente robusta con respecto a fallos de hosts, conmutadores y cables. Continuamente se mapea a sí misma y usa rutas alternativas, si están disponibles, para sortear fallos [65].

El que todas las especificaciones de Myrinet sean públicas ha motivado a muchos grupos de investigación a implementar sus propias capas de mensajes y es una de las principales razones de la popularidad de Myrinet. Existen drivers disponibles para Linux, Solaris, Windows NT, DEC, UNIx, Irix y VxWorks en Pentium, Sparc, Alpha, MIPS y procesdaores Power PC. Además hay disponible un parche del compilador GNU de C para desarrollar programas MCP [7].

➲ La nueva generación de Myrinet

Myrinet 2000, también llamada 3º generación, se diferencia de las anteriores solamente en el nivel físico y de enlace de datos. Esta nueva generación no intenta cambiar la arquitectura solo mejorar la tecnología. La motivación principal han sido los avances de la tecnología, se ha intentado ir paso a paso y no innovar en todas las dimensiones (políticas de encaminamiento, arquitectura, reconfiguración, mejoras tecnológicas). Siendo totalmente compatible con la arquitectura, protocolos y programación de los productos actuales de Myrinet (2ª generación).

Con Myrinet 2000 nacen una nueva serie de Interfaces de red Myrinet PCI (PCI64B), que tiene un procesador más rápido que las anteriores (PCI64A ratio de 2.28+2.28 Gb/sg), con un ratio de 2+2 Gb/sg. Con esto disminuye la latencia punto a punto, aproximadamente a 9µs.

El sucesor del chip que tenía la 2ª generación de Myrinet (LANai 7 tecnología CMOS de 0,35µm) es LANai9 basado en CMOS de 0,25 µm, que ofrece:

1. Incremento de la velocidad del procesador RISC de 66 Mhz a 132 Mhz, en los productos de la clase B y en los de la clase C a 100 Mhz. Esto mejora la ejcución de código compilado x1.7, y reduce la latencia punto a punto de GM de 13.5 µs to 9 µs aproximadamente.
2. Incrementa el ratio de 2.28 Gb/sg a 2 Gb/sg Full-Duplex en la Myrinet-2000. Este chip LANai 9 puede funcionar a 2.28 Gb/sg para ser totalmente compatible con la versión anterior.
3. Pueden operar en 32 bit-33 Mhz, 64 bits-33 Mhz, 32 bit-66 Mhz, 64 bits-66 Mhz para ser compatibles con las anteriores y a 3.3 o 5 voltios.

Myrinet-2000 introduce tres nuevos enlaces físicos, que pueden operar a 2 Gb/sg Full-duplex:

- *Myrinet SAN-2000* (M3M) de 3 metros compatible con SAN-1280 de la 2ª generación,
- *Myrinet-2000 Serial* (M3S) convergen con el estándar de la industria HSSDC, High Speed Serial Data Conector (Conector de para datos serie de alta velocidad). Esta cable es blindado, y es más pequeño que los anteriores DB-37 y permite que la construcción de conmutadores en cascada sea más densa. Con cables de 10 metros.
- *Myrinet-2000 Fiber* (M3F) utilizan 50/125 fibra multimodo sobre los 200 metros de longitud.

Los switches Myrinet-2000 funcionalmente son similares a los de la segunda generación. Poseen una serie de líneas de control y monitorización SNMP/Ethernet del

switch. Por último, comentar que la evolución de Myrinet se centra en los siguientes puntos:

1. Incrementar el ratio de instrucciones RISC. Convergencia con otras tecnologías de red como InfiniBand, Fiber Channel, GigabitEthernet.
2. Construcción de "Superswitches", para grandes cluster ya que actualmente los avances tecnológicos lo permiten [48, 67]

2.7.4. Evolución de los distintos componentes

En el apartado anterior se ha expuesto la evolución de la tecnología de red y ha quedado claro que los requisitos de ancho de banda que exigen las aplicaciones multimedia van a poder cumplirse sin problemas con las nuevas tecnologías ya existentes en eel mercado. También se ha mencionado que la evolución de las CPU nos hace disponer cada vez de forma más fácil y más barata de máquinas con una elevada potencia de cálculo. Por lo tanto, vemos que tanto las máquinas como la tecnología de interconexión, redes y buses, siguen el camino de la evolución a pasos agigantados.

Tan sólo hemos dejado de mencionar un aspecto importantísimo en el que el ritmo de evolución no se acerca ni de lejos a los de las máquinas y las tecnologías de interconexión, y es el de la entrada/salida, y en particular la industria de los discos. El acceso a la infomación almacenada en disco (cada vez de mayor capacidad, eso sí) es una parte esencial de la entrada/salida. Los servidores tendrán que acceder a la información que tienen almacenada en ellos para responder a las peticiones de sus clientes, y aunque los clientes se comuniquen mediante una red de alta velocidad, después, el servidor tiene que buscar la información que tendrá almacenada de forma local, o en otros servidores, si lo que se tienen son cluster de servidores.

La ley de Amdahl implica que la ganancia obtenida está limitada por el componente más lento del sistema. Por tanto, el rendimiento del sistema de discos es un factor dominante en las posibles causas del cuello de botella existente en el comportamiento del sistema. Es necesario mejorar la entrada/salida. La capacidad de almacenamiento y la velocidad de acceso con cuestiones críticas que hay que tener en cuenta en el diseño de los sistemas. Los arrays de discos y la entrada/salida paralela son temas actuales de intensa investigación. Aunque el uso de dispositivos de entrada/salida incremente el rendimiento del sistema, no reduce la probabilidad de fallos de disco. De ahí que se hayan propuesto mecanismos para mejorar aspectos relacionados con la fiabilidad, capacidad y rendimiento, y que la investigación en temas relacionados con el caching, prefetching, sistemas de ficheros con y sin servidor, patrones de acceso e interfaces de entrada/salida sea de total actualidad. Los sistemas de ficheros están modificándose para aprovechar las ventajas de la posibilidad de usar algunos dispositivos en paralelo. Además estos mecanismos se han mejorado para usar recursos situados en nodos de un cluster o en redes de almacenamiento tales como las SAN (*Storage Area Network*) y las NAS (*Network-Attached Storage*) las cuales proporcionan interfaces de entrada/salida eficientes y una rica semántica [42]. Todos estos temas los detallaremos en el capítulo 4 cuando expongamos los temas relacionados con nuestra investigación.

Este será el argumento principal que utilizaremos para resaltar la importancia de nuestra técnica implementada, ya que irá directamente a acelerar la parte del proceso que hemos detectado como el principal cuello de botella y el principal problema para la escalabilidad del cluster. Volveremos más adelante a tratar este tema, pero es ahora el momento de reflexionar sobre lo que se va a plantear más adelante y en la idea de que no todo en los cluster evoluciona a la misma velocidad. En una aplicación paralela multimedia, por lo general, la parte de procesamiento va a ser mayor que la parte de E/S, pero la parte de procesamiento se reducirá utilizando máquinas más rápidas, o los cada vez más potentes biprocesadores. Pero par mejorar la parte de E/S no solo bastará con utilizar una red más rápida, sino que habrá que optimizar el acceso a la informaicón final, y ese va a ser nuestro principal objetivo.

En [7] se pueden encontrar intersantes gráficos sobre la evolución de las CPU's y de las máquinas que se utilizan en los clusters

También mencionar que el software evoluciona a la par del hardware, y que el nacimiento de productos para cluster (entornos de programación, herramientas de desarrollo, etc.) cada vez más perfeccionados es una realidad.

2.8. APLICACIONES

2.8.1. Introducción

Existe un crecimiento importantísimo en la evolución de procesadores de alto rendimiento, redes de alto ancho de banda y baja latencia, e infraestructura de desarrollo para facilitar el uso de los clusters. Los clusters de alto rendimiento son ya una de las plataformas más competitivas. Hay un gran número de problemas que pueden ser resueltos por estos sistemas de forma altamente eficiente. Para la actual generación de cluster, las áreas más significativas para aplicaciones científicas son el rendimiento de sistemas de entrada/salida y las herramientas de compilación, depuración y monitorización de rendimiento para aplicaciones paralelas [15].

Vamos a centrarnos en un tipo especial, aunque muy amplio y de interés creciente actualmente, de aplicaciones. Vamos a centrarnos en las aplicaciones multimedia, y vamos a exponer los principales problemas con este tipo de aplicaciones en los cluster de computadoras.

2.8.2. Aplicaciones Multimedia

A todos nos suena bien la palabra multimedia, siempre que la oímos pensamos en animaciones, películas o juegos de ordenador. Pero no sólo existen aplicaciones de entretenimiento sino también de otras cosas más como educación, marketing, servicios de información, teleconferencia, publicidad, televisión interactiva, etc. Sin duda es una industria que crece y crece sin parar.

Nos suena algo muy moderno, divertido, futurista, y relacionado con Internet seguramente. Vemos a nuestro alrededor distintos cursos, masters, diplomas, y certificados de expertos en multimedia, parece que es algo que puede ir muy bien. Es una de las palabras clave que intentamos que aparezca en nuestros artículos, para

asegurarnos, quizás, que sea aceptado en algún congreso importante. Parece que cualquier investigación que se realice es mejor, si al final, también se ve alguna relación con “algo” multimedia.

Además, en nuestro entorno universitario los campos de estudio relacionados con las aplicaciones multimedia son muy numerosos, y un gran número de investigadores en todas las universidades del mundo están trabajando en temas como la gestión del tráfico multimedia, reserva de recursos, control de la congestión, diseño de la calidad de servicio, seguridad, www e hipermedia, resistencia a errores, compresión, comunicaciones multimedia inalámbricas, routers multimedia, multicast, y muchos, muchos más.

Pero es que esa es la realidad, las aplicaciones multimedia se han apoderado de la tecnología y ahora ellas marcan el ritmo y no hay quien las pare, ni a ellas ni a sus creadores, diseñadores e investigadores.

En este trabajo se ha tomado una aplicación multimedia real y estándar para que genere el tráfico en la red y para que exija unas demandas de servicio en una arquitectura cliente-servidor típica.

No vamos a generalizar a todo el amplio espectro de las comunicaciones multimedia, con todo lo que esto conlleva. Bien es verdad que la mayoría de las aplicaciones multimedia están destinadas a viajar por Internet, y de ahí el enorme trabajo que se está realizando tanto del lado de las aplicaciones como del propio lado de Internet para convertirla, o mejor, extenderla de una simple red de datos a una gran red multimedia.

Nos vamos a centrar en las aplicaciones multimedia sobre clusters de estaciones de trabajo, lo que implica de forma natural el procesamiento paralelo para poder manejar, almacenar y transmitir la información que este tipo de aplicaciones genera. Son muchas las aplicaciones multimedia que se van a instalar en los clusters con distintos fines como ahora veremos, y en éstas centraremos nuestra investigación y estudio.

Los clusters se están usando ya en infinidad de aplicacionesde Internet, entre ellas el comercio electrónico. Ya que aportan escalabilidad, disponibilidad, rendimiento, almacenamiento masivo y soporte a bases de datos. Intentan también ser soporte para detección de cyber ataques y control [8]

Tener una gran potencia de cálculo con comunicaciones de alta velocidad y a bajo coste, son las características más interesantes que nos pueden ofrecer. Esto nos puede servir para grandes aplicaciones científicas, pero también nos es útil en el día a día de las empresas. El disponer de mayor potencia de cálculo es interesante también para servidores web o para grandes sistemas gestores de bases de datos. Aunque en los primeros la velocidad del canal sea crítica y en los segundos la velocidad de acceso a disco sea crítica, un servidor web necesita mucha potencia de cálculo, y un sistema gestor de bases de datos que realice muchas consultas complejas y concurrentes, también. Pero la potencia de cálculo no es la única razón de interés, también lo son la escalabilidad y la tolerancia a fallos, por ejemplo [74].

Veamos algunas de las interesantes aplicaciones multimedia que están invadiendo nuestra vida cotidiana, quizás sin darnos cuenta.

Entre ellas se encuentran MHEG-5 para televisión interactiva, las librerías digitales, los sistemas de vídeo bajo demanda cada vez más sofisticados, y en general toda una serie de trabajos para mejorar la distribución de vídeo.

- MHEG

El grupo de trabajo ISO/IEC/JTC1SC 29 (WG12), más conocido por MHEG, proporcionan estándares para la representación de objetos con información multimedia e hipermedia que se intercambian entre aplicaciones y servicios usando una variedad de medios. Los objetos definen la estructura de la presentación hipermedia multimedia.

El trabajo actual del grupo se basa en el estándar MHEG-5, ISO/IEC 13522-5, que es el estándar base para la difusión de la televisión digital interactiva. Este estándar se completó en 1998, y se implementó en la televisión del Reino Unido, y actualmente está siendo evaluado por otros países. El trabajo en MHEG tiene que ver con los tests de interoperatibilidad y con proporcionar un formato de codificación alternativo en XML para MHEG-5. Esto último se diseñó para promocionar el uso de MHEG por la comunidad WWW y en Internet [51]

Algunas de las áreas en las que se están estudiando extensiones de MHEG-5 son:

- Soporte para objetos 3D
- Paso de parámetros y otros objetos entre MHEG-5 y otras entidades externas, como tecnologías web, bases de datos, etc.
- Clases de conexión adicionales
- Temas de convergencia para tecnologías WWW y MHEG-5

Los documentos fuente se corresponden con los ISO/IEC 13522: Tecnología de la Información – Codificación de la Información Hipermedia Multiemdia:

- Parte 1: Representación de objetos – Notación Base (ASN.1) (Recomendación ITU-T T.171)
- Parte 3: Representación de Intercambio de *Script*
- Parte 4: Procedimiento de registro para el identificador de formato MHEG
- Parte 5: Soporte para Aplicaciones Interactivas de nivel base
- Parte 6: Soporte para Aplicaciones Interactivas mejoradas
- Parte 7: Tests de Interoperatibilidad y ajuste para ISO/IEC 13533-5
- Parte 8: Notación XML para el ISO/IEC 13522 (MHEG-XML)

La parte 3 especifica un conjunto de extensiones para intercambio de objetos, la parte 5 especifica el subconjunto MHEG para implementaciones de nivel base tales como aquellas que se utilizan en el Vídeo bajo Demanda y los servicios de Compra en casa. La parte 6 cubre el uso de Java como una máquina virtual MHEG y proporciona un API para MHEG-5. La parte 8 proporciona una representación XML para MHEG-5.

El estándar MHEG proporciona un conjunto estandarizado de clases de objetos que se pueden usar para controlar la presentación de informaicón hipermedia multimedia. Define las siguientes clases de objetos:

- Una clase *content* para objetos que contienen datos presentables
- Una clase *multiplexed content* para objetos que contienen conjuntos de datos que han sido multiplexados en un flujo de distribución único
- Una clase *composite* para objetos que contienen más de un tipo de datos u objetos inmersos
- Una clase *action* para objetos que controlan acciones relacionadas a la preparación de datos, creación de objetos en tiempo real, presentación, interpretación, interacción o activación de objetos de datos
- Una clase *link* para objetos que definen los enlaces entre objetos de datos
- Una clase *script* para objetos que contienen scripts que definen complejas relaciones entre objetos de datos
- Una clase *descriptor* para objetos que contienen descripciones de objetos intercambiados
- Una clase *container* que poropociona un contenedor para agrupar objetos que son intercambiados como un conjunto

Los productos y servicios basados en MHEG están empezando a estar disponibles. El proyecto de investigación OMHEGA ESPRIT está preparando un conjunto de herramientas MHEG que van a ser probadas como parte del proyecto de Entorno de Aprendizaje Abierto Colaborativo Europeo DELTA. El consorcio DAVIC [18]usó MHEG-5 como la base para el intercambio entre proporcionadores de servicios de televisión digital (difusión, satélite y cable) y las unidades instaladas en las casas [19].

Distintas empresas en todo el mundo están desarrollando software para MHEG-5 para implantar la televisión interactiva [9]

Figura 2.8. Ejemplo de televisión interactiva

Existe un campo activo de investigación alrededor de este estándar, un ejemplo es otra interesante herramienta para aplicaciones MHEG-5, *Media Touch*, en desarrollo

en el centro de estudios y laboratorio de Telecomunicaciones de Turín. Se centran en la creación de aplicaciones MHEG-5 y proporcionan la presentación de *Media Touch,* la herramienta que han desarrollado para crear estas aplicaciones.

MHEG-5 hace posible formar aplicaciones orientadas a páginas, lo que indica que una aplicación se compone de un conjunto de escenas con objetos comunes a todas las escenas. Solo hay una escena activa en un instante de tiempo y la navegación con una aplicación se realiza en forma de transición entre escenas. La navegación entre aplicaciones también es posible.

Lo que más nos interesa a nosotros es que MHEG-5 va orientada a aplicaciones distribuidas en una arquitectura cliente/servidor. Normalmente estas aplicaciones residen en el servidor y solo las porciones requeridas en un punto dado de la presentación se bajan al cliente.

Es responsabilidad del cliente tener una herramienta en tiempo real que interperte las partes de la aplicación, realice la presentación al usuario, y gestione la interacción local con el usuario. Mientras que la aplicación es independiente de la máquina, el software del cliente sí depende de la plataforma actual, y se debería optimizar para el hardware específico donde corre.

Sin embargo, su uso no está limitado a servicios de almacenamiento y recuperación. En los entornos de difusión, por ejemplo, el conjunto de canales transmitidos en una red de difusión pueden ser considerados como un servidor virtual donde los mecanismos para bajarse la información se basan en la redifusión de todas las porciones de una aplicación

Han probado que *Media Touch* es una herramienta adecuada para generar aplicaciones tales como películas bajo demanda, noticias bajo demanda, información turística y televisión interactiva. Estas aplicaciones han ayudado en la demostración de ARMIDA (Applications Retrieving Multimedia Information Distributed over ATM) [44] y *Media Touch* en un gran número de eventos en Telecom Italia [50].

Por lo que podríamos tener otro ejemplo de aplicación y uso de información compartida mediante NFS

- Video-bajo-Demanda

La interactividad en las distribuciones de vídeo dio lugar a nuevas ideas para servicios que empezaban a surgir. En los servicios de Vídeo bajo demanda el usuario puede elegir programas cuando quiera. Esta nueva clase de servicios fue posible por la tecnología ATM, las técnicas de compresión eficientes y otros desarrollos en telecomunicación. Muchas compañías empezaron a ofrecer productos de Vídeo bajo de manda y aún esta aplicación es objeto de interesantes investigaciones y mejoras en compañías y universidades de todo el mundo que no dejan de crear productos y estándares. También los operadores de teléfono y televisión por cable están invirtiendo en sus redes para proporcionar este tipo de servicios.

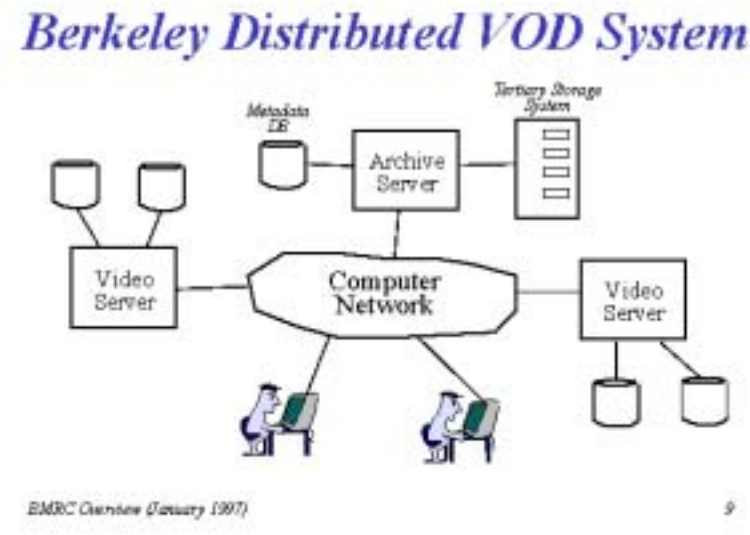
Actualmente el Vídeo bajo demanda incluye muchos más servicios y oportunidades. La tecnología actual permite a los operadores de red ofrecer servicios

como compra en casa, juegos y películas bajo demanda. Estos servicios deberían tener un precio competitivo comparable al alquiler de vídeo, y los clientes no tendrían necesidad de trasladarse a buscar los servicios. Esto se ha conseguido gracias al desarrollo de la industria de la telecomuniación y la electrónica.

Los servicios interactivos cubren un amplio rango de servicios desde películas bajo demanda hasta aprendizaje a distancia, seminarios bajo demanda, etc, y se pueden clasificar en distintas tecnologías:

- *Broadcast* (No-VoD). Servicios similares a la televisión por difusión, en los que el usuario es un participante pasivo y no tiene control sobre la sesión
- *Pay-per-view* (PPV): servicios en los que el usuario paga por una programación específica
- *Cuasi Vídeo bajo demanda* (Q-VoD): en estos servicios los usuarios se agrupan en umbral de interés. Los usuarios pueden realizar un control temporal muy simple cambiándose a un grupo diferente
- *Casi Vídeo bajo demanda* (N-VoD): servicios en los que el rebobinado hacia delante y hacia atrás se simulan por transiciones en intervalos de tiempo discreto. Esta capacidad se puede proporcionar mediante múltiples canales con la misma programación sesgada en el tiempo
- *Verdadero Vídeo bajo demanda* (T-VoD): el usuario tiene un control completo sobre la presentación de la sesión. Necesita solo un único canal por cliente.

Los servicios más fáciles de implementar son los de PPV y los más complicados son los de T-VoD, ya que en servicio como PPV donde el usuario quiere ver una película o un partido de futbol, un controlador local puede filtrar múltiples canales para conseguir el servicio. T-VoD requiere una señal bidireccional del usuario a un controlador centralizado. Los elementos necesarios para el uso del servicio completo son servidores de vídeo, la red local, oficinas de conmutación, unidades de conexión y



una red central.

Figura 2.9. Sistema de Vídeo bajo Demanda de Berkeley

Los proporcionadores ofrecerán servicios que seleccionen la tecnología correcta, características, rendimiento, precio, fiabiliad y facilidad de uso. Los equipos se

desarrollarán de forma que se permita operar en diferentes entornos y en una amplia variedad de servicios [68].

Nosotros estamos sobre todo interesados en la arquitectura de los servidores de vídeo, ya que podría ser que entre esos servidores la información se pudiera compartir mediante NFS y una optimización en el tiempo de acceso a la información mejoraría mucho el sistema global. Vamos a dar ahora un breve recorrido por distintos estudios que se están realizando en este tema.

Para ilustrar el estudio veamos un ejemplo de la arquitectura de un sistema de Vídeo bajo demanda de la Universidad de Berkeley.

Los servidores de vídeo utilizan NFS entre sus posibles sitemas de ficheros. Este sistema de ficheros es siempre un elemento presente en cualquier servidor de ficheros de vídeo. Los temas de investigación sobre los sistemas de VoD están relacionados con los propios servidores, los dispositivos de almacenamiento y el software asociado, la gestión de almacenamiento jerárquico (caching), el indexado y las consultas sobre el contenido, las interfaces de usuario, el flujo adaptativo y la distribución segura por redes heterogéneas.

Anteriormente se daba mucha importancia a la producción de datos de los servidores de VoD. En aplicaciones de Vídeo bajo Demanda Interactivas (IVoD), tales como librerías digitales, la disponibilidad de servicios y los tiempos de respuesta son más visibles al usuario que la producción de datos que hay por debajo. La producción de datos es una medida de la utilización eficiente de los recursos. En [33] se proponen diversas estrategias para controlar la secuencia de admisión de peticiones de vídeo pendientes. Los resultados muestran que estrategias híbridas pueden mejorar el número de peticiones admitidas y reducir el tiempo de espera en colas sin poner en peligro la producción de datos. Esas técnicas son independientes de las técnicas de planificación de discos que exista por debajo, así que se pueden emplear en mejorar el rendimiento que percibe el usuario de los servidores de Vídeo bajo Demanda, en general. También en [13] se exponen técnicas para reducir el tiempo medio de espera del usuario (la cantidad de tiempo que el usuario gasta en esperar que el vídeo comience), bajo diferentes cargas del servidor y configuraciones hardware. Y en [81]se exponen de forma general estrategias para minimizar el tiempo de acceso para aplicaciones multimedia distribuidas.

En [46], por otra parte, se estudian distintas aproximaciones para reconfigurar dinámicamente el sistema de VoD y poder alterar el número de copias de cada película que se mantiene en el servidor conforme la demanda de acceso a esas películas varíe. Un resumen muy interesante de las principales técnicas de planificación de los flujos de vídeo surgidas en los últimos años lo podemos encontrar en [28]

Tambien se proponen técnicas de broadcast para reducir los problemas de E/S del servidor, como la que se presenta en [35]

- Comercio electrónico y ASP

Otros servicios de reciente actualidad, nacidos y desarrollados gracias a la evolución de Internet, y que también podrían sacar un gran partido de este tipo de redes

en las que existen distintos servidores donde se almacena la información, son el comercio electrónico y los ASP (*Application Server Providers*).

Del comercio electrónica ya existen dos categorías claramente diferenciadas:

- El *Business-to-Business* (B2B): donde la venta es a otras empresas y por tanto las claves del éxito son la eficacia y la rapidez.
- El *Business-to-Consumer* (B2C): donde la venta es a clientes que lo utilizarán personalmente. El cliente podrá entrar pero no comprar siempre, por lo que la velocidad no es un factor tan importante como en el caso anterior. Aquí lo fundamental es una navegación sencilla y una forma de pago segura y cómoda.

Cualquier empresa que ofrezca un servicio de comercio electrónico deberá contar con un conjunto de máquinas en donde la información sea fácilmente accesible, y como esa información puede ser de cualquier tipo (datos, imágenes, vídeo, etc.) y se pueden tener distintos clientes accediendo al catálogo de productos, al carro de la compra o a los ficheros necesarios para realizar su firma digital o dejar su número de tarjeta, queremos hacer mención a este servicio como posible benefactor de las optimizaciones que se van a proponer en este trabajo. Para todos estos sistemas no es aconsejable usar un único servidor, por cuestiones de robustez a fallos y demás, por lo que por lo general se tendrán varios servidores compartiendo la información ya que pueden ser accedidos en cualquier momento [10].

Otro de los servicios interesantes a nivel empresarial lo forman los *Proveedores de Servicios de Aplicaciones.* Esto es algo que puede cambiar el mundo empresarial totalmente. Nos referimos a que una empresa no necesita tener programas que le gestiones sus datos, sino que enviará esos datos a otra empresa que les hará todo el procesamiento que necesiten y simplemente les devolverá los resultados. Es decir la idea es elegir las aplicaciones más difíciles de gestionar y traspasarlas a una empresa que garantice que las hará funcionar correctamente. Los ASP deben proporcionar garantías de seguridad, privacidad, disponibilidad, rendimiento y soporte y, como mínimo, algo de historial que avale esas garantías. Por lo tanto la disponibilidad de múltiples aplicaciones debe estar asegurada. Lo más importante es que los usuarios puedan acceder al recurso. Y aunque suene un poco a fábula, la empresa alquila aplicaciones y esto le proporciona un ahorro enorme en cuanto a licencias, hardware y costes de integración. Para proporcionar servicios a precios más agresivos, los ASP van a intentar hospedar numerosos clientes en un mismo hardware servidor [83].

¿Qué hay detrás de esos ASP? Pues por lo general una red de distintos servidores web, de aplicaciones web, de lógica de negocios, de bases de datos, de usuarios y de respaldo, que llevarán a conseguir los objetivos propuestos. Todos estos servidores van a compartir información, y posiblemente puedan hacerlo mediante un sistema de ficheros tan comercial y mejorado como NFS. Deben estar a punto para responder a las peticiones que se les pueda hacer en cualquier momento. Esas peticiones van a conllevar la lectura de distintos ficheros de todo tipo, o de parte de ficheros y escrituras de datos. Por lo tanto es una generalización perfecta de los problemas planteados en este estudio, y, por supuesto, la solución que planteemos optimizaría este tipo de sistemas.

En España y más cerca de nosotros, en Albacete, empresas como ONO ya ofrecen este tipo de servicios a nivel de empresa.

2.8.3. El codificador de vídeo MPEG-2

Para entender el proceso de compresión de vídeo, lo mejor es empezar exponiendo la entidad que vamos a manejar. Las señales de vídeo son señales espacio temporales, es decir, una secuencia de imágenes variando en el tiempo. Una imagen monocroma se puede representar matemáticamente por *x(h, v)*, donde x es el valor de la intensidad en una posición determinada por las coordenadas horizontal y vertical, h y v. La correspondiente señal de vídeo monocroma se representa por *x(h, v, t)*, donde x es el valor de la intensidad en las posiciones espaciales y temporales, h, v y t, respectivamente. Una señal de vídeo en color es simplemente una superposición de las distribuciones de intensidad de los tres colores primarios (R, G, B) o, de forma equivalente, de una luminancia (Y) y dos componentes de crominancia (U, V).

La forma más común de la señal de vídeo actualmente es, todavía, analógica. Esta señal se obtiene a través de un proceso conocido como escaneo, que puede ser progresivo o entrelazado.

Hay tres estándares de vídeo analógico conocidos como *Composite*, *Component* y *S-Video*. En el estándar *Composite*, la luminancia y las dos componentes de crominancia se codifican juntas como una señal única. Lo contrario al estándar *Component*, donde los tres colores primarios o los distintos componentes de color se codifican como tres señales distintas. El estándar *Composite*, incluye el formato NTSC que se ha adaptado en Estados Unidos y Japón, y el formato PAL/SECAM usado en Europa. El estándar *S-Video*, también referido como Y/C, consiste en señales de vídeo Y y C analógicas independientes.

Actualmente, la tecnología va dirigida a la completa integración de las industrias de telecomunicación, ordenadores y vídeo, en una plataforma multimedia. La señal de vídeo debe ser escalable, independiente de la plataforma, capaz de proporcionar interactividad y edición, robusta, y resistente a errores. La señal de vídeo analógica, por desgracia, falla en estos requisitos. Además la calidad de la señal se deteriora con múltiples reproducciones. Para añadir más problemas, los formatos de vídeo *Composite* presentan algunos artificios visuales.

El movimiento al campo digital no elimina todos los problemas mencionados, pero abre una puerta a un amplio rango de técnicas de procesamiento de vídeo que pueden definir mejor la imagen.

Para digitalizar la señal espacio-temporal *x(h, v, t)*, se realiza un muestreo de la señal analógica en las tres direcciones. Cada muestra es un cuadro se llama píxel. El proceso de muestreo produce el conjunto completo de parámetros necesarios para representar una señal de vídeo digital, como por ejemplo, número de píxeles por línea y número de líneas por cuadro, entre otros.

La evolución de los estándares de vídeo digital ha sido bastante firme. La recomendación CCIR 601 define el estándar para la industria de la televisión, como aparece en la siguiente tabla (tabla 2.11).

Tabla 2.11. Recomendación CCIR 601

PARÁMETROS	**CCIR 601 NTSC**	**CCIR 601 PAL**
Píxeles/línea activos (L)	720	720
Líneas/Imagen activas (L)	485	576
Caudal temporal	60 campos/s	50 campos/s
Razón de aspecto	4:3	4:3
Entrelazado	2:1	2:1

Con el objetivo de comprimir la señal, el primer paso consistirá en pasar a otro espacio de color en el que se utilicen menos bits, como es por ejemplo el formato de luminancias y crominancias, sin perder calidad de la señal, ya que el ojo humano es más sensible al brillo (luminancia) para poder percibir los detalles, que a las diferencias de color (crominancias). Así, en la mayoría de los estándares de compresión se realiza una transformación en el espacio de colores, desde el formato RGB a otro más conveniente [72].

Este muestreo implicará un ahorro de ancho de banda a tener en cuenta. Hay diferentes esquemas de muestreo para las crominancias [72], los más utilizados son: 4:4:4, 4:2:2, 4:2:0 y 4:1:1, como se puede ver en la siguiente tabla (2.12).

Las coordenadas de color YUV es el formato de colores básico utilizado por los sistemas de televisión estándares NTSC y PAL. La luminancia (Y) representa la componente de blanco y negro, y las crominancias U, V, las componentes de color. Este es también el formato utilizado en el sistema de televisión digital de alta definición *DigiCipher HDTV* propuesto por la *American Television Alliance* [20].

En el formato 4:4:4, la luminancia y las crominancias son muestreadas a la misma frecuencia, y por tanto, no existe ahorro del ancho de banda. En el formato 4:2:2, también llamado formato de muestreo de la recomendación CCIR 601, las señales de crominancia son muestreadas a la mitad de frecuencia que las de luminancia, en la dirección horizontal. En el formato 4:2:0, las señales de crominancia se muestrean a la mitad de frecuencia que las de luminancia, tanto en la dirección horizontal como en la vertical. El posicionamiento de las muestras de las crominancias puede variar según aplicaciones, en algunos casos las muestras de crominancias se centran alrededor de cuatro muestras de luminancia (estándar H.261, MPEG-1), y en otros, se centran entre dos muestras de columnas impares de muestras de luminancia (MPEG-2). En el formato 4:1:1, las señales de crominancia son muestreadas a un cuarto de frecuencia que las de luminancia, en la dirección horizontal.

Tabla 2.12. Formatos de crominancia para una imagen 720x480

Formato Croma	Y Muestras Por línea	Y Líneas por cuadro	CMuestras por línea	C Líneas por cuadro	Sub-muestreo horizontal	Sub-muestreo vertical
4:4:4	720	480	720	480	Ninguno	Ninguno
4:2:2	720	480	360	480	2:1	Ninguno
4:2:0	720	480	360	240	2:1	2:1
4:1:1	720	480	180	480	4:1	Ninguno

En relación al tamaño de las imágenes, en la recomendación CCIR-601 [23] se establecen dos opciones, en cuanto al número de líneas, frecuencia de imágenes y relaciones de aspecto.

Tabla 2.13. Opciones de codificación. Formato CCIR 601

	FRECUENCIA	**NºLÍNEAS**	**REL.ASPECTO (LUMINANCIA)**	**REL.ASPECTO (CROMINANCIA)**
América	30 Hz	525	720x480	360x480
Europa	25 Hz	625	720x576	360x576

La señal de vídeo puede ser entrelazada o progresiva. En vídeo entrelazado, cada cuadro de imagen se divide en dos campos, uno alto (*top field)* y otro bajo (*bottom field*), separados temporalmente por un periodo de campo. Los campos se forman al escanear la imagen en líneas alternativas, el campo alto se forma con las líneas impares de la imagen, y el campo bajo se forma con las líneas pares. Los dos campos se entrelazan posteriormente para formar la imagen original. En vídeo entrelazado, las líneas adyacentes en el espacio, no lo son en el tiempo.

En vídeo progresivo, no existe el concepto de campo, y sólo existe el de cuadro (*frame*). A diferencia del vídeo entrelazado, en el progresivo, las líneas adyacentes en el espacio, también lo son en el tiempo.

Las señales de vídeo procedentes de cámaras de TV están en un formato entrelazado, mientras que las de las películas están en formato progresivo. El formato progresivo es más eficiente cuando la escena contiene muchos detalles y poco movimiento, mientras que el entrelazado es más eficiente cuando la escena contiene movimientos rápidos [27]

Debido al creciente interés que han despertado un gran número de aplicaciones de vídeo, tales como librerías y documentos digitales, vídeo bajo demanda, videoconferencia, televisión de alta definición, etc., aumenta también la investigación y desarrollo de técnicas capaces de manejar el enorme volumen de datos a tratar, y de almacenar, recuperar y transmitir imágenes de una forma eficiente. La compresión de imágenes digitales se está convirtiendo, por tanto, en una tarea crucial en este sentido.

La transmisión de imágenes, con o sin movimiento, es una de las aplicaciones que consume más ancho de banda en la actualidad. El vídeo, como caso particular, se transmite casi invariablemente en forma comprimida. El objetivo principal al utilizar un algoritmo de compresión es, reducir las redundancias, espacial y temporal, presentes en

una secuencia de vídeo, manteniendo un nivel aceptable de calidad visual. Por lo tanto, la esencia del proceso de compresión es intentar alcanzar una representación más compacta de la señal digital, mediante la eliminación de redundancias presentes en la misma, para minimizar el caudal de bits necesario para su transmisión o almacenamiento, intentando mantener la calidad.

Muchas aplicaciones tratan con secuencias de imágenes que representan alguna forma de movimiento. Por ejemplo, una secuencia de imágenes tomadas desde satélite cada cierto tiempo sobre el mismo tramo de territorio, que después del apropiado reposicionamiento, son sucesivas instancias de la misma escena con pequeños cambios debidos al movimiento de objetos o al cambio de las condiciones meteorológicas. De este modo, en la compresión, se puede sacar provecho de ésto, bastará, por tanto, conocer solamente los cambios que se producen de una imagen a la siguiente.

En algunas áreas (como medicina) se tratan más las imágenes a niveles de gris, mientras que en otras, se utilizan imágenes en color (espectáculos y diversión). La calidad de una imagen procesada o comprimida se juzga de diferente manera según sea en color o a niveles de gris. En el caso del color, el que la calidad de la imagen sea aceptable, va a depender ampliamente de la aplicación específica. Por ejemplo, en HDTV, no importa un cambio en el color de la ropa de un presentador, por ejemplo, pero sí en el color de la piel (una cara verde no le favorece a nadie).

Realizando una primera pasada por los distintos métodos de compresión de imágenes, vídeo y audio que se pueden utilizar en la actualidad, se puedes hacer la siguiente clasificación:

- ***Esquema de compersión con pérdidas***
- ***Esquema de compresión sin pérdidas***

Un esquema de *compresión sin pérdidas* es el adecuado cuando se acepta una relación de compresión baja. En muchas aplicaciones, el decodificador tiene que reconstruir los datos originales sin ninguna pérdida. En este esquema, los datos reconstruidos y los datos originales deben ser idénticos para cada una de las muestras. Es un proceso reversible. La elección de este método implica un equilibrio entre los siguientes parámetros: complejidad del codificador y eficiencia, y retardos en la codificación.

La mayoría de las aplicaciones con imágenes o datos de vídeo no requieren que los datos reconstruidos y los originales sean idénticos, este esquema se denomina *con pérdidas*. Por lo tanto, se permiten algunas pérdidas y es un proceso de compresión irreversible. La elección de este método implica tener en cuenta los siguientes parámetros: complejidad del codificador, eficiencia y retardos en la codificación, y calidad de la señal. Muchas aplicaciones aceptarán compresión con pérdidas mientras los resultados de calidad sean buenos. En algunas aplicaciones críticas, tales como imágenes médicas y observación militar, no se pueden tolerar pérdidas de información, aunque se pueden utilizar las versiones comprimidas para consultas remotas y acceso rápido a la información.

En un análisis más detallado, las técnicas de compresión de imágenes y vídeo se pueden clasificar en los siguientes grupos:

1. *Codificación de Muestras*. En la codificación de muestras, se utiliza únicamente información de los pixeles o muestras individuales para comprimir la imagen. En estas técnicas se incluyen la Modulación de Impulsos Codificados (PCM), la Codificación de Longitud de Series y la Codificación de Entropía.

2. *Codificación Perceptiva*. Estas técnicas tienen como base el conocimiento de la percepción psicovisual del ojo humano. Una aplicación de la codificación perceptiva, sería la generación de las señales de diferencia de color y de la luminancia a partir de la señal RGB. El ojo humano no aprecia prácticamente ningún cambio en la calidad de señal, a pesar de la reducción del ancho de banda al submuestrear las señales de diferencia de color con algunos de los formatos de muestreo vistos.

3. *Codificación por Transformada*. Las técnicas de codificación por transformada, transforman la información a otro dominio donde los datos están mucho más decorrelados que en el dominio espacial, y la información se acumula en un pequeño número de muestras. Las transformadas rápidas más comunes que son utilizadas para realizar esta transformación y que presentan un buen comportamiento de compactación son, la Transformada de Karhunen Loeve (KLT), Transfornada de Fourier Discreta (DFT), Transformada Discreta de Coseno (DCT) y Transformada de Walsh Hadamard entre otras. La transformada elegida por la mayoría de los estándares establecidos, es la transformada DCT, ya que contiene coeficientes reales solamente, es una transformada rápida, y tiene una excelente compactación de la energía y decorrelación de los datos entre otras propiedades. Un excelente y detallado estudio de la DCT y de sus propiedades, así como de algoritmos para calcularla se pueden encontrar en [72]. Mediante la transformada DCT, se pasa de una representación espacial en pixeles a una representación de componentes de frecuencia, que es más adecuada para aplicar después técnicas de compresión con pequeñas pérdidas de información. Por esta razón, en la mayoría de los estándares de codificación, la codificación por transformada se combina con un *proceso de cuantificación* ponderado, donde la mayoría de los coeficientes de alterna son puestos a cero [72]. El *proceso de cuantificación* asigna por aproximación un valor a cada coeficiente de frecuencia dentro de una limitada gama de valores admitidos. El codificador utiliza una *matriz de cuantificación ponderada* que determina el modo en que será cuantificado cada uno de los coeficientes del bloque transformado. Esta matriz se diseña teniendo en cuenta el comportamiento psicovisual del ojo humano y el factor de compresión deseado. Es en la etapa de cuantificación donde se pierde parte de la información. Es por lo tanto un proceso con pérdidas o irreversible.

4. *Codificación Predictiva*. Las técnicas de codificación predictiva, explotan la correlación temporal y espacial de las imágenes para codificar eficientemente la información. La idea en la que se basan las técnicas predictivas[5], es eliminar la redundancia que existe en el dominio espacio-temporal en el que se representan las secuencias de imágenes, prediciendo el valor de un pixel a través

de su vecindad espacial o temporal, y codificar solo la diferencia entre los valores predichos y los valores actuales que tengan los píxeles. Algunas de las técncias predictivas son la Técnica predictiva DPCM (*Differential Pulse Code Modulation) y la Codificación Intercuadro* (Compensación de movimiento).

Una secuencia de vídeo es una serie de imágenes fijas que se suceden rápidamente para dar la sensación de movimiento continuo. Aunque cada una de estas imágenes es distinta de las adyacentes, la mayor parte de la información que contienen es muy similar. Esto quiere decir que hay una gran redundancia temporal entre las imágenes adyacentes. La técnica de *compensación de movimiento*, es un método con el que se pretende eliminar esta redundancia temporal, a través de evaluar de un modo eficiente el movimiento de los objetos de una imagen a la siguiente, de forma que este movimiento se tenga en cuenta al buscar la redundancia entre imágenes [5].

La diferencia entre una imagen, y la siguiente se debe fundamentalmente al movimiento de objetos. Con el método de compensación de movimiento, se crea una imagen de predicción con la imagen actual, o nueva, y la imagen precedente (ya codificada y en *memoria*) mediante la *estimación del movimiento* entre las dos imágenes y una compensación para el movimiento. La diferencia entre la imagen actual y la predicción que se ha hecho para esta imagen actual, es lo que se llama *residuo de movimiento compensado*. En una secuencia típica de vídeo, este residuo, que contendrá valores muy pequeños y próximos a cero, tiene una entropía mucho menor que la del vídeo original debido a la eliminación de la redundancia temporal

Para la estimación del movimiento se supone que en las imágenes de vídeo consecutivas, aparecen los mismos componentes de la escena aunque posiblemente en diferentes posiciones. El movimiento puede ser global o local dentro de la imagen y para optimizar el procedimiento se realiza la estimación del movimiento en cada región local de la imagen. El modelo más normal para el movimiento local es la traslación, que es mas bien restrictivo ya que no puede representar otros posibles movimientos como el de rotación o de cambio de escala.

La imagen de vídeo de entrada, imagen actual o nueva, se compara con una imagen transmitida previamente que se ha almacenado en memoria. Utilizando la estimación del movimiento, la comparación se hace mediante los macrobloques de la imagen previa que se examinan para determinar si hay alguno, dentro del *área de búsqueda* definida previamente, que presente la máxima semejanza con la imagen actual. Si ocurre así, se genera un *vector de movimiento* que describe la dirección y la distancia a la que el bloque se ha movido. Combinando todos los macrobloques en los que se ha encontrado una mayor igualdad se forma una imagen predecible (imagen referencia cambiada de posición, de acuerdo con los vectores de movimiento para cancelar el movimiento entre imágenes) que comparándola con la nueva imagen pixel por pixel, proporciona una imagen diferencia (residuo del movimiento compensado). Con predicción de movimiento compensado es necesario que el decodificador disponga de una imagen inicial de referencia, para arrancar el proceso de predicción

Para imágenes de vídeo típicas, la predicción mediante compensación de movimiento es muy satisfactoria. Sin embargo, en algunas imágenes como los de cambio de escena, la predicción no da buenos resultados. Incluso en una imagen determinada, algunas regiones pueden predecirse bien con compensación de

movimiento, pero no así otras, en las que no resulta adecuado el modelo de movimiento de traslación. El modelo de compensación de movimiento anteriormente descrito se conoce como *codificación intercuadro*.

Para solventar los problemas en los casos en los que no es conveniente aplicar codificación predictiva, hay que recurrir a la codificación de una imagen sin tener en cuenta otras. A este proceso se le conoce como *codificación intracuadro*, y en la mayoría de los estándares se lleva a cabo mediante una codificación basada en transformada DCT.

El proceso más costoso computacionalmente, es el de la estimación del movimiento, el cual debe de determinar el macrobloque en la imagen de referencia que mayor semejanza tiene con el macrobloque de la imagen actual. El criterio de semejanza utilizado en la mayoría de las ocasiones es aquel que minimiza el *Error Absoluto Medio* [5]. Este coste computacional de los algoritmos de búsqueda, es una de las razones por las que se define una *área de búsqueda*, para restringir la búsqueda a una área local del macrobloque en cuestión y reducir así su costo.

Por último, la compensación de movimiento vista aquí, reduce la redundancia temporal de la señal de vídeo, pero todavía queda redundancia espacial en el residuo del movimiento compensado que puede ser eliminado con algunas de las técnicas vistas anteriormente. Por esta razón, la mayoría de los estándares de compresión, utilizan un sistema híbrido de compresión con codificación intercuadro y transformada basada en DCT para obtener elevados factores de compresión [72]. Este sistema aplica una codificación intercuadro primero, para eliminar la redundancia temporal y una transformada DCT después, para eliminar la redundancia espacial

La clave de la compresión de imágenes es, codificar imágenes o secuencias de imágenes con tan pocos bits como sea posible, para que el mecanismo decodificador reconstruya la imagen original con información y calidad visual aceptable. Además, el tiempo invertido en comprimir y descomprimir una secuencia de vídeo, sobre todo en aplicaciones en tiempo real, o en aquellas en las que el caudal al que la fuente produce los datos es muy alto, también es un factor muy importante a tener en cuenta.

Hay dos estándares aceptados actualmente para compresión de imágenes estáticas y en movimiento, son JPEG (Joint Pictures Expert Group) y MPEG (Moving Pictures Expert Group). Estos esquemas, proporcionan altos factores de compresión con buena calidad en la imagen reconstruida. Sin embargo, la cantidad de cálculos requeridos para ambos es, generalmente, demasiado alta para aplicaciones en tiempo real.

La idea básica detrás de la codificación MPEG es, explotar la localidad temporal. Exceptuando ciertos tipos de vídeos, como vídeos musicales, las imágenes no cambian mucho en pequeños intervalos de tiempo. MPEG, toma ventaja de esto, codificando una imagen con relación a otras imágenes temporalmente cercanas a ella. [32]

El trabajo principal de MPEG será, por tanto, tomar señales de vídeo analógicas o digitales y convertirlas en paquetes de información digital, cuyo transporte es más eficiente en las redes actuales. MPEG comprime el vídeo en mucha menos información, consumiendo además, menos ancho de banda en la transmisión.

Las principales técnicas usadas por MPEG, , son las siguientes:

- Transformación del espacio de color RGB, lo que produce automáticamente un factor de compresión de 2:1.
- Codificación JPEG basada en DCT (*Discrete Cosine Transform*) y cuantización, seguidas de algún esquema de compresión sin pérdidas, que produce factores de compresión tan altos como 30:1, con una buena calidad de imagen.
- Compensación de movimiento, en la que una imagen puede codificarse en términos de las imágenes previa y siguiente.

Sin embargo, estas técnicas limitan bastante la velocidad a la que una secuencia de imágenes puede comprimirse. El algoritmo de compresión MPEG es bastante asimétrico, es decir, la cantidad de cálculos necesarios para la codificación es mucho mayor que la necesaria para la decodificación. Algunos codificadores MPEG requieren de 15 a 20 minutos de cómputo para comprimir 1 minuto de vídeo.

Es difícil alcanzar compresión de vídeo en tiempo real, y muy caro, cuando el tamaño de la imagen es grande, o se necesita una alta calidad de la misma con un bajo caudal de datos. Por ello, el procesamiento paralelo, se convierte en la elección natural para los elevados requisitos de computación, de la compresión de vídeo en tiempo real. En el capítulo 4 aparece una exposición del codificador paralelo desarrollado y utilizado, después, como aplicación multimedia en nuestro sistema experimental real.

Por la necesidad de establecer normas internacionales para estos esquemas de compresión, los organismos de estandarización mundiales han desarrollado diferentes estándares para el almacenamiento y transmisión de vídeo y su audio asociado. Algunos de ellos se muestran en la siguiente tabla (2.14)

Tabla 2.14. Estándares de vídeo

JBIG	Joint Binary Image Group
JPEG	Joint Photographic Expert Group
Motion JPEG	Motion Photographic Expert Group
ITU H.261	Video Codec for Audiovisual Services at px64 Kbps
MPEG-1	Moving Picture Experts Group.Digital Storage Media up to 1.5 Mbps
MPEG-2	Moving Picture Experts Group. Generic coding of moving pictures and

	associated audio
MPEG-4	Multiple bit-rate Audiovisual coding up to 1024Kbps for video and up to 64Kbps for audio
MPEG-7	Multimedia Content Description Interface
ITU H.263	Expert Group on Very Low bit-rate Video Telephony

En 1988, la ISO/IEC creó el MPEG (Moving Picture Expert Group) con el objetivo de desarrollar un estándar para comprimir vídeo para aplicaciones multimedia, su audio asociado y sincronizar los flujos de bits. Como resultado el grupo de expertos que formaban MPEG, cuya identificación formal es el grupo ISO/IEC JTC1 SC29 WG11 ratificó su primer estándar en 1991, el estándar ISO 11172, *Código de imágenes en movimientos y audio digital asociado para medios de almacenamiento digital para 1,5 Mbit/s*, MPEG-1. Este estándar, está dividido en tres partes (vídeo, audio y sistema), que posteriormente evolucionó a MPEG-2. Posteriormente se desarrollaron otros: MPEG-4, MPEG-7 y MPEG21. odos los estándares MPEG son genéricos, es decir, independientes de la aplicación. En los estándares desarrollados no se especifica como debe hacer el trabajo el codificador, sino que especifican la sintaxis del flujo de bits codificado y el proceso de decodificación, proporcionando una gran flexibilidad en las especificaciones, que permita a los implementadores disponer de gran libertad para optimizar el proceso [82].

El estándar MPEG-1

El algoritmo de codificación MPEG-1 es un algoritmo de codificación con pérdidas, aplicable a un amplio rango de formatos de entrada y de aplicaciones. Sin embargo, se ha optimizado para aplicaciones que soportan un caudal continuo de información de 1,5 Mbits/s. MPEG-1 no reconoce fuentes entrelazadas. El vídeo entrelazado, tal como el generado por una cámara de TV, debe convertirse a código no entrelazado antes de la codificación. Este estándar no especifica como debe realizarse esa conversión.

El estándar hace uso de una nueva técnica denominada predicción bidireccional, que consiste en almacenar las imágenes presentes, basándose en las imágenes pasadas y futuras a ésta. Más adelante se explicará con mayor detalle el proceso, aunque esta operación obligaba a codificar las imágenes en un orden diferente al orden en el que serían representadas posteriormente en la visualización. Esto motivó la creación de una estructura básica de decodificación independiente del flujo codificado denominada Grupo de Imagenes (GoP).

De entre las características más importantes que se propusieron al realizar este estándar tenemos algunas como acceso aleatorio de cuadros de vídeo en un tiempo limitado, posibilidades de búsqueda y rebobinado rápido, robusto frente a errores, sincronización entre señal de vídeo y de audio, retardo de codificación/descodificación, editabilidad, compatibilidad de formatos, soporte para una amplia gama de dimensiones

y espacios de color, etc, y no estar limitado a determinadas clases de imágenes en escena.

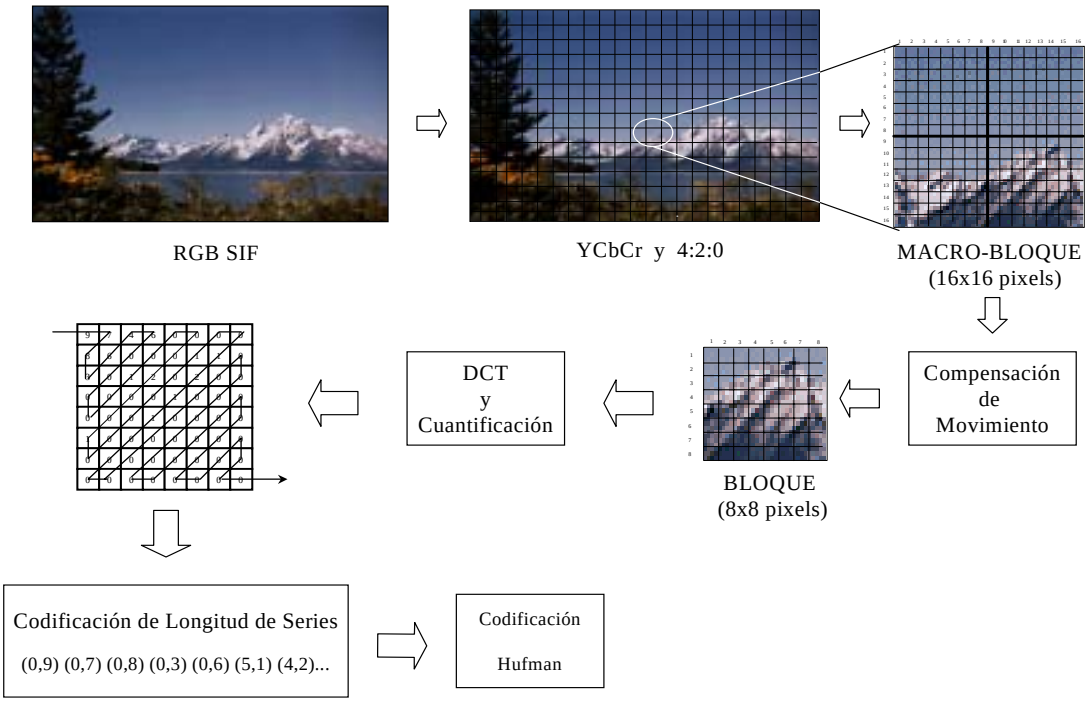


Figura 2.10. Etapas en el proceso de codificación MPEG-1.

MPEG-1 consigue compresiones óptimas en el rango de 1 a 1,5 Mbps, en secuencias de vídeo progresivo en color en formato SIF (*Source Input Format*) como NTFS o PAL, con un formato de muestro 4:2:0 y el espacio de color YCbCr. El factor de compresión obtenido con este estándar es aproximadamente de 26:1.

El estándar MPEG-1 se basa en (ver figura 2.10):

- Uso de la transformada discreta del coseno (DCT).
- Compensación de movimiento para reducir la redundancia temporal.
- Un proceso de cuantificación, codificación de longitud de series (RLC) y codificación de entropía mediante códigos Huffman.

El estándar impone unos valores mínimos a los parámetros que determinan el flujo de bits que como mínimo un decodificador MPEG-1 compatible debería soportar.

Tabla 2.15. Parámetros de codificación de MPEG-1.

PARÁMETROS DE CODIFICACIÓN	VALOR MÁXIMO
Tamaño de imagen horizontal	768 píxeles
Tamaño de imagen vertical	576 líneas
Macrobloques	396

Caudal de píxeles	396x25 macrobloques/s
Caudal de imágenes	30 imágenes/s
Tamaño del buffer de entrada	327,680 bits
Rango de los vectores de movimiento	+/- 64 píxeles
Caudal de bits	1,856 kbits/s

Las imágenes se dividen lógicamente en macrobloques de 16x16 píxeles. Cada macrobloque se divide en 6 bloques, 4 de ellos para la luminancia y 2 para las crominancias.

El estándar MPEG-1 posee tres tipos de imágenes codificadas o cuadros:

- *Imágenes de tipo I (Intra).* Utilizan únicamente información contenida en la propia imagen y no depende de otra. (Codificación Intra-cuadro). Se fundamentan en el estándar JPEG y proporcionan una moderada compresión.
- *Imágenes de tipo P* (*Forward Predicted Frames*). Utilizan para su codificación, la información contenida en el cuadro previo, ya sea de tipo I o de tipo P, más próximo (codificación Inter-cuadro). A esta técnica se le denomina predicción hacia delante y proporciona mayor compresión, aunque introduce una pequeña componente de error.
- *Imágenes de tipo B* (*Bidirectionally predicted frames*). Estas imágenes utilizan para su codificación, la información contenida en los cuadros pasado y futuro, ya sean de tipo I o de tipo P, más próximos. A esta técnica se le denomina predicción bidireccional y proporciona la mayor compresión de las tres clases de imágenes.

Las imágenes de tipo B no propagan errores, pues no se utilizan para calcular ninguna otra imagen, pero las imágenes I y P si, siendo las imágenes I las más críticas. Las imágenes I tienen su razón de ser en la necesidad de editabilidad y acceso aleatorio, si bien la frecuencia de inserción de imágenes de este tipo puede variar, según la variación de cambios en la escena. Para una secuencia de vídeo en el que la imagen es muy estática, la frecuencia de inserción de imágenes I puede ser menor que para una secuencia donde las imágenes experimentan rápidos cambios. También se puede fijar el número de imágenes de tipo B que se insertan entre imágenes de tipo I o P.

La sintaxis de la secuencia MPEG-1 está definida en una estructura de capas para facilitar el cumplimiento de los objetivos que se marcaron para el estándar. De esta forma el flujo de bits es una secuencia de entidades independientes entre sí. MPEG-1 está formado por seis capas.

Cada capa inferior incluye un número de elementos de la capa superior. Esta estructura jerárquica también se mantiene en el flujo de bits. Cada capa tiene una cabecera con información, seguida de un número de elementos de la siguiente capa.

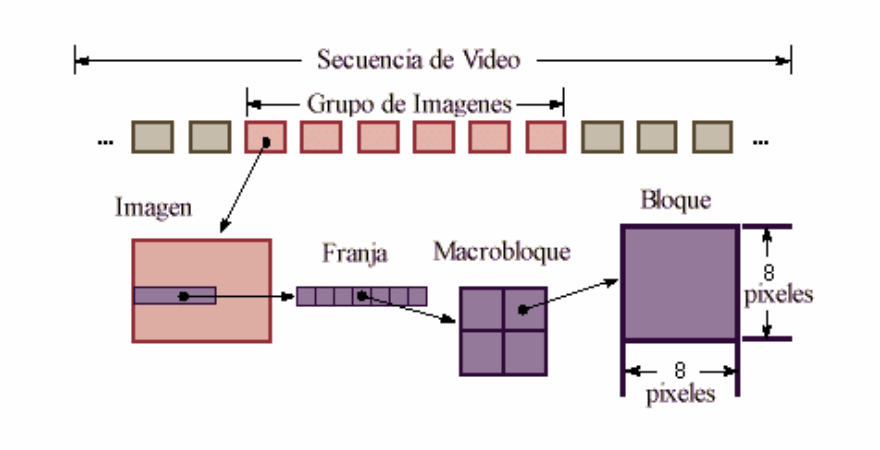


Figura 2.11. Estructura de una secuencia de vídeo MPEG-1

Cada secuencia es precedida de una cabecera de secuencia que contiene parámetros como el tamaño del cuadro o la relación de la dimensiones, seguida de un número de GoPs y finalmente una marca de fin de secuencia. Cada GoP es precedido por una cabecera con información de inicio de GoP y después por un número de cuadros. Cada uno de esos cuadros se descompone en un número de franjas, y cada franja en un número de macrobloques. Cada macrobloque se descompone en varios bloques. Podemos ver esta estructura en la figura 2.11.

De esta forma, posemos representar la sintaxis MPEG-1 para el flujo de bits de vídeo mediante una representación jerárquica de seis capas[5]:

1. *Capa de secuencia*: Define una secuencia completa de vídeo codificada, que puede ser independientemente visionada o editada. Aquí se definen parámetros como la dimensión de los cuadros, la razón de aspecto de los píxeles, si se cumple o no con el conjunto de parámetros restringidos o las matrices de cuantificación.

2. *Capa de grupo de imáges* (GoP, *Group of Pictures*). MPEG-1 introduce imágenes Intra (tipo I) con una frecuencia mayor para facilitar la edición. Divide la secuencia de vídeo en unas unidades más pequeñas que son independientes entre sí, y que pueden ser descodificadas por sí solas, es decir, una serie de imágenes en orden de presentación contiguos en el tiempo denominadas Grupos de Imágenes (GoP), y siempre son definidas al principio de la secuencia de vídeo. Su forma puede variar mientras cumpla la sintaxis de flujo de bits y tenga al menos un cuadro I, para que los demás puedan ser referenciados a él.

3. *Capa de imagen*: Esta capa posee toda la información necesaria para formar una imagen. La cabecera de cada cuadro contiene una referencia temporal del cuadro, el tipo de codificación (imagen de tipo I, B, o P), el rango de los vectores de movimiento, etc.

4. Capa de franja o slice Esta capa fue introducida para hacer el sistema más robusto, pues un error, sin utilizar esta capa supone el perder una imagen

completa, pero al usarla, la imagen se separa en franjas, por lo que sólo se pierde la parte que sucede en la franja al error producido. Por tanto es una capa que no es fundamental para el correcto funcionamiento del codificador. Al usar franjas, las predicciones para los vectores de movimiento y el coeficiente de continua de los bloques es iniciado en cada una de ellas. El tamaño horizontal de una franja puede ser cualquiera, desde un macrobloque hasta el cuadro entero, si bien se suele tomar una franja vertical completa.

5. *Capa de macrobloque*: Cada franja está formada por macrobloques. El macrobloque es la unidad básica de compensación de movimiento. Cada uno de estos macrobloques está formado por cuatro bloques de luminancia y dos de crominancia. El orden en el que se leen los macrobloques es de izquierda a derecha y de arriba abajo. Algunos de los parámetros que podemos encontrar en los macrobloques son la posición del macrobloque en la franja, el factor de escala Q de cuantificación, el patrón de codificación de los bloques del macrobloque, el tipo de macrobloque, los vectores de movimiento para ese macrobloque, etc.

6. *Capa de bloque*: Cada macrobloque se descompone a su vez en bloques de 8x8 valores, que pueden representar bien píxeles reales (Intra) o valores del residuo del movimiento compensado (Inter). Es sobre estos bloques sobre los que aún se aplica la DCT, la cuantificación RLC y la codificación de entropía mediante Huffman, lo que elimina la redundancia espacial, pues hasta ahora sólo se había eliminado la redundancia temporal.

🎬 El estándar MPEG-2

La segunda fase de MPEG, llamada MPEG-2, y concluída en 1994 también consta de tres partes o estándares, cubiertas por la: ISO/IEC 13818-1 Sistemas MPEG-2 (Draft ITU-T Rec. H.222), ISO/IEC 13818-2 Vídeo MPEG-2 (Draft ITU-T Rec. H.262) y ISO/IEC 13818-3 Audio MPEG-2. Estas fueron aprobadas finalmente como Estándar Internacional (IS) por la asamblea N° 29 de la ISO/IEC JTC1/SC29/WG11 (MPEG) hecha en Singapore en Noviembre de 1994.

El registro ITU-T H.262 trata con codificación de vídeo de alta calidad con posible vídeo entrelazado de NTSC, PAL o Televisión de Alta Definición (HDTV). Esto es un intento para operar en un rango de 2 a 15 Mbit/s. Sin embargo puede funcionar a velocidades superiores de 100 Mbit/s. Se tratan un amplio rango de aplicaciones, velocidades, resolución, calidades de las señales y servicios, incluyendo todas las formas de medios de almacenamiento digital, televisión (incluyendo HDTV), broadcasting y comunicaciones.

Entre las varias mejoras o extensiones introducidas en los codificadores MPEG-2, podemos destacar las siguientes:

- Nuevos modos de predicción de campos y tramas para scaneo entrelazado.
- Cuantización mejorada.
- Nuevos códigos intra-trama de longitud variable (VLC).
- Extensión escalada de resoluciones para compatibilidad, servicios jerárquicos y robustos

- Dos nuevas capas de sistema para multiplexaje y transporte que provee celdas/paquetes de vídeo de alta o baja prioridad, cuando son llevados a través de una red conmutada.
- Incrementos soportados por accesos aleatorios.
- Soporte resistente para incremento de errores.
- Compatibilidad con MPEG-1.

Al igual que el H.261 y JPEG (Joint Photographic Expert Group), el estándar MPEG-2 es un esquema híbrido de compresión para imágenes en pleno movimiento que usa codificación inter-trama y codificación intra-trama y combina la codificación predictiva con la codificación con transformada DCT. La DCT se aplica, por lo general, a un bloque de 8x8 elementos de imagen, dentro de un cuadro. Elimina redundancia en la imagen a través de la compresión de la información contenida en 64 píxeles. El cuantizador otorga los bits para los coeficientes DCT más importantes.

El concepto de MPEG-2 es similar al MPEG-1, pero incluye extensiones para cubrir un amplio rango de aplicaciones. La principal aplicación destinada durante el proceso de definición de MPEG-2 fue la transmisión de vídeo con calidad de TV codificado a velocidades entre 5 y 10 Mbit/s. Su estructura es similar a la de MPEG-1. Sin embargo, la característica más destacada con respecto a MPEG-1 es la sintaxis para codificación eficiente de vídeo entrelazado.

MPEG-2 es una recomendación muy compleja, tiene una gran variedad de aplicaciones posibles. Sus parámetros se clasificaron en un reducido conjunto de combinaciones que se definen como *perfiles* y dentro de éstos determinados *niveles* para definir la sintaxis de la secuencia de bits y facilitar el diseño y la implementación. La combinación de un perfil y un nivel produce una arquitectura muy bien definida para una cadena particular de bits. Los perfiles limitan la sintaxis (por ejemplo los algoritmos), hay cinco diferentes perfiles y cada uno es progresivamente más sofisticado y agrega herramientas adicionales. Mientras, los niveles limitan los parámetros (velocidad de muestreo, dimensiones de las tramas, velocidad binaria codificada, etc.), proveen un rango de cualidades potenciales, definen los máximos y mínimos para la resolución de la imagen, muestras Y por segundo (luminancia), el número de capas de audio y vídeo soportados por los perfiles escalados, y la máxima velocidad binaria por perfil. Se definieron cuatro niveles posibles. [52, 82, 76, 1, 34].

Por otro lado, La sintaxis de MPEG-2 puede ser *escalable* o *no escalable*.

- La sintaxis *no escalable* es un superconjunto de la sintaxis de MPEG-1, con extensiones adicionales que soportan la codificación de señales de vídeo entrelazado, caudal de vídeo variable, escaneo alternado, etc.
- La sintaxis *escalable* permite la codificación por capas de un flujo de vídeo. Se estructura el flujo de bits en una capa base y dos de realce (la de relación señal ruido (SRN) escalable y la Espacial escalable). La capa base puede usar la sintaxis no escalable. El motivo de usar esta estructura de capas es facilitar el proceso de descodificación y prevenir ambigüedades.

Actualmente hay cuatro modos escalables en MPEG-2. Estos modos rompen el vídeo MPEG-2 en diferentes capas (base, media, y alta) para propósitos de prioritización de datos de vídeo o para divisiones complejas

Por otro lado, el multiplexaje y transporte definidos bajo MPEG-2 especifican el formato de codificación para multiplexar audio, vídeo y datos dentro de una forma manejable para almacenar o transmitir. Cada flujo de bits elemental se trocea en cadenas elementalmente paquetizadas (*PES Packetized Elementary String*).

Terminando ya el resumen de MPEG-2, añadir algo más de sus campos de aplicación. Ha revolucionado la transmisión de vídeo digital de alta calidad y podemos encontrarlo ya en televisión de alta definición o en sistemas de almacenamiento de vídeo como el DVD. Algunas de sus aplicaciones actuales:

- *DBS (Emisión Directa de Satélite):* El servicio Hughes/USSB usa vídeo y audio MPEG 2. Hughes/USSB DBS comenzó a dar servicios en Norte América en 1994. Dos satélites a 101 grados al oeste comparten los requerimientos de poder de unos transponder con 120 Watts y 27 MHz.

- *CATV (Televisión por cable):* La industria del cable ha colocado más o menos vídeo MPEG-2. Y el audio en menor proporción.

- *DigiCipher*: La sintaxis del DigiCipher I es similar a la del MPEG 2, pero usa pequeños macrobloques de predicción y no tramas B. La especificación DigiCipher II incluye modos para soportar tanto GI como sintaxis del perfil principal de vídeo MPEG 2. Servicios como HBO fueron actualizados con DigiCipher II en 1994.

- *HDTV:* en Estados Unidos, Grand Alliance (un consorcio de compañías que formalmente compiten por el estándar HDTV) ha agregado el uso de vídeo MPEG-2 y sintaxis de los sistemas (incluyendo imágenes o tramas B). Serán soportados ambos modos, entrelazado (1440x960x30 Hz) y progresivo (1280x720x60 Hz).

En septiembre de 1993, un consorcio de 85 compañías europeas firmó un convenio para invertir en un proyecto conocido como Emisión de Vídeo Digital (DVB) el cual desarrolló un estándar para transmisión terrestre y de cable, y éste esquema usa MPEG-2. Este consorcio puso el último clavo en el ataúd del esquema D-MAC, para una migración gradual hacia un todo digital: HDTV [52, 76, 34].

También en el campo de la investigación sigue siendo objeto de interesantes estudios. Algunos ejemplos podrían ser los trabajos de mejora del tráfico MPEG en Internet [49], el estudio de un cliente de vídeo MPEG-2 de banda ancha en [36], el estudio de trazas de vídeo codificado MPEG-2 sobre enlaces de satélite en [11], el modelado de fuentes escalables MPEG en [16] y la codificación en tiempo real de vídeo MPEG-2 usando múltiples DSPs en [41].

El estándar MPEG-4.

Se corresponde con el estándar ISO/IEC JTC1/SC29/WG11 N3444. MPEG-4 es el resultado de otro esfuerzo internacional que involucra centenares de investigadores y ingenieros de todo el mundo. MPEG-4 cuya designación del estándar ISO/IEC es

ISO/IEC 14496. Fue finalizado en octubre de 1998 y se convirtió en una Norma Internacional en los primeros meses de 1999.

Las extensiones MPEG-4 Versión 2 son totalmente compatibles hacia atrás y fueron ratificadas a finales de 1999, para adquirir el estado de Norma Internacional formal a principios del 2000, si bien es cierto que todavía están en marcha algunos desarrollos, en las extensiones en dominios específicos. Es un nuevo proyecto desarrollado por el grupo MPEG, destinado a aplicaciones de comunicación multimedia, a muy bajo caudal, en las que se requiere comunicación de información audiovisual, ya sea en tiempo real o no, con posibilidad de interactividad con el usuario.

MPEG-4 proporciona elementos tecnológicos que permiten la integración de la producción, distribución y el acceso de contenidos de los tres campos. La norma MPEG-4 proporciona un juego de tecnologías para satisfacer necesidades demandadas por autores, proveedores de servicio y usuarios terminales. Un uso típico de este estándar será la transmisión de vídeo a bajo caudal por Internet y su interoperabilidad con las aplicaciones web.

El estándar MPEG-4, por el contrario a como lo hacen MPEG-1 y MPEG-2, que se basan en la codificación de cuadros de vídeo, realiza una codificación *basada en objetos*. Estas unidades de audio, visuales o audiovisuales, llamadas *objetos audiovisuales*, pueden ser de origen natural o sintético. Esto significa que podrían grabarse con una cámara o micrófono, o generarse con una computadora y pueden unirse. Describe la composición de estos objetos para crear objetos audiovisuales compuestos que forman las escenas audiovisuales.

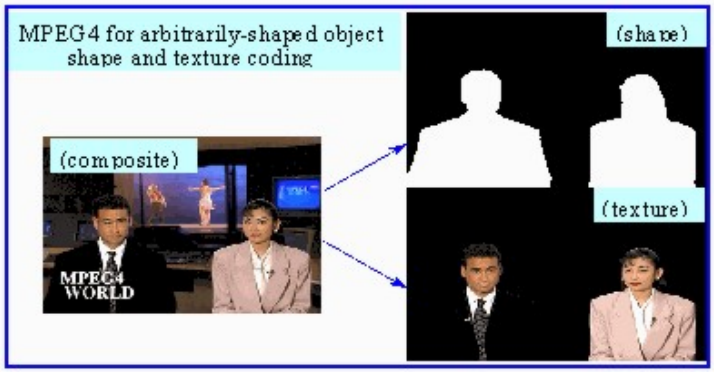


Figura 2.12. Composición de una secuencia MPEG-4.

Se debe multiplexar y sincronizar los datos asociados a los objetos, por lo que deben ser transmitidos por canales de red que proporcionen una determinada calidad de servicio (QoS). En la figura 2.12 se puede ver un ejemplo de esta tecnología.

Como podemos ver, la secuencia se puede separar en el fondo y los objetos dinámicos, que a su vez tienen una forma y una textura. En el codificador, los objetos audiovisuales y sus relaciones temporales son utilizadas por el codificador para generar los flujos de bits codificados. Estos flujos de bits, tras una opcional protección de errores, son multiplexados con objetos almacenados localmente y transmitidos. Los flujos de bits pueden ser transmitidos a través de múltiples canales, donde cada canal puede ofrecer diferentes calidades de servicio. El multiplexador del sistema MPEG-4

combina flujos elementales de datos en un único flujo de bits. El demultiplexador proporciona las funciones necesarias para recuperar el reloj del sistema, sincronizando los múltiples flujos de bits, etc.

Antes de que los objetos audiovisuales sean transmitidos, el codificador y el decodificador intercambian información de configuración. Esto permite al codificador determinar que clase de algoritmos, herramientas y otros objetos necesita el decodificador para procesar los objetos audiovisuales. Posteriormente, las definiciones de las clases de objetos no existentes en el decodificador son enviadas por el codificador. Esto no sucedía en MPEG-1 y MPEG-2, donde las capacidades del decodificador eran asumidas por defecto por el codificador.

Al igual que los anteriores estándares, MPEG-4 también realiza una representación por capas de la información. Cada cuadro de vídeo se segmenta en un número de objetos, llamados *planos de objeto de vídeo* (VOP). Sucesivos VOPs pertenecientes a un mismo objeto físico se denominan *objetos de vídeo* (VO). Un VO podríamos verlo como un GOP en los estándares MPEG-1 y MPEG-2. La forma, el movimiento y la información de la textura de los VOPs pertenecientes a un mismo VO son codificados en una *capa separada de objeto de vídeo* (VOL). Información adicional necesaria para identificar cada una de los VOLs y como se componen los diferentes VOLs son también codificados. Esto permite la descodificación selectiva de VOPs y proporciona también una escalabilidad a nivel de objeto en el decodificador.

Para la descodificación de la forma de los VOPs se suele recurrir a técnicas de compresión sin pérdidas, aunque también se da la posibilidad de aplicar codificación con pérdidas para obtener mayores factores de compresión.

Cada VOP se divide en un número de macrobloques, donde cada macrobloque que forma el VOP, se divide en cuatro bloques de luminancia y dos de crominancia. Para la codificación de su movimiento se utilizan técnicas de estimación y compensación de movimiento similares a los usados en MPEG-1.

Para la codificación de su textura, tanto los VOPs intra como sus residuos de movimiento compensado, son codificados utilizando una DCT 8x8. Después de aplicar la DCT, los coeficientes son cuantificados, leídos en orden de zig-zag para su codificación mediante RLC. Para los valores de DC y AC se utilizan eficientes predicciones tanto para bloques intra como Inter para su codificación.

La información de la forma del VOP, vectores de movimiento e información de los coeficientes DCT son multiplexados para formar el flujo de bits del VOL. En un escenario típico de codificación, el 75% de la información corresponde a las texturas de los objetos, el 14% a información de la forma o contorno y un 4.2% a información de movimiento. El resto es información de control utilizada por el codificador.

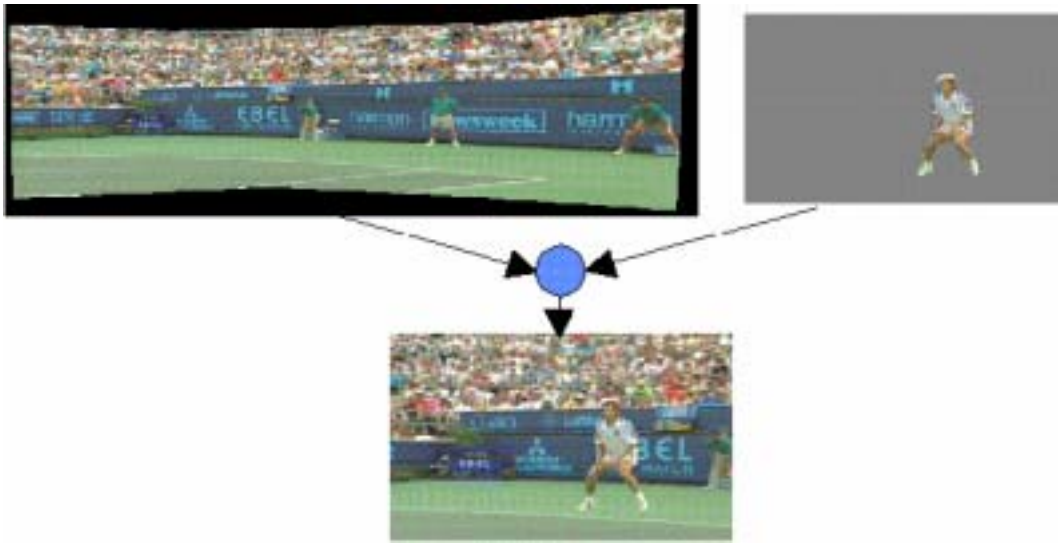


Figura 2.13. Ejemplo de codificación de una secuencia de Vídeo MPEG-4

Las escenas audiovisuales MPEG-4 están compuestas de varios objetos, organizados de forma jerárquica. En los niveles más bajos de esa jerarquía, encontramos objetos primitivos, como:

- Imágenes (como un fondo fijo, por ejemplo),
- Objetos de vídeo (una persona hablando - sin el fondo)
- Objetos de audio (la voz asociada a esa persona), etc.
- Otros objetos como texto, etc.

La representación de cada objeto es independiente del resto de objetos de la escena y su codificación se hace lo más eficientemente posible, de forma que se van asociando objetos (que están relacionados de forma arborescente), para formar objetos más complejos y en definitiva la escena, y su estructura, se asemeja al lenguaje de realidad virtual VRML, ampliamente utilizado en la actualidad [54, 40].

MPEG-4 se ha desarrollado para ser usado en:

- Televisión digital
- WWW.
- Multimedia interactiva (distribución de/acceso a contenidos), etc.
- Almacenamiento de proyecciones en vídeo digital.

El estándar MPEG-7.

Los estándares MPEG-1 y MPEG-2 fueron diseñados para el almacenamiento y transmisión de las señales de vídeo y audio. MPEG-4 tenía objetivos parecidos aunque orientado a las aplicaciones multimedia y con una filosofía muy diferente, basada en objetos. Aun así todos ellos tenían como objetivo extender el uso de la información audiovisual. Sin embargo, cuando el volumen de este tipo de información empieza a hacerse demasiado grande, encontrar cierta información empieza a ser algo complicado. Actualmente existen soluciones para búsqueda en texto. Sin embargo para otro tipo de contenidos multimedia no existe ningún estándar.

El grupo MPEG se puso en ello y fruto de esta investigación ha surgido el estándar *Multimedia Content Description Interface*, conocido por MPEG-7, cuya denominación por la ISO/IEC es JTC1/SC29/WG11 N3445, que fue aceptada en junio del 2000, y que no pretende comprimir aún mejor el vídeo y el audio, sino que es un estándar de localización de información en grandes redes (Internet), basándose en descripciones muy minuciosas de los diferentes elementos como vídeo-clips de vídeo, piezas musicales, fotografías, textos, etc., basándose en notas, colores, texturas, etc. El creador del archivo ha de indicar toda una serie de datos específicos a MPEG-7, antes de colocar el mismo en la red. Así, cuando se realice una búsqueda, el motor de localización de MPEG-7 discriminará con suma exactitud todos los elementos que no se correspondan con la petición realizada.

Su uso no se orienta a ninguna aplicación particular. Este será el final de las 37.456 páginas encontradas que nos puede devolver cualquier buscador Internet actual en respuesta a una búsqueda cualquiera [54].

Se beneficiarán de los servicios de MPEG-7 las bibliotecas digitales, los servicios de directorios multimedia, selección de canales de TV Digital o Radio y las ediciones multimedia.

El estándar MPEG-21.

Es el último estándar en el que trabaja MPEG, MPEG-21, *Multimedia Framework*, denominado por la ISO/IEC como JTC1/SC29/WG11/N3162, ha empezado a ser estudiado en junio de 2000. Su objetivo es definir un marco de trabajo multimedia de manera formal y resolver problemas que se vienen arrastrando desde hace tiempo, como relación entre los distintos elementos multimedia, etc.[54].

2.8.4. Una necesidad de entendimiento

A veces se olvida que la red no es el final, es solo un medio para posibilitar aplicaciones distribuidas. Es importante recordar que las aplicaciones determinan la red (o al menos, así debería ser). Es muy común que los diseñadores de red piensen de abajo a arriba, es decir, seleccionando primero tecnologías y productos, y determinando después qué aplicaciones podrían soportar las redes resultantes. Un buen diseño de red debería siempre empezar desde los requisitos de la aplicación. La necesidad de los usuarios de la red debería determinar la tecnología y equipos que mejor van a realizar las tareas

El término *entorno de aplicación,* conlleva un amplio rango de factores, entre los cuales está el número y tipo de usuarios, la distribución geográfica, los tipos de equipos, la arquitectura de la aplicación (cliente-servidor, por ejemplo), y los programas de aplicación en uso y sus requisitos de comunicación de datos [73].

De ahí la necesidad de seguir investigando para conseguir mejores elementos, protocolos, dispositivos y capas sioftware que lleven a la mejor explotación de los recursos teniendo siempre en cuenta las aplicaciones de interés que sobrela red se van a manejar. Nuestro estudio va orientado en esta dirección, con este trabajo queremos aportar una mejora importante en una capa software muy utilizada en arquitecturas cliete-servidor a la hora de compartir información, más concretamente, información

multimedia, aunque, como más adelante veremos, las optimizaciones conseguidas benefician a cualquier tipo de información. Pero, lo que ocurre es, que los problemas más serios van a aparecer con el tratamiento de la informaciòn multimedia y por eso nuestra motivación y entusiasmo de mejora.

2.8.5. Ejemplos de cluster orientados al trabajo multimedia

El número de trabajos que existen actualmente relacionados con la computación paralela en clusters dedicados es innumerable, podemos encontrar interesante trabajos en multitud de importantes conferencias internacionales y en la mayoría de las universidades de todo el mundo.

Hemos visto también un resumen, en secciones anteriores, de alguno clusters comerciales. Y, profundizando más, también hay que destcar el desarrollo de productos software tales como *mosix*, orientados a dar capacidad a las máquinas de soportar computacion de clusters flexibles y conseguir balanceo de carga y reparto de trabajos [53].

Por ello, voy a concluir este capítulo con algunos de los cluster utilizados en otras universidades también para realizar dintistos trabajos con aplicaciones multimedia, así quedará demostrado que es un campo de gran interés y que muchos investigadores dedican sus horas diarias de trabajo a temas relacionados.

La Universidad de Chicago es un claro ejemplo de clusters aplicados donde existen distintos clusters dedicados a clases de visualización y multimedia, capacidades de vídeo conferencia y laboratorio de medios digitales [12]. En la Universidad de Clemson tambien existe clusters dedicados usados en investigación de electromagnética computacional, comunicaciones inalámbricas y computación paralela. En esta universidad se han desarrollado otros importantes trabajos como el cluster Beowulf y el sistema de ficheros virtual paralelo [14]. También son destacables los trabajos realizados en la Universidad George Washington con clusters dedicados a probar modelos paralelos para la recuperación, basada en contenido, de bases de datos multimedia [85]. El cluster Kaláka de la Universidad de Auckland, Nueva Zelanda, de 214 nodos, dedicado a computación paralela y elemento básico de numerosos proyectos internacionales muy importantes [84]. Los clusters de Macintosh orientados también a la computación paralela de UCLA y de otras universidades en todo el mundo [4]. La interfaz de fibra óptica paralela para aplicaciones multimedia orientadas a cluster desarrollada en la universidad del sur de California [71]. Y miles de ejmplos más que podríamos poner.

En general, en todos los centros de computación europea de todo el mundo existen cluster dedicados al procesamiento paralelo y orientados a determinadas aplicaciones, muchas de ellas, multimedia, por supueto [78].

CAPÍTULO 3

ARQUITECTURA DEL CLUSTER MULTIMEDIA

3.1. INTRODUCCIÓN

En este capítulo examinaremos a fondo los componentes del cluster utilizado en este trabajo, y que en mayor o menor parte tienen mucho que ver con las prestaciones finales del sistema. Daremos un recorrido por todos ellos, describiéndolos y sentando las bases de las mejoras propuestas en capítulos posteriores.

Para diferenciar claramente estos elementos, diseñamos una arquitectura, en la que apilamos los distintos aspectos que van a intervenir en nuestra plataforma (Figura 3.1).

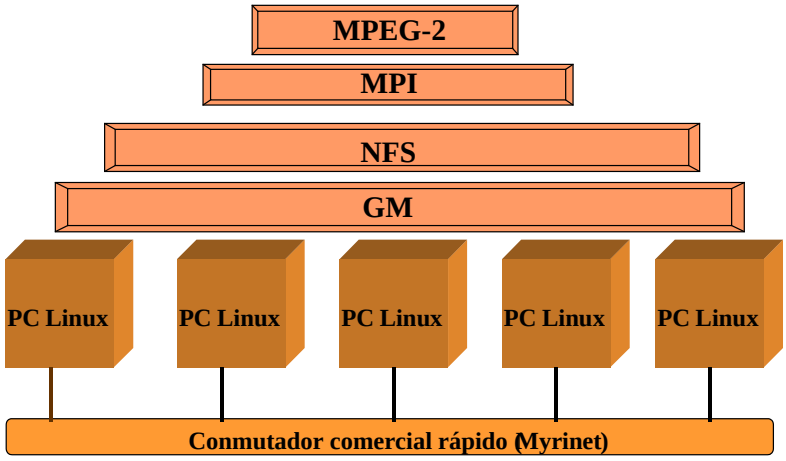


Figura 3.1. Arquitectura de la plataforma multimedia

En la capa más alta de la arquitectura se sitúan las aplicaciones que vamos a utilizar, y que nos van a servir de banco de pruebas en todos nuestros experimentos.

Utilizaremos, el codificador de vídeo MPEG-2 en sus versiones paralelas, desarrollado en su versión inicial en la Universidad de Hong-Kong y que hemos mejorado y modificado con distintas implementaciones, explotando distintas posibilidades que nos van a ayudar a sacar conclusiones sobre nuestra plataforma y, en particular, sobre aspectos concretos, como la importancia del sistema de ficheros utilizado, que ya iremos desarrollando en este capítulo y en los posteriores.

Bajo esta capa, la librería de paso de mensajes utilizada. Veremos asuntos tan importantes como el reparto de la carga y las soluciones adoptadas. Esta es la capa que nos va a ayudar a que todo funcione bien en paralelo, la capa que nos va a permitir implementar nuestro programa paralelo y que se va a entender con la red que coloquemos en el nivel más inferior, utilizaremos MPI (*Message Passing Interface)*. Y como implementación posible y efectiva de MPI, MPICH [135], después de haber probado otras opciones como LAM o BIP, que después comentaremos.

A continuación, nos enfrentamos con una capa de software intermedio (*middleware*) donde se ubican los sistemas de ficheros distribuidos, para compartir la información multimedia. Daremos un breve recorrido por los principales, los más maduros como NFS y AFS, otras soluciones más recientes como Samba y Coda, y otras más experimentales como Trapeze. Pero, sobre todo, veremos la importancia que va a tener este módulo en los resultados de rendimiento finales, así que será el principal elemento que vamos a probar, ajustar y optimizar hasta encontrar la solución más adecuada para nuestra plataforma final.

Por último describiremos la capa base, donde aparecen las redes utilizadas en estas pruebas. Describiremos brevemente la plataforma utilizada en la Universidad de Hong-Kong, un cluster de 16 estaciones de trabajo Sun Ultra-1 conectadas mediante un conmutador ATM, que sirvió de punto de partida para todo el trabajo posterior, y la que hemos utilizado en nuestro grupo de investigación (RAAP, Redes y Arquitecturas de Altas Prestaciones), un cluster de 8 Pentiums-II basado en Linux conectados mediante conmutadores Myrinet, que abarca casi la totalidad de experimentos. También describiremos el software asociado a Myrinet, como gm.

3.2. EL CODIFICADOR PARALELO DE VÍDEO MPEG-2

3.2.1. Introducción

La transmisión de imágenes, con o sin movimiento, es una de las aplicaciones que consume más ancho de banda en la actualidad. El vídeo, como caso particular, se transmite casi invariablemente en forma comprimida. La esencia del proceso de compresión es intentar alcanzar una representación más compacta de la señal digital, mediante la eliminación de redundancias presentes en la misma, para minimizar el caudal de bits necesario para su transmisión o almacenamiento, intentando mantener la calidad. Los estudios de compresión de imágenes van orientados, generalmente, a analizar la calidad de la reconstrucción de la imagen comprimida, el factor de compresión alcanzado, y la complejidad y velocidad del algoritmo en sí.

Una imagen, una secuencia de vídeo o las señales de audio se pueden comprimir debido a los siguientes factores:

- Hay una considerable redundancia estadística en la señal
- Hay bastante información en la señal, que es irrelevante desde el punto de vista perceptual humano.

Para una aplicación dada, los esquemas de compresión, pueden explotar uno o todos los factores anteriores, para alcanzar el factor de compresión de datos deseado.

Como se vio en el capítulo 2, se han establecido normas internacionales para estos esquemas de compresión, los organismos de estandarización mundiales han desarrollado diferentes estándares para el almacenamiento y transmisión de vídeo y su audio asociado, como JBIG, JPEG, MPEG, etc. [87]. Los fundamentos de MPEG también han sido descritos de forma general en el capítulo 2.

Hay dos estándares aceptados actualmente para compresión de imágenes estáticas y en movimiento, son JPEG (Joint Pictures Expert Group) y MPEG (Moving Pictures Expert Group). Estos esquemas, proporcionan altos factores de compresión con buena calidad en la imagen reconstruida. Sin embargo, la cantidad de cálculos requeridos para ambos es elevadísima.

La idea básica detrás de la codificación MPEG es, explotar la localidad temporal. Exceptuando ciertos tipos de vídeos, como vídeos musicales, las imágenes no cambian mucho en pequeños intervalos de tiempo. MPEG, toma ventaja de esto, codificando una imagen con relación a otras imágenes temporalmente cercanas a ellas.

El algoritmo de compresión MPEG es bastante asimétrico, es decir, la cantidad de cálculos necesarios para la codificación es mucho mayor que la necesaria para la decodificación. Por lo que es muy necesario obtener un tiempo de codificación que mejore los resultados de la versión secuencial. Además, una implementación software del codificador es más deseable, efectiva y flexible que algunas realizaciones con hardware de propósito especial, ya que permite mejoras algorítmicas para dar nuevas soluciones.

Así, explotaremos el enorme poder computacional ofrecido por sistemas de computación paralelos con el codificador de vídeo MPEG-2 basado en software. Presentaremos cuatro métodos diferentes de distribución de datos sobre un grupo de procesadores, con paralelismo espacial y temporal.

3.2.2. Necesidad de paralelismo

Se han realizado algunas aproximaciones, muchas de ellas usando hardware especial, para paralelizar las operaciones del *codec* con las secuencias de vídeo. Por

ejemplo, uno de los primeros trabajos fue la codificación de vídeo en movimiento completo en CD-I en un computador paralelo, donde el paralelismo se consigue dividiendo las etapas del *codec* en tareas, y asignando una tarea a un grupo de procesadores. Esto tiene el siguiente problema, deben leerse muchas imágenes antes de que todos los procesadores tengan algunas tareas que ejecutar. Además, esta implementación requiere hardware de propósito especial. El codificador original se desarrolló en un computador VAX-8800. Cuanto más larga era la secuencia de vídeo a comprimir, más lento era el sistema, por lo que se decidió llevar el algoritmo a un computador paralelo [86]. Como la etapa de preprocesamiento consume una pequeña parte del tiempo de codificación, no se incorpora en la versión paralela, centrándose por tanto en las dos etapas centrales del proceso: Estimación de movimiento y Compresión de vídeo.

La máquina que se utilizó fue la POOMA (*Parallel Object Oriented Machine*), un computador paralelo experimental desarrollado en *Philips Research.* Una arquitectura MIMD débilmente acoplada. El prototipo estaba compuesto por 100 nodos. Los programas para esta máquina, se escriben en POOL (*Parallel Object Oriented Language*), un lenguaje de programación diseñado en el proyecto POOMA, en C extendido con sistema operativo para creación de procesos, comunicación y sincronización. El codificador original, se dividió en tareas, asignando una tarea a cada procesador.

Ventajas de utilizar la plataforma paralela: para una aplicación de CD interactivo, por ejemplo, de duración aproximada de 30 minutos, en un Vax se necesitaría un mes, mientras que el POOMA sólo necesita un día para hacer lo mismo.

Actualmente, el amplio campo del procesamiento paralelo está causando un gran impacto en áreas tales como las comunicaciones de datos visuales y de audio. Se han desarrollado muchos tipos de técnicas de procesamiento paralelo que incluyen memoria distribuida y compartida, procesamiento débil y fuertemente acoplado, conmutación de paquetes y paso de mensajes, grano fino y grueso, y distintas topologías.

Por lo tanto, la idea de almacenar y transmitir grandes cantidades de datos digitales audiovisuales, lleva a la necesidad de alto poder computacional y, en consecuencia, al uso del procesamiento paralelo en el campo de reducción de datos.

Una secuencia de vídeo puede verse como una señal tridimensional, dos dimensiones en el dominio espacial y una en el dominio temporal. Así, pueden usarse dos aproximaciones diferentes en la paralelización del algoritmo de codificación, la espacial y la temporal.

Con el paralelismo espacial, cada imagen de una secuencia de vídeo se divide en partes y cada parte se asigna a un elemento de proceso. La secuencia de vídeo se procesa imagen a imagen. El retardo en los resultados es mínimo, pero, evidentemente, este procedimiento de disminuir el tiempo de cómputo aumentando el número de procesadores, tiene un límite, debido a la resolución espacial limitada de una secuencia de vídeo.

Con *paralelismo temporal*, diferentes elementos de proceso, procesan diferentes imágenes de vídeo de una secuencia. No hay límite superior en el número de procesadores, pero hay un retardo en el rendimiento total.

Una de las implementaciones de MPEG-1 más destacables, es la realizada en la Universidad de Berkeley, en California [87]. Donde se han usado diferentes métodos para incrementar la velocidad de ejecución. Los avances más importantes han estado orientados en dos direcciones:

1) Distribuir la tarea de compresión entre varios procesadores, bien sean éstos los componentes de un cluster de estaciones de trabajo, o de un multicomputador.

En esta línea hay que destacar los trabajos de la Universidad de Purdue, que han llevado el codificador paralelo de Berkeley al Paragon de Intel, obteniendo mejoras significativas [88]. El codificador paralelo basado en Split-C, desarrollado inicialmente sobre el CM-5 y pensado para ejecutarse sobre redes de estaciones de trabajo conectadas mediante redes de alta velocidad conmutadas, dentro del proyecto NOW de la Universidad de California, en Berkeley [89].

Por otro lado, la distribución de tareas, se puede realizar, tal y como plantean Yu y Anastassiou, sobre un grupo de estaciones de trabajo SUN, conectadas mediante Ethernet, obteniendo un caudal de ejecución de 4.7 imágenes/s [90]. También hay que destacar los trabajos realizados para codificar datos de audio. Uno de los más recientes es el realizado en la Universidad de Miami en la codificación en paralelo de MPEG-1, utilizando un array de procesadores conectados en malla y con estructura cúbica [91].

2) Utilizar aceleradores hardware específicos de esta aplicación. Un ejemplo de esto, es el desarrollado por Texas Instruments, el Procesador de Vídeo Multimedia (MVP) [92]. Este acelerador comprime secuencias de vídeo con formato CIF en tiempo real. Existiendo también actualmente en el mercado otras implementaciones del mismo tipo.

3.2.3. La codificación en paralelo.

Vamos a describir ahora el prototipo del codificador paralelo de la Universidad de Hong-Kong, como modelo de referencia para nuestros estudios. No se emplea ningún hardware de propósito especial, ni primitivas de programación, sino que es completamente portable, flexible y escalable.

La implementación paralela del MPEG-2 se basa en el paradigma de programación SPMD o *data-parallel* [101]. Este paradigma permite que el software sea portable a través de un amplio rango de arquitecturas, desde sistemas escalables de alto rendimiento, hasta redes de estaciones de trabajo de alta velocidad.

En la idea inicial, el reparto de datos se hacía imagen a imagen, dividiendo cada una de ellas en bloques, y asignando un bloque a cada procesador

Se organiza la topología de procesadores como una rejilla bidimensional. A los procesadores se le asignan las coordenadas x-y, que facilitan la comunicación entre ellos, e identifican los procesadores vecinos. Los datos se mapearán en esa rejilla virtual de procesadores.

Para una computación paralela eficiente, es muy importante el diseño de las estructuras de datos distribuidas. Este diseño debe hacerse desde el punto de vista de balancear lo más posible la carga computacional en los procesadores individuales, y de ganar el máximo paralelismo de las comunicaciones de datos entre procesadores [93].

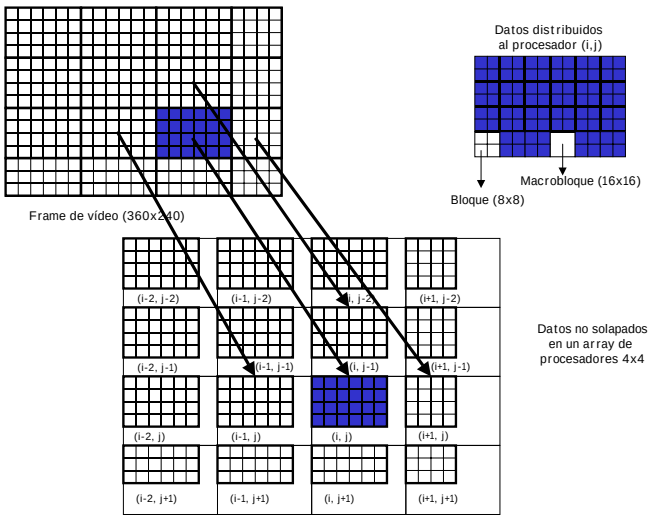


Figura 3.2. Distribución de un frame en la topología de red virtual

La distribución de datos entre los procesadores es bastante simple. Se reparte un frame entero, tan uniformemente como sea posible. Se reparten los datos entre los procesadores tal como son mapeados en la rejilla bidimensional, como se muestra en la figura siguiente:

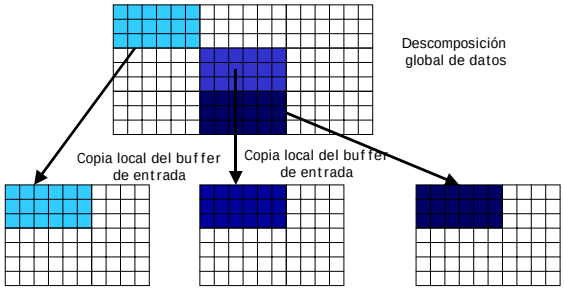


Figura 3.3. Reparto de datos sin solapamiento

Pero, en este caso, es necesaria la comunicación entre los nodos, conforme la ventana de búsqueda se mueve por los bordes (Fig3.4).

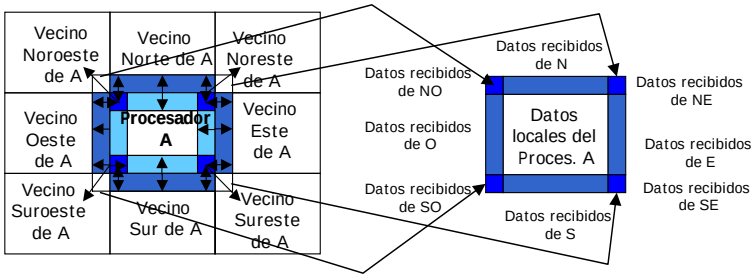


Figura 3.4. Flujos de datos de un procesador a sus procesadores vecinos

Como cada procesador tiene suficiente memoria como para almacenar la ventana de búsqueda completa, es posible eliminar esta enorme cantidad de comunicación, distribuyendo los datos del frame de forma solapada entre los procesadores (Fig 3.5). A cada procesador se le asignan algunos datos redundantes, necesarios para formar el área de búsqueda completa.

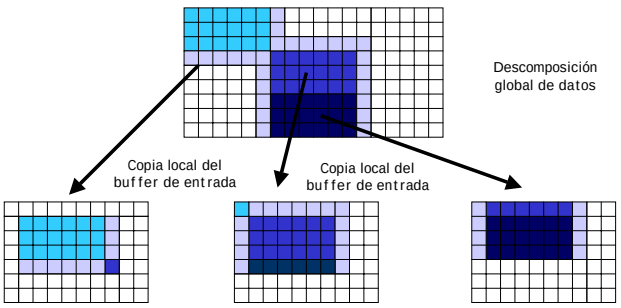


Figura 3.5. Reparto de datos con solapamiento

Los primeros estudios, además de plantear varias estrategias de optimización del código, que fue donde nos incorporamos nosotros, iban encaminados a realizar distintas

pruebas con algoritmos de estimación de movimiento diferentes y a utilizar otras librerías de paso de mensajes tales como PVM y MPI (*Message-Passing Interfaz*).

El objetivo al desarrollar este codificador fue conseguir un algoritmo de compresión de vídeo independiente de la arquitectura, sin comprometer la velocidad de procesamiento. La idea fue, no emplear ningún hardware de propósito especial, ni primitivas de programación concretas, para hacerlo completamente portable, flexible y escalable.

Bajo el paradigma SPMD los datos se parten en pequeñas piezas que se asignan a procesadores diferentes. Un único programa se escribe en todos los procesadores, que lo ejecutarán asíncronamente para sus datos locales. Con el objetivo de alcanzar velocidad en la estimación de movimiento, se realizó búsqueda logarítmica 2D, y se registró el correspondiente rendimiento. Para medir la calidad del vídeo, se utilizó la PSNR (*Peak signal to Noise Ratio*). Valores grandes de la misma indican mejor calidad.

3.2.4. Análisis de diferentes técnicas de distribución de datos

Partiendo de estas ideas básicas del codificador paralelo, hemos implementado distintos métodos de reparto de datos diferentes. Haremos *paralelismo espacial* dividiendo cada una de las imágenes de la secuencia entre los distintos procesadores y *paralelismo temporal* dividiendo la secuencia completa entre los distintos procesadores.

- ***Paralelismo espacial****: división del frame*

Los datos deben distribuirse entre los procesadores tal que cada procesador codifique uno o más macrobloques.

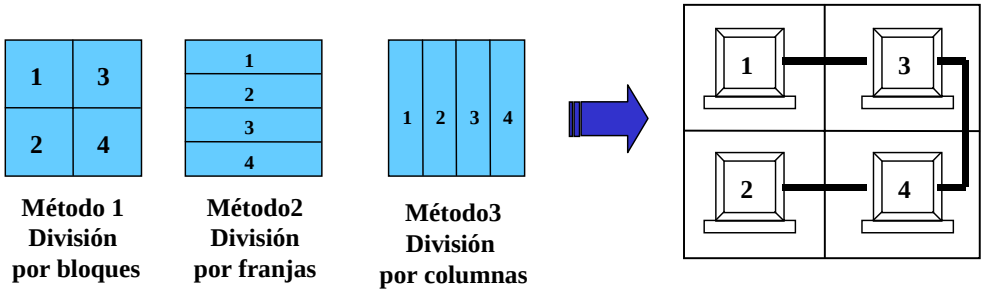


Figura 3.6. Diferentes métodos de división del frame

Veamos, brevemente, los fundamentos matemáticos de las distintas divisiones. Sea el tamaño de frame igual a M×N (en píxeles) donde:

M = h × tamaño_macrobloque

N = v × tamaño_macrobloque

Con tamaño_macrobloque = 16 y h, v como el número de macrobloques y slices, respectivamente. Si el tamaño de la rejilla de procesadores 2D es m×n, el conjunto de procesadores que tendremos será ((Pij)), i= 1, 2, …k, …m, j= 1, 2, …, l, …n, y, además los siguientes valores a tener en cuenta:

- Píxeles no asignados en la dimensión m: α = MOD(M, (m×16))
- Píxeles asignados a cada procesador en la dimensión m: β= (M-α)/m
- Píxeles no asignados en la dimensión n: γ = MOD(N, (n×6))
- Píxeles asignados a cada procesador en la dimensión n: δ = (N-γ)/n
- Se deduce que: α = s×16, γ = t×16, siendo **s** y **t** enteros positivos, los macrobloques no asignados en cada dimensión.
- Los procesadores se numeran en orden consecutivo por columnas, denotaremos ese número por **r**.

Método 1: *División en bloques*

Para este método el tamaño local del frame de datos en el procesador Pij se determina de la siguiente manera: en la actual implementación, el tamaño local de datos es, el tamaño calculado anteriormente, más algunos datos solapados (necesarios para formar la ventana de búsqueda). Esto se hace de forma similar en los siguientes métodos.

$$\text{Cantidad de datos para Pij} = \begin{cases} (\beta+16) \times (\delta+16) & \text{si } i \le k, j \le l \\ \beta \times (\delta+16) & \text{si } i > k, j \le l \\ (\beta+16) \times \delta & \text{si } i \le k, j > l \\ \beta \times \delta & \text{si } i > k, j > l \end{cases} \quad \begin{matrix} k \le s \\ l \le t \end{matrix}$$

Método 2: *División horizontal del frame*

Para este método tendremos: α=0, β= M, γ = MOD (N, (m×n×16)), δ = (N-γ) / (m×n), γ = t×16, ya que el tamaño local de datos para todos los procesadores será M × una fracción de N, M × (N/m×n). N se dividirá ahora entre todos los procesadores y no sólo entre los procesadores situados en la dimensión n de la rejilla de procesadores. Así, el tamaño local del frame en el procesador Pij se determina como sigue:

$$\text{Cantidad de datos para Pij} = \begin{cases} \beta \times (\delta + 16) & \text{si } r < t \\ \beta \times \delta & \text{si } r >= t \end{cases}$$

Método 3: *División vertical del frame*

Para este método los nuevos valores son:
α = MOD(M, (m×n×16)), β = (M-α) / (n×n), γ = 0, δ =N, α = s×16. Ahora M se dividirá entre todos los procesadores. Así, el tamaño local de datos en el procesador Pij se puede calcular como:

$$\text{Cantidad de datos para Pij} = \begin{cases} (\beta + 16) \times \delta & \text{si } r < s \\ \beta \times \delta & \text{si } r >= s \end{cases}$$

- ***Paralelismo temporal****: División de la secuencia*

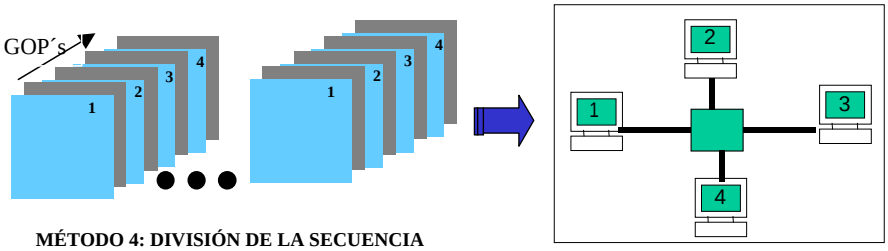
Veamos otro método para distribuir el vídeo entre los procesadores. Consideremos los siguientes valores: *nframes* es el número de frames en la secuencia, *N*, el número de frames en un GOP, *ngops* será nframes/N, *m×n* el tamaño de la rejilla de procesadores 2D y:

- α = MOD (ngops, (mHn)), número de GOPs no asignados
- β = (ngops - α) / (mxn), número de GOPs asignados a los procesadores
- Los GOPs asignados al procesador Pij, con rango(pij) = r, serán calculados como:

$$ng = r + (k \times m \times n)$$

Con las siguientes condiciones:
si $\alpha = 0$ entonces k = 0 … β-1
si $\alpha \neq 0$ entonces
si r < α entonces k = 0 …β
sino si r ≥ α then k=0 … β-1.

Después de ésto, los cálculos para obtener los frames asignados a cada procesador, *nf*, son los siguientes, con N = número de frames en un GOP. Si ng = x entonces *nf = x, …, (x+N-1)*



MÉTODO 4: DIVISIÓN DE LA SECUENCIA

Figura 3.7.División de datos con paralelismo temporal

Con estos cuatro métodos se hicieron los primeros experimentos y se fijaron los puntos de mayor interés para su investigación y mejora. En el siguiente capítulo veremos brevemente las pruebas realizadas con estos cuatro métodos, las principales conclusiones y la elección justificada del método elegido para el posterior banco de pruebas

3.3. LIBRERÍAS PARA EL PASO DE MENSAJES

En el capítulo 1 aparece una breve descripción del paradigma del paso de mensajes, así como de los principales representantes del mismo. En este capítulo nos vamos a centrar en las librerías que nosotros hemos utilizado para tal fin. Como parte componente de los primeros experimentos, no tenemos, ni queremos olvidarnos de las pruebas realizadas para comparar, evaluar y decidir que librería utilizar. Al final la seleccionada fue MPICH, no por capricho sino por hechos. En el capítulo 4 de resultados también mencionaremos este tema, los trabajos que se hicieron y las conclusiones a las que llegamos. Pero, como en la exposición de estas pruebas se van a mencionar otras interfaces distintas a MPICH, es ahora el momento de describirlas brevemente.

- **MPICH** y **LAM** ya se han descrito en el capítulo 1, y se puede encontrar una extensa documentación sobre ellas en sus respectivas páginas web [55, 44] MPICH es una implementación portable y disponible libremente que soporta todas las especificaciones del estándar MPI para una gran variedad de entornos de computación paralela, por lo que es bastante general.

- **MPI-BIP** es una implementación de MPI sobre Myrinet que está construido a partir de MPICH y exclusivamente para redes Myrinet. Usa el nivel de comunicación BIP y los mensajes BIP (*Basic Interface for Parallelism*) y no es más que un pequeño API implementado para la red Myrinet. Su importancia radica en su capacidad para conseguir el máximo rendimiento posible del hardware. El principio básico de los mensajes BIP es implementar todas las comunicaciones en una librería a nivel de usuario con acceso directo al hardware de la red sin llamadas al sistema. MPI-BIP no utiliza TCP, explota las ventajas de BIP, MPI y la red Myrinet, consiguiendo máximo rendimiento para las aplicaciones que se ejecuten sobre esta implementación [94]. A pesar, de poder ofrecer “mejores” resultados, como se puede ver en [142], los problemas que surgen al aumentar el número de procesadores son insufribles. Toda la experimentación realizada en el proyecto BIP parte de la conexión entre dos máquinas, pero no se enfrentan al problema del paralelsimo y de utilizar un elevado número de procesadores, que a través de BIP, se comuniquen. Los continuos problemas en la sincronización de los procesadores nos han llevado a dejar de utilizar esta interfaz hasta que aparezcan versiones optimizadas. Por otra parte MPICH tiene un carácter más general y su principal ventaja es ser válido para numerosos sistemas paralelos.

- **GM**: es un sistema de comunicación basado en mensajes para Myrinet [95]. Como muchos otros sistemas de mensajes, sus objetivos incluyen baja sobrecarga de CPU, portabilidad, baja latencia y alto ancho de banda. Además, GM incluye algunas características propias, como el dar soporte a cluster de

hasta diez mil nodos, proporcionar dos niveles de prioridad de mensajes, mapeos automáticos de redes Myrinet, entrega de mensajes fiable y ordenada, etc. Nosotros hemos usado gm versión 1.4, que en secciones siguientes describiremos de forma particular.

3.4. FORMAS DE COMPARTIR LA INFORMACIÓN

3.4.1. Introducción a los sistemas de ficheros

Un sistema de ficheros es una estructura de directorios que se usa para organizar y almacenar ficheros. El término *sistema de ficheros* se usa de diferentes formas:

- Para describir un tipo particular de sistema de ficheros: basado en disco, basado en red, o virtual.
- Para describir un árbol de ficheros completo, desde el directorio raíz hacia abajo.
- Para describir la estructura de datos de un trozo del disco o de otro medio de almacenamiento.
- Para describir una porción de una estructura de árbol de ficheros que se adjunta a un punto de montaje en el árbol de ficheros principal, para que sea accesible.

La administración del sistema de ficheros es una de las tareas más importantes de administración del sistema. Podemos distinguir tres tipos principales:

1. *Basados en disco*: se almacenan en medios físicos tales como discos duros, CD-ROMs, y disquetes. Hay varios formatos disponibles como el *UFS* (*Unix File System*), *HSFS* (*High Sierra and ISO 9660 File System)* que usa CD-ROM y es de solo lectura, o el *PCFS* (*PC File System*) que permite accesos de lectura/escritura a datos y programas en discos formateados DOS para computadoras personales basadas en DOS. Cada uno de los tipos anteriores está asociado con un dispositivo, UFS con disco duro, HSFS con CD-ROM, y PCFS con disquete.

2. *Basados en red*: son aquellos a los que se accede a través de la red, como *NFS* (*Network File S*ystem) y *AFS* (*Andrew File System*), que trataremos en profundidad en las siguientes secciones.

3. *Virtuales*: son sistemas de ficheros basados en memoria que proporcionan acceso a información y servicios del kernel. Muchos sistemas virtuales no usan espacio de disco. Sin embargo, algunos utilizan el disco para contener una cache, o el espacio swap del disco. Algunos de ellos son *CacheFS* (*Cache File System*), el *TMPS* (*Temporary File System)*, el *LOFS* (*Loopback File System*), el *PROCFS* (*Process File System*) Otros adicionales que no requieren administración, son: FIFOFS (First-in-firs out file System), FDFS (File Descriptors File System), NAMEFS, SPECFS (Special File System) y SWAPFS [96].

Entremos un poco más a fondo en los sistemas de ficheros UNIX. La estructura jerárquica de directorios y archivos es una visión lógica del sistema de archivos. Internamente, UNIX organiza el disco y mantiene un control de los archivos de una manera diferente. El sistema de archivos UNIX asocia cada nombre de archivo con un número llamado número de inodo, e identifica cada archivo con su número de inodo. UNIX guarda todos esos números inodo en una lista, denominada lista de inodos [97].

En la mayoría de los casos, los sistemas de ficheros que usan el sistema Linux no cambian con demasiada frecuencia. Por ello, se puede especificar con facilidad una lista de sistema de ficheros que Linux monta cuando arranca y desmonta cuando lo detenemos. Esos sistemas son los listados en el archivo /etc/fstab [98].

Los sistemas de ficheros se pueden adjuntar a la jerarquía de directorios disponibles en el sistema. A este proceso se le llama *montaje.* Un sistema de ficheros montado se adjunta al árbol de directorios del sistema en un *punto de montaje* especificado (un directorio concreto), y ya se hace disponible al sistema. El sistema de ficheros raíz (/) siempre se monta. Cualquier otro puede conectarse o desconectarse del sistema de ficheros raíz. Hay que determinar los siguientes asuntos:

- Si el sistema de ficheros se va a incluir en /etc/fstab (tabla de sistemas de archivos), para que sea montado cada vez que se levanta el sistema.
- Si el sistema de ficheros se puede montar de forma apropiada utilizando un automontador.
- Si el sistema de ficheros sólo se utilizará temporalmente
- Si el sistema de ficheros se puede montar desde la línea de comandos, usando el comando mount.

Además, para montar un sistema de ficheros necesitamos:

- Ser super usuario
- Un punto de montaje en el sistema local
- El nombre del recurso del sistema de ficheros que se va a montar (por ejemplo, /usr).

Por ejemplo, en NFS, veremos como los clientes montarán los directorios exportados por el servidor. Por lo tanto, para montar un sistema de ficheros NFS, éste debe estar disponible desde el servidor. Desde la línea de comandos ejecutaríamos una orden como la siguiente:

```
# mount –t nfs [-o opciones de montaje] servidor:/directorio punto-de-montaje
```

La operación contraria se denomina *desmontaje*, la cual elimina el sistema de ficheros del punto de montaje, y borra la entrada del fichero /etc/fstab. Se utilizará el comando umount.

Linux mantiene un conjunto de estructuras que sirven para unificar, de cara al usuario, todos y cada uno de los posibles tipos de estructuras de almacenamiento de datos. Esta capa intermedia se denomina sistema de ficheros virtual, VFS (*Virtual File System).* El VFS esconde las peculiaridades de cada sistema de ficheros y unifica el manejo a través de un API común. El VFS proporciona métodos genéricos para las diversas operaciones que se pueden realizar con inodos y ficheros, funciones que luego son personalizadas para cada sistema en particular

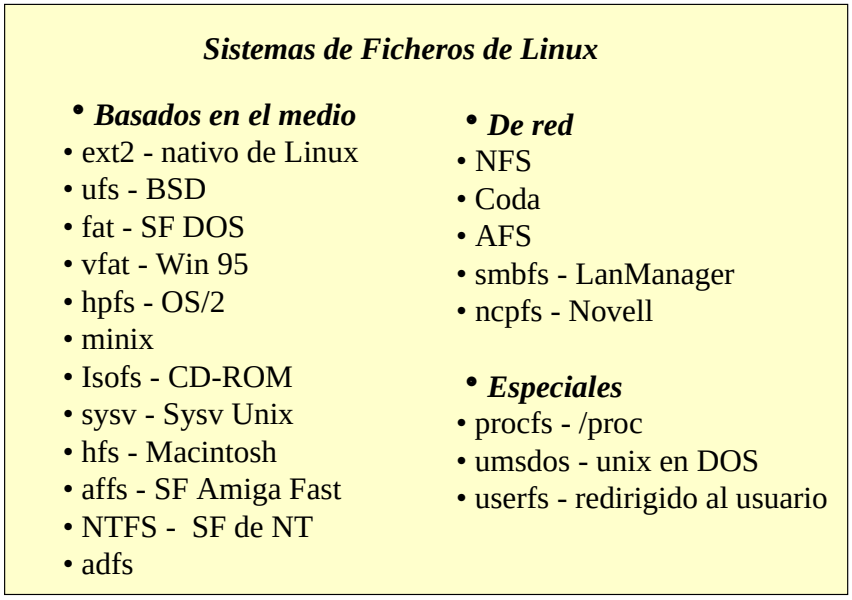


Figura 3.8. Organización de la lista de sistemas de ficheros

A nivel de implementación, los diversos sistemas de ficheros se agrupan como una lista encadenada de estructuras que definen cada sistema de ficheros. La figura 3.2 ilustra esta lista. La esencia de los sistemas de ficheros es que se pueden montar dentro del árbol de directorios [99].

3.4.2. Llega el momento de compartir

Desde los años setenta, la habilidad de conectar computadoras en una red, ha revolucionado la industria informática. El incremento de conectividad en la red ha avivado el deseo de compartir ficheros entre distintas máquinas. Los primeros esfuerzos en esta dirección estaban restringidos a copiar ficheros completos de una máquina a otra, tales como el programa de copia de Unix a Unix (*uucp*) y el protocolo de transferencia de ficheros (*ftp*). Tales soluciones, sin embargo, no satisfacían la visión de poder acceder a ficheros en máquinas remotas como si estuvieran en los discos locales.

A mediados de los ochenta se empezó a ver la emergencia de algunos sistemas de ficheros distribuidos que permitían acceso transparente a ficheros remotos sobre una red. Éstos incluían el Sistema de Ficheros en Red (*NFS*) de Sun Microsystems, el Sistema para compartir Ficheros Remotos (*RFS*) de AT&T, y el Sistema de Ficheros de Andrew (*AFS*) de la Universidad Carnegie-Mellon. Los tres eran claramente diferentes en sus objetivos de diseño, arquitectura y semántica, aunque intentaran resolver los

mismos problemas fundamentales. Actualmente, RFS está disponible en muchos sistemas basados en *System V*. NFS ha cobrado una amplia aceptación y está disponible en numerosos sistemas Unix y no Unix. El desarrollo de AFS ha pasado a Transarc Corporation, donde ha evolucionado en el Sistema de Ficheros Distribuido (*DFS*) componente del entorno de Computación Distribuida (DCE) de la Fundación de Software Abierto [100].

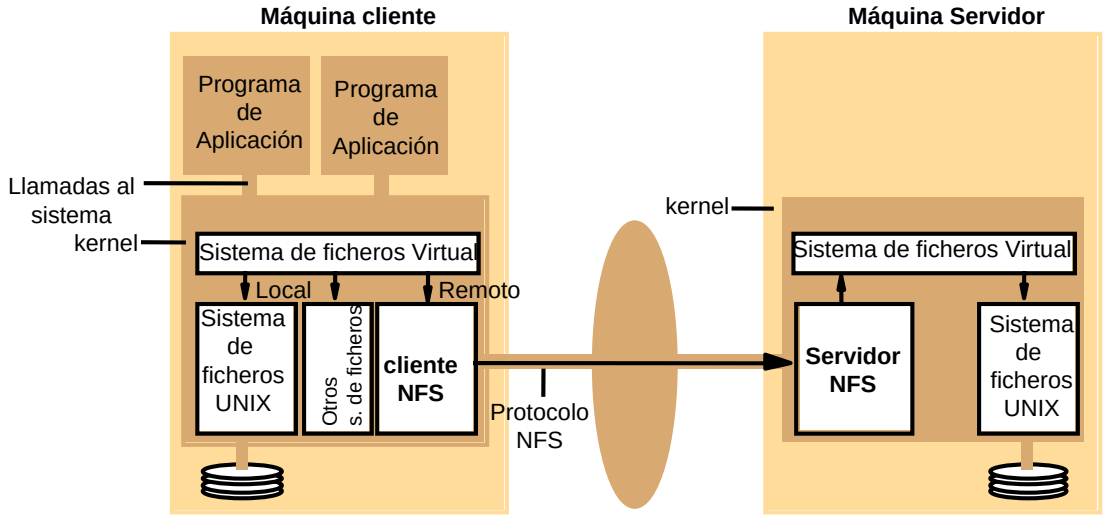


Figura 3.9. Protocolo de transferencia de ficheros y Protocolo de acceso directo al fichero

Protocolos de acceso a ficheros como NFS se diseñaron para eliminar la necesidad de transferir el fichero (ver figura 3.3). El fichero permanece donde está, en el servidor, y se manipula allí mismo. Las ventajas de los protocolos de acceso a ficheros frente a los protocolos de transferencia de ficheros son las siguientes:

- El usuario tomas solo lo que necesita
- Los ficheros remotos parece que fueran locales
- No hay datos que no estén actualizados
- Los clientes pueden no tener disco
- No hay esperas
- Hay cerrojo de ficheros

Aunque los detalles de estos sistemas varían considerablemente, todos comparten una propiedad fundamental: *los datos en un fichero no se traen en masa, el sitio remoto participa potencialmente en cada operación de lectura y escritura individual.* Algunos sistemas emplean almacenamiento en buffers y lecturas adelantadas para mejorar prestaciones, pero el sitio remoto está aún envuelto en cada operación de lectura/escritura. Se llama a esta propiedad *"Remote Open"* [101].

Un sistema de ficheros centralizado, convencional, permite a múltiples usuarios, en un sistema único, compartir el acceso a ficheros almacenados localmente en la máquina. Un sistema de ficheros *distribuido* o *basado en red* extiende esta posibilidad de compartir ficheros a usuarios en diferentes máquinas, interconectados por una red de comunicación. Los sistemas de ficheros distribuidos se implementan usando un modelo cliente-servidor. El cliente es una máquina que accede a un fichero, mientras que el servidor almacena los ficheros y permite a los clientes que accedan a ellos. Algunos

sistemas pueden requerir que los clientes y servidores sean máquinas distintas, mientras que otros pueden permitir que una única máquina actúe tanto como cliente como servidor. Un sistema de ficheros distribuido, en cambio, es una capa software que maneja la comunicación entre sistemas operativos convencionales y sistemas de ficheros. Está integrado con el sistema operativo de las máquinas y proporciona un servicio de acceso de fichero distribuido a sistemas con kernels centralizados.

Hay algunas propiedades importantes de los sistemas de ficheros distribuidos. Cada sistema de ficheros puede tener alguna o todas estas propiedades. Esto nos da una base para evaluar y comparar distintas arquitecturas.

- *Transparencia de red*: los clientes deben ser capaces de acceder a ficheros remotos usando las mismas operaciones que se aplican a los ficheros locales.
- *Transparencia de posición:* el nombre de un fichero no debe revelar su posición en la red
- *Independencia de posición:* el nombre del fichero no debe cambiar cuando cambia su posición física
- *Movilidad de usuarios*: los usuarios deben ser capaces de acceder a los ficheros compartidos desde cualquier nodo en la red.
- *Tolerancia a fallos*: El sistema debe continuar funcionando después de un fallo de un componente (un servidor o un segmento de red). Puede, sin embargo, que se degrade su rendimiento o que parte del sistema de ficheros no esté disponible.
- *Escalabilidad*: el sistema debe escalar bien, conforme su carga se incremente. También debería ser posible el crecimiento del sistema de forma incremental, añadiendo componentes.
- *Movilidad de ficheros*: debería ser posible mover ficheros de una posición física a otra en un sistema en ejecución.

También hay algunos temas importantes que considerar en el diseño de un sistema de ficheros distribuido. Éstos tienen que ver con la funcionalidad, semántica y rendimiento. Compararemos diferentes sistemas de ficheros de acuerdo a cómo traten estos temas:

- *Espacio de nombres:* algunos sistemas de ficheros distribuidos proporcionan un espacio de nombres uniforme, tal que cada cliente usa el mismo camino para acceder a un fichero dado. Otros permiten a cada cliente personalizar su espacio de nombres montando sub-árboles compartidos en directorios arbitrarios en la jerarquía de ficheros.
- O*peraciones sin estado o de estado completo:* un servidor de *estado completo* es aquel que retiene información sobre las operaciones de cliente entre peticiones, y usa esta información de estado para servir las peticiones siguientes correctamente. En un sistema *sin estado*, cada petición es auto contenida, y el servidor mantiene estados no persistentes sobre los clientes. Los servidores de estado completo son más rápidos, ya que el servidor puede tomar ventaja de su conocimiento del estado del cliente para eliminar un montón de tráfico en la red. Sin embargo, tienen mecanismos de recuperación y consistencia complejos. Los servidores sin estado son más simples de diseñar e implementar, pero no producen tan buenas prestaciones. Por ejemplo, UDP es un protocolo sin estado que puede perder paquetes o

distribuirlos desordenados. TCP es un protocolo de estado completo que garantiza que los paquetes van a llegar y, además, que se distribuyen en orden.

- *Semántica de la operación de compartir*: los sistemas de ficheros distribuidos deben definir la semántica que aplican cuando múltiples clientes acceden a un fichero de forma concurrente. La semántica de Unix requiere que los cambios hechos por un cliente sean visibles a los demás clientes.
- *Métodos de acceso remoto:* un modelo cliente-servidor puro usa el método de servicio remoto de acceso a ficheros, en el que cada acción se inicia por el cliente, y el servidor es simplemente un agente que hace lo que el cliente ofrece. En muchos sistemas distribuidos, en particular los de estado completo, el servidor juega un papel mucho más activo. No sólo sirve las peticiones de los clientes, sino que participa en mecanismos de coherencia de cache [100].

Los beneficios de la utilización de servicios distribuidos están más que demostrados. El dilema no es si se implementan o no, sino qué solución elegir. La elección de un servicio de ficheros distribuido adecuado es uno de las decisiones importantes que afectan al acceso transparente y fiable de la información. Los distintos productos varían significativamente en su habilidad para satisfacer requisitos clave como escalabilidad, acceso rápido y transparente, protección de la información, facilidad de administración y amplio soporte por una variedad de vendedores.

EL sistema de ficheros distribuido NFS es un componente de la solución de computación distribuida ONC+ de SunSoft. Es una solución de empresa que proporciona seguridad, alto rendimiento y acceso transparente a la información en redes heterogéneas de todo el mundo. Publicado originalmente en 1985, es una solución robusta, completamente caracterizada que estaba instalada en 1995 en ocho millones de sistemas. Su crecimiento se dio tanto en entornos de PCs como de estaciones de trabajo/servidores. El comité de SunSoft para el desarrollo de la tecnología NFS que reúne las necesidades de la comunidad que comparte ficheros distribuidos, asegura, que la curva de crecimiento ha continuado creciendo vertiginosamente.

Desde su primera introducción, NFS ha continuado desarrollándose para reunir los requisitos que implica el compartir ficheros distribuidos. Algunas de las características más importantes que proporciona NFS son:

- Escalabilidad para soportar redes pequeñas y grandes
- Acceso transparente y continuo a ficheros globales gracias a nueva facilidad de montaje automático.
- Una nueva revisión del protocolo NFS, NFS versión 3, que incrementa la escalabilidad y el rendimiento.
- Cache de disco local y rendimiento mejorado, altamente competitivo, que habilita el acceso rápido a la información.
- Una arquitectura de seguridad flexible y extensible, que asegura la protección de ficheros con intrusos no autorizados [102].

El servicio NFS habilita a computadoras de distintas arquitecturas que ejecutan distintos sistemas operativos, a compartir sistemas de ficheros a través de la red. El entorno NFS se puede implementar en diferentes sistemas operativos debido a que define un modelo abstracto de sistema de ficheros, en lugar de una especificación de arquitectura. Cada sistema operativo aplica el modelo NFS a su semántica de sistema de ficheros. Por tanto, los beneficios de los servicios NFS son los siguientes:

- Permite que múltiples computadoras usen los mismos ficheros, así, todos en la red pueden acceder a los mismos datos
- Reduce los costos de almacenamiento al tener computadoras que comparten aplicaciones, en lugar de necesitar espacio de disco local para cada aplicación de usuario.
- Proporciona consistencia y fiabilidad ya que todos los usuarios pueden leer el mismo conjunto de ficheros.
- Hace el montaje de los sistemas de ficheros transparentes a los usuarios
- Hace el acceso a ficheros remotos transparente al usuario
- Soportan entornos heterogéneos
- Reduce la sobrecarga de la administración del sistema

NFS se diseñó para dar a los usuarios altas prestaciones y acceso transparente, para servir sistemas de ficheros en redes globales. Algunos de los principios de diseño más importantes son el a*cceso transparente,* la *portabilidad*, la *rápida recuperación de fallos,* la *independencia del protocolo de red*, el *rendimiento,* y la s*eguridad.*

Estas características se implementan en el marco de trabajo cliente/servidor, que reduce los costos posibilitando compartir recursos heterogéneos. Además, NFS consiste de un programa cliente, un programa servidor y un protocolo para la comunicación entre los dos sobre la red. Los servidores hacen que sus ficheros se puedan compartir a través de un proceso de *exportación*. Los clientes acceden a los sistemas de ficheros, añadiéndolos a su árbol de ficheros local mediante el proceso de *montaje.* El protocolo NFS proporciona el medio de comunicación entre los procesos servidor y cliente sobre la red.

Para exportar un sistema de ficheros, el administrador del sistema debe editar un fichero de configuración y especificar tres cosas:

1. El camino completo del directorio que se va a exportar
2. Las máquinas cliente que accederán al directorio exportado
3. Cualquier restricción de acceso

Al arrancar el sistema, se lee la información de este fichero de configuración y se informa al kernel del sistema operativo de los servidores de los permisos aplicables a cada jerarquía de ficheros exportada. Una vez completado este proceso, el servidor está preparado para aceptar las peticiones de los clientes [102].

Ejemplo 1:

En nuestro cluster de ocho máquinas conectadas mediante conmutadores myrinet, las máquinas se nombran desde myrinet1 hasta myrinet8, siendo la myrinet1 la que actúa de servidor, y el resto las máquinas cliente.

En el fichero */etc/exports* de la myrinet1 se especifica el sistema de ficheros a compartir, que será **/home/**, por ejemplo, seguido del nombre de las máquinas que pueden utilizar este sistema de ficheros, es decir, desde myrinet2 hasta myrinet8. Cuando se levanta el servidor, chequea la existencia de /etc/exports y ejecuta *exportfs* para hacer su sistema de ficheros disponible a los clientes.

Por otro lado, en las máquinas cliente, en el fichero */etc/fstab* aparecerá el sistema que se quiere montar, así tendremos lo siguiente:

myrinet1:/home/ /home/ nfs rsize=1024, wsize01024, hard, intr, 0 0

Lo que indica que se va a montar el directorio /home/ existente en myrinet1, en /home/ de cada una de las máquinas y que va a ser un montaje tipo nfs con distintos parámetros que más adelante especificaremos.

Así, en lugar de duplicar directorios comunes, tales como /home/, en cada sistema, NFS proporciona una única copia del directorio que es compartido por todos los sistemas en la red. Para el usuario, NFS significa no tener que introducir su *login* en otro sistema para acceder a un fichero, no hay necesidad de usar rcp, ni de mover ficheros de un sitio a otro.

Los términos *cliente* y *servidor* se usan para describir los papeles que juega una computadora cuando comparte un sistema de ficheros. Si un sistema de ficheros reside en un disco de una máquina y esa máquina hace el sistema de ficheros disponible a otras máquinas de la red, entonces esa máquina actúa como servidor. Las computadoras que acceden a ese sistema de ficheros se llamarán clientes. El servicio de NFS hace posible que cualquier máquina acceda a cualquier sistema de ficheros de otra máquina, y, al mismo tiempo, que proporcione acceso a su propio sistema de ficheros. Una computadora puede ser cliente, servidor, o ambas cosas, en un momento dado, en la red.

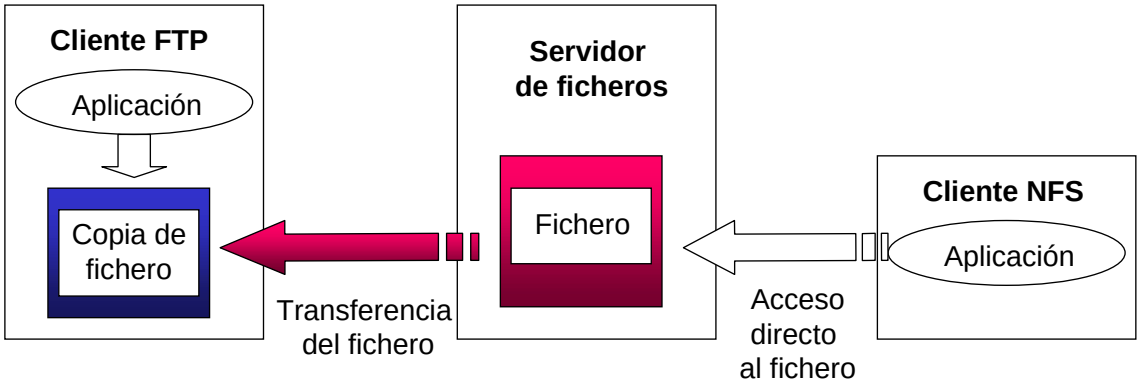


Figura 3.10. Arquitectura de NFS

Un servidor puede proporcionar ficheros a clientes sin disco local. Los clientes acceden a los ficheros del servidor montando los sistemas de ficheros compartidos del servidor. Cuando un cliente monta un sistema de ficheros remoto, no hace una copia del sistema de ficheros; en su lugar, el proceso de montaje usa una serie de llamadas de

procedimiento remoto (RPC, *Remote Procedure Call*) que habilitan al cliente el acceso al sistema de ficheros del disco del servidor, de forma transparente [103].

Exportar subdirectorios es similar a crear vistas en una base de datos relacional. Se eligen las porciones de la base de datos que un usuario necesita ver, ocultando la información que no le interesa o la información más delicada [104].

3.4.3.Componentes de NFS

NFS se construye sobre la base de los protocolos *XDR* (*External Data Representation)* y *ONC RPC* (*Open Network Computing Remote Procedure Call).* NFS se define sobre RPC que usa XDR para codificar los datos que se van a transportar por la red, como muestra la figura 3.5. Además NFS no podría implementar un sistema de ficheros distribuido completo sin dos protocolos muy importantes que lo acompañan, como son el protocolo *MOUNT* y el protocolo *NLM* (*Network Lock Manager*).

Ⓟ XDR (External Data Representation)

Los datos NFS en los mensajes RPC se deben representar en un formato que puedan ser entendidos tanto por el emisor como por el receptor. La necesidad de esa representación de datos común es lo que viene a solucionar XDR, que se ocupa de la representación de valores primitivos como enteros y cadenas, tipos estructurados como arrays, listas enlazadas y uniones. Una descripción detallada de este lenguaje y la notación para describir los datos codificados se puede encontrar en [105, 106]

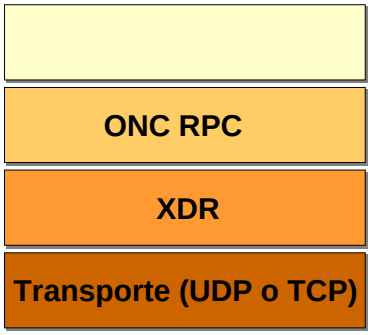


Figura 3.11. Capas del protocolo NFS

Como NFS utiliza la Representación de Datos eXterna (XDR) de Sun, es independiente del lenguaje y de la máquina. XDR define el tamaño, orden de los bytes, y alineación de los tipos de datos básicos como los enteros, arrays, uniones, etc. Así se pueden intercambiar datos entre sistemas con diferentes convenciones para el orden de los bytes [102].

Ⓟ RPC (Remote procedure Call)

El protocolo NFS es independiente del tipo de sistema operativo, arquitectura de red y protocolos de transporte. Esta independencia se debe, en parte, al protocolo RPC [107, 108]. Una llamada a procedimiento remoto es un mensaje que se envía a un servidor en espera de un mensaje de respuesta. El mensaje de llamada identifica un programa en el servidor e invoca un procedimiento en el programa. Los datos

codificados en el mensaje son los parámetros para el procedimiento. El mensaje de respuesta puede ser tanto un código de error o datos de respuesta. Por ejemplo, cuando un cliente NFS quiere leer un fichero, envía una llamada RPC al servidor NFS para invocar al procedimiento READ y le proporciona tres parámetros: el manejador de fichero que identifica al fichero, un *offset* en el fichero y el número de bytes que se van a leer. El mensaje de respuesta contiene los datos del fichero pedido.

En una llamada a un procedimiento local, hay un hilo lógico de ejecución; el control se transfiere al código del procedimiento y después vuelve al llamador cuando la ejecución del procedimiento se termina. Una llamada de un procedimiento remoto no es diferente; el programa que llama espera hasta que el procedimiento remoto se completa y devuelve su respuesta. No es un requisito que el programa que llama no haga nada mientras espera que acabe el procedimiento remoto (podría hacer otro trabajo mientras espera). La habilidad para trabajar mientras espera suele expresarse como alguna forma de multihilo en el programa cliente, donde los hilos hacen las llamadas RPC y se bloquean hasta que la llamada remota se complete. (Como veremos en el capítulo 5, del diseño de las mejoras planteadas, hemos hecho uso del multihilo para acelerar el proceso).

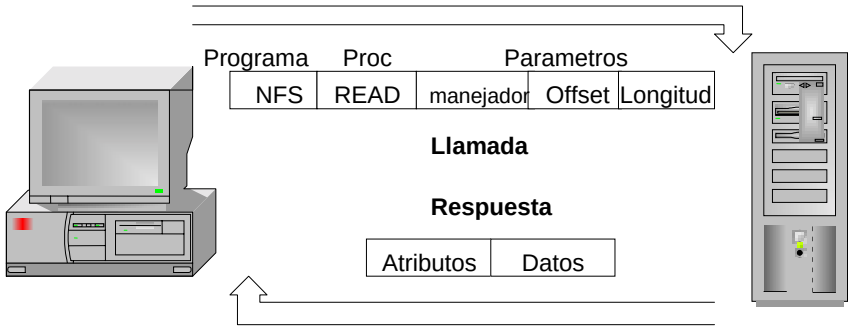


Figura 3.12. Estructura de una llamada a procedimiento remoto

Profundizando un poco más hay que decir que un mensaje RPC comprende una cabecera de llamada o de respuesta seguida por los parámetros o por los resultados. La cabecera de la llamada contiene un identificador de transacción único seguido por el número de programa y por el número de procedimiento que va a ser invocado en el programa. La cabecera de la llamada también incluye una credencial y un verificador que se puede utilizar para identificar la llamada. La cabecera de respuesta contiene un identificador de transacción que debe emparejarse con el de la llamada, seguido por el verificador del servidor que puede ser usado para autentificar el servidor al cliente. La cabecera de la respuesta puede contener, también, códigos de error de los servicios RPC o del programa.

El servidor, por lo general, espera que le llegue un mensaje de llamada, decodifica los argumentos, los envía al procedimiento apropiado, codifica la respuesta y espera que le llegue el siguiente mensaje de llamada.

Aunque las llamadas de procedimiento remoto son similares a las llamadas de procedimiento local, hay algunas diferencias importantes:

- La gestión de errores. Cuando se usan RPCs hay que encargarse de los fallos en el servidor remoto o en la red
- El rendimiento: los procedimientos remotos trabajan, por lo general, uno o más órdenes de magnitud más lentos que lo llamadas locales.
- Autentificación: es necesaria ya que las llamadas pueden viajar por redes no seguras.

RPC es independiente de la capa de transporte, pero no intenta ocultar las limitaciones que tenga esta capa de transporte. Si por debajo de RPC se tiene UDP, como UDP no garantiza la distribución de mensajes, RPC usa la política de retransmisiones de los *time-outs*, que luego contaremos al hablar de los parámetros NFS. En cambio, si por debajo de RPC se tiene TCP, como éste sí que proporciona su propia política de retransmisiones, pues RPC la utilizará en lugar de la suya propia. Aunque, en la práctica, también puede haber retransmisiones por parte de RPC sobre TCP debido a que puede haber cortes o problemas en la conexión TCP. Por esta razón es preferible que los procedimientos remotos sean *idempotentes*, es decir, que si un procedimiento se repite con los mismos argumentos, siempre genere los mismos resultados. Pero no es siempre posible que los procedimientos sean idempotentes, por lo que el servidor para mitigar el efecto de la retransmisión de las peticiones no idempotentes mantiene una cache de la peticiones recientes, para evitar acciones que tengan efectos no deseados. Cuando un cliente hace una petición RPC, establece un periodo de tiempo durante el cual el servidor debe atenderle. Si el servidor no recibe la petición, seguramente porque esté sobre cargado para completar la petición en ese periodo de tiempo, el cliente retransmite la petición.

Cada protocolo RPC está identificado por su número de programa y por su versión. Cuando un cliente necesita enviar una petición al servidor que soporta RPC, necesita saber la dirección de red del servidor (su dirección IP usualmente) así como el punto final de transporte, lo que se identifica como número de puerto para UDP y TCP. Si un servicio RPC tiene un número de puerto bien conocido, entonces el cliente puede enviar sus peticiones a ese puerto. Por ejemplo, los servidores NFS tien por defecto el puerto 2049, así un cliente envía automáticamente las peticiones a este puerto. El servicio de *portmapper* permite a un servicio RPC escuchar peticiones en cualquier puerto libre. Los clientes usan el portmapper para descubrir el puerto en el que el servicio RPC está escuchando. Un servicio RPC registra su puerto TCP o UDP usando una llamada RPC local al portmapper. Un cliente remoto puede entonces consultar al portmapper para la asignación de puerto y enviar las llamadas RPC directamente al servicio. El servicio del portmapper se registra en un puerto TCP y UDP bien conocido, el 111 [105].

Por lo tanto, RPC es un protocolo de sesión, que proporciona un mecanismo por el que una estación hace una llamada de procedimiento que aparece como parte del proceso local, pero que se está ejecutando realmente en otra máquina en la red. Por lo general, la estación en la que se ejecuta la llamada de procedimiento tiene recursos que no están disponibles en la estación que realiza la llamada. Esto impone una relación cliente/servidor entre las dos estaciones [104].

Ⓟ El protocolo MOUNT

Un servidor NFS proporciona un manejador de fichero para cada directorio o fichero al que un cliente necesite acceder. También proporciona la operación LOOKUP que permite al cliente obtener el manejador correspondiente. Un cliente evalúa un *path* como una secuencia de operaciones LOOKUP. Pude parecer extraño, pero el protocolo no proporciona ningún procedimiento para obtener un manejador inicial para la raíz de sistema de ficheros exportado. El manejador inicial se obtiene usando el protocolo MOUNT, como muestra la siguiente figura 3.7.

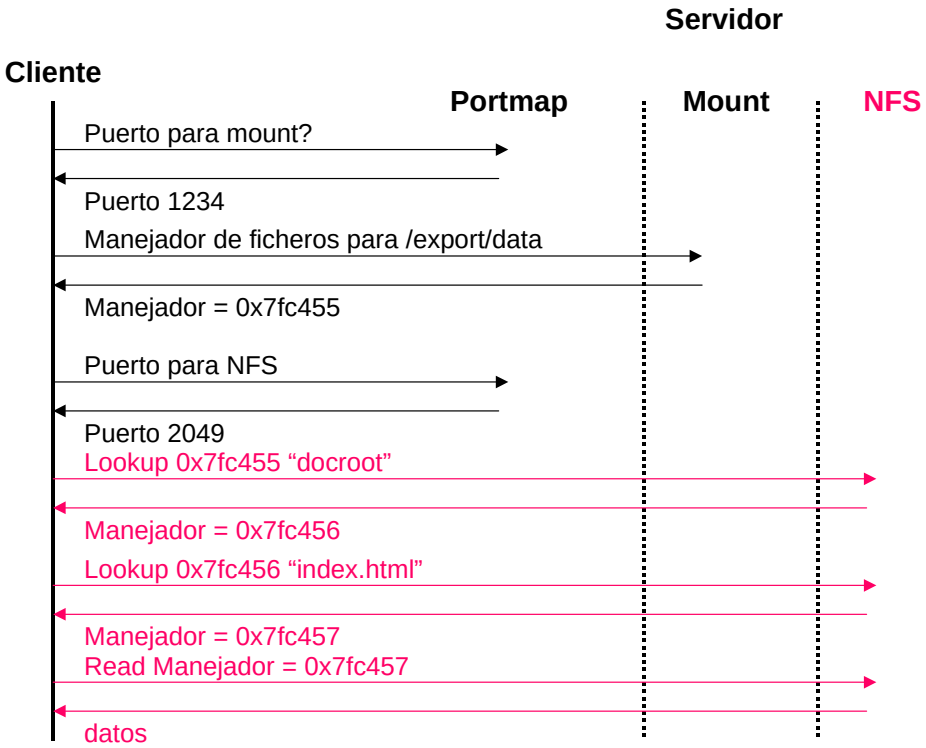


Figura 3.13. Utilización del protocolo *mount*

Un cliente no puede acceder a ningún fichero sin primero obtener un manejador mediante el protocolo mount. El tráfico NFS de un cliente a un servidor empieza con una llamada al portmapper para localizar el puerto UDP para el servicio mount. Una vez que se obtiene el manejador inicial, el cliente se dirigirá de nuevo al portmapper para verificar el puerto NFS. El tráfico inicial NFS empezará probablemente con una serie de llamadas al procedimiento LOOKUP para evaluar un path desde el manejador inicial.

¿Por qué tener un protocolo separado de NFS? ¿Por qué no integrar las características del protocolo mount con las de NFS? El equipo que diseñó el protocolo NFS quiso que estas funciones quedaran fuera del protocolo NFS por dos razones: para chequear el acceso a los sistemas de ficheros exportados, y para no limitar NFS a sistemas UNIX. El demonio mount del servidor mantiene una tabla con los clientes de ese servidor y los sistemas de ficheros exportados que los clientes han montado.

Todas las implementaciones de NFS usan un protocolo mount común, dirigido por la necesidad de interoperar con otras implementaciones. Aunque el protocolo ha tenido dos revisiones no es muy probable que vaya a cambiar más. Los clientes y

servidores WebNFS hacen un uso opcional del protocolo mount, y NFS Versión 4 quiere evitar la necesidad del protocolo mount. Estos dos casos los veremos un poco más adelante.

Ⓟ El protocolo *Network Lock Manager*

El equipo que diseñó el protocolo NFS omitió las operaciones de cerrojo de ficheros a propósito y conscientemente. Los cerrojos por su naturaleza implican un servidor de estado completo con complejidad adicional para recuperar estados posteriores a la caída o al reinicio de un servidor. El protocolo *Network Lock Manager* depende del protocolo RPC, y del protocolo *Network Status Monitor* para notificar a los clientes y /o servidores la pérdida de un estado de cerrojo resultante de una caída o de reiniciar una máquina [105].

3.4.4. El Protocolo NFS

El protocolo NFS consta de un conjunto de procedimientos remotos que habilitan al cliente a manipular ficheros y directorios remotos en el servidor como si fueran locales. Usando las rutinas del protocolo NFS, los clientes envían peticiones sobre operaciones de ficheros a los servidores, y los servidores responden intentando realizar la operación (siempre que el usuario tenga el permiso adecuado) y entonces envía los resultados o los valores de error.

La primera versión, NFS versión 1, nunca fue publicada. La segunda versión, NFS versión 2, está implementada en una gran variedad de sistemas operativos y plataformas hardware, y comprende la mayoría de la base instalada actualmente. En 1993, una nueva revisión del protocolo, NFS versión 3, se diseñó y testeó por primera vez. Esta versión ofrece muchas mejoras sobre la versión 2.

El protocolo NFS se diseñó para una rápida recuperación de fallos. Los parámetros de cada llamada de procedimiento NFS contienen toda la información necesaria para completarlos. Esto no solo habilita a NFS a recuperarse más rápido de los fallos del sistema y de la red, sino que también lo hace más fácil de administrar. El protocolo NFS está diseñado para ser independiente de la máquina, del sistema operativo, de la arquitectura de red, y del protocolo de transporte.

Un sistema de ficheros montado usando NFS proporciona dos niveles de transparencia:

- El sistema de ficheros aparece como residente en un disco unido al sistema local, y todas las entradas del sistema de ficheros (ficheros y directorios) se ven de la misma forma, ya sea local o remoto. NFS oculta la posición del fichero en la red.
- El sistema de ficheros montado no contiene información sobre el servidor de ficheros del que se montó. El servidor de ficheros NFS puede tener una arquitectura diferente o correr un sistema operativo con una estructura de sistemas de ficheros distinto. NFS oculta las diferencias en la estructura del sistema de ficheros remoto y hace parecer a ese sistema de ficheros de igual estructura que el cliente.

NFS alcanza el primer nivel de transparencia definiendo un conjunto genérico de operaciones de sistema de fichero que se realizan en el VFS. El segundo nivel viene de la definición de *nodos virtuales*, que está relacionado con los inodos de UNIX, que ocultan la estructura actual del sistema de ficheros físico debajo de ellos. Los vnodos y la especificación VFS, juntos, definen el protocolo NFS.

Hay una interfaz de llamadas del sistema UNIX única que opera en los ficheros, y el VFS la interfaz de vnodos subyacente traslada la semántica de esas llamadas del sistema en acciones apropiadas para cada tipo de sistema de ficheros. Es importante marcar la diferencia entre *sintaxis* y *semántica* de las llamadas del sistema. Una sintaxis consistente indica que las llamadas del sistema toman los mismos argumentos independientemente del sistema de ficheros subyacente. La semántica se refiere a lo que las llamadas del sistema hacen. Mantener las semánticas a través de diferentes tipos de sistemas de ficheros indica que las llamadas del sistema tendrán los mismos efectos en cada sistema de ficheros. Las semánticas de los sistemas de ficheros UNIX se refieren colectivamente a la forma en que se comportan los ficheros UNIX cuando se hacen varias secuencias de llamadas del sistema

NFS tiene unas semánticas ligeramente diferentes a las del sistema de ficheros local de UNIX, pero intenta mantener las semánticas de UNIX. Una aplicación que trabaje con un sistema de ficheros local, trabaja igual de bien con un sistema de ficheros NFS y no podrá distinguirlos. La consistencia a nivel de la interfaz de vnodo hace a NFS una herramienta poderosa para crear jerarquías de sistemas de ficheros usando muchos servidores NFS distintos.

Los demonios del servidor NFS, llamados demonios *nfsd*, corren en el servidor y aceptan las llamadas RPC de los clientes. Los servidores NFS también ejecutan el demonio *mountd* para gestionar las peticiones de montaje del sistema de ficheros y algunos cambios de nombres. En el cliente, el demonio *biod* (*Block I/O Daemon*) se ejecuta, usualmente, para mejorar el rendimiento de las NFS, pero no es necesario. En Linux no existe este demonio. Las llamadas del sistema del cliente que acceden a los ficheros montados hacen llamadas RPC a los servidores NFS. Cuando en un cliente, un usuario ejecuta, por ejemplo, un chmod (), en un fichero montado NFS, el sistema de ficheros virtual pasa esta llamada del sistema a NFS, que ejecuta entonces una llamada de procedimiento remoto para poner los permisos en el fichero, como se especifique en la llamada. Cuando se completa la RPC, la llamada del sistema vuelve al proceso de usuario.

Los procedimientos básicos realizados en un servidor NFS que veremos a continuación se pueden agrupar en operaciones de directorios, operaciones de ficheros, operaciones para linkar, y operaciones de sistemas de ficheros. Por tanto, las llamadas del sistema UNIX básicas se mapean en llamadas RPC NFS. Señalar, que el protocolo RPC de NFS y la interfaz de v-nodo son cosas diferentes. La interfaz de vnodo define un conjunto de servicios del sistema operativo que se usan para acceder a todos los sistemas de ficheros, NFS o locales. El protocolo RPC de NFS es una realización específica de una de esas interfaces de v-nodo. Se usa para realizar operaciones vnodo específicas en ficheros remotos.

El protocolo NFS es sin estado, lo que indica que no necesita mantener información sobre el protocolo en el servidor. El cliente sigue la pista de toda la información requerida para enviar peticiones al servidor, pero el servidor no tiene información sobre las peticiones NFS previas, o de la relación existente entre varias peticiones NFS.

Todas las operaciones NFS utilizan gestores de ficheros para designar los ficheros o directorios en los que serán realizadas. Los gestores de ficheros se crean en el servidor y contienen información que excepcionalmente identifica al fichero o directorio en el servidor. Estos gestores son opacos al cliente. En muchas implementaciones NFS en máquinas UNIX, son una codificación del número del inodo del fichero, del número del dispositivo de disco, y del número de generación del inodo. Los gestores de ficheros se convierten en inválidos cuando los inodos a los que apuntan (en el servidor) son liberados o re-usados.

Los demonios NFS (nfsd) hacen una llamada del sistema que nunca vuelve, y esa llamada del sistema ejecuta el código NFS apropiado en el kernel del sistema. El correr los demonios en procesos a nivel de usuario permite al servidor tener múltiples e independientes hilos de ejecución, así el servidor puede gestionar algunas peticiones NFS a la vez. Los demonios NFS sirven peticiones en un modo semi-*round robin*. Corriendo múltiples copias del demonio *nfsd* se permite al servidor empezar múltiples operaciones de disco al mismo tiempo y gestionar rápidas peticiones de respuesta. El contexto de usuario asociado con un proceso también permite al kernel parar uno de esos hilos de ejecución, esperando por un disco u otro recurso del sistema [104].

3.4.5. Utilización de la cache en NFS

La cache conlleva mantener los datos que se usan frecuentemente "cercanos" a donde se necesiten, o precargarlos anticipándonos a operaciones futuras. Se puede utilizar la cache en la lectura de datos de disco, hasta que una escritura posterior los invalide, y en las escrituras de datos a disco, de modo que muchos cambios consecutivos del mismo fichero se escriban en una única operación. En NFS, la cache en el lado del cliente indica no tener que enviar una petición RPC sobre la red al servidor: los datos se guardan en la cache del cliente y se pueden leer de la memoria local en lugar del disco remoto. Dependiendo de la estructura y uso del sistema de ficheros, algunos esquemas de cache se prohiben para ciertas operaciones para garantizar la integridad o consistencia de los datos cuando múltiples procesos están leyendo o escribiendo en el mismo fichero. Las políticas de cache en NFS aseguran que el rendimiento es aceptable mientras que también se previene la introducción de estado en la relación cliente-servidor.

No todas las operaciones de los sistemas de ficheros tocan los datos de los ficheros; muchas de ellas cogen o ponen los atributos del fichero tales como la longitud, propietario, tiempo de modificación, y número de inodo. Ya que estas operaciones de solo atributos son frecuentes y no afectan a los datos en el fichero, son las principales candidatas para usar la cache. Cosas como ls -l son un ejemplo clásico de una operación de sólo atributos: toma información sobre los directorios y ficheros, pero no mira el contenido de los mismos.

Un cliente NFS necesita mantener consistencia de cache con la copia del fichero en el servidor NFS. Se usan atributos de ficheros para realizar chequeos de consistencia. El tiempo de modificación del fichero se usa como chequeo de validación de la cache.

Pero entremos un poco más en detalle, y veamos las técnicas de *caching* y *prefetching* utilizadas en NFS. En esta sección vamos a ver una primera parte introductoria que nos va aclarar en qué consiste el *caching* y *prefetching* y también nos va a explicar las técnicas de *caching* y *prefetching* existentes en NFS.

Después, en el capítulo de resultados experimentales, volveremos a este tema para ver los efectos de estas técnicas gráficamente y además hacer distintos estudios analíticos para completar este desarrollo experimental preliminar.

El término *caching* se refiere al *almacenamiento temporal de datos de fichero en un dispositivo de almacenamiento local de rápido acceso llamado cache* [102]. Después del *caching*, el *prefetching* es la siguiente etapa lógica para incrementar el rendimiento de los sistemas de E/S.

Se han realizado variados e interesantes trabajos sobre *prefetching* en los últimos 10 años. Desde la publicación en 1991 del trabajo *"Optimal Prefetching via Data Compression"* de Vitter y Krishman [109], tantas veces referenciado en trabajos posteriores, hasta las más actuales publicaciones orientadas a multimedia y entornos inalámbricos [110, 111], existen toda una serie de trabajos en los que se diseñan, implementan, simulan y testean distintos algoritmos de *prefetching* útiles en distintas situaciones y con distinto propósito, ya sea con sistemas uni o multiprocesador u orientado a clusters, para acceso secuencial o paralelo, para predicción de bloque de datos o de ficheros completos, pero eso sí, todos con el mismo propósito: mejorar el rendimiento del sistema anticipándose a las peticiones de los usuarios. Por tanto, el prefetching se podría definir como una *técnica general que mejora el rendimiento de E/S leyendo datos en la cache antes de que se pidan, solapando E/S con computación.*

Como ya hemos comentado, el caching implica mantener los datos más frecuentemente usados cerca de donde se necesitan, o precargar datos antes de que operaciones futuras los necesiten. Las lecturas de datos de disco pueden ser cacheadas hasta que escrituras posteriores las hagan inválidas. En NFS, el caching de datos en el cliente indica no tener que enviar una petición RPC sobre la red al servidor: los datos se cachean y pueden leerse de la memoria local en lugar de un disco remoto. Las políticas de cache en NFS aseguran un rendimiento aceptable.

Existen distintas formas de *caching*, veámoslas detenidamente

Caching de atributos

Una vez que los datos de fichero (incluyendo páginas e información de atributos y directorios) son cacheados, las peticiones de cliente siguientes van directamente a la cache y no requieren transferencias de datos sobre la red [102].

Si cada operación en un fichero remoto necesitase un acceso a la red, el rendimiento NFS podría ser intolerablemente lento. De ahí que muchos clientes NFS recurran tanto a caching de bloques de datos de fichero como de atributos de fichero [112].

Y es que no todas las operaciones de los sistemas de ficheros tocan los datos de los ficheros, como ya hemos comentado, muchas de ellas sólo toman o ponen los atributos de los ficheros (longitud, propietario, tiempo de modificación, nº de inodo, etc.). Además, este tipo de operaciones son muy frecuentes, por lo que son las primeras candidatas para usar cache. NFS cachea los atributos de ficheros en el lado del cliente tal que operaciones como GETATTR y SETATTR no tienen que ir todas las veces al servidor NFS. Cuando se leen los atributos de un fichero, quedan válidos en el cliente por algún periodo mínimo de tiempo. Si el cliente modifica el fichero, ese cambio se hace en la copia local de los atributos y el periodo de validez se extiende otro tiempo mínimo. Si los atributos se mantienen estáticos por algún periodo máximo (normalmente 60 segundos) se limpian de la cache y se escriben al servidor si han sido modificados.

El mismo mecanismo se usa para atributos de directorio, aunque se dan por un tiempo de vida mínimo más largo (ya que la actualización de los directorios es menos frecuente). El caching de atributos permite a un cliente hacer un flujo de actualizaciones a un fichero sin tener que tomar o poner continuamente los atributos en el servidor. Se cachean los atributos intermedios y el efecto de todas las actualizaciones es finalmente escrito al servidor cuando el periodo de cache de atributos máximo expira. Estos son los únicos datos del sistema de ficheros que no se cometen a almacenamiento no volátil después de ser modificados.

Esta cache es peligrosa, ya que el cliente no tiene forma de saber si los contenidos de la cache son válidos, por lo tanto, se deben enviar consultas al servidor cada cierto tiempo. Los clientes chequean la consistencia de la cache verificando que el tiempo de modificación del fichero no ha cambiado desde que los datos cacheados se leyeron del servidor [104]. Si los datos cacheados son más nuevos que el tiempo de modificación, entonces se mantienen válidos, sino es así, entonces los datos deben limpiarse. Los bloques de datos en la cache introducen problemas de consistencia si uno o más clientes abren el fichero para escribir, que es una de las motivaciones para limitar el periodo de validez de la cache de atributos. Si los datos de la cache de atributos no expiran nunca, los clientes que abren ficheros para leer no tendrían razón para chequear el servidor para posibles modificaciones por otros clientes. Las operaciones de NFS sin estado requieren que cada cliente sea ajeno a los demás y confíen en su cache de atributos solo para asegurar consistencia.

Los tiempos de vida de los datos cacheados se determinan por cuatro parámetros en el instante de montaje:

Acregmin: tiempo de vida mínimo para atributos de fichero (3 s por defecto)

Acregmax: tiempo de vida máximo para atributos de fichero (60 s por defecto)

Acdirmin: tiempo de vida mínimo para atributos de directorio (30 s por defecto)

Acdirmax: tiempo de vida máximo para atributos de directorio (60 s por defecto) [104].

Caching de datos en el cliente

En la implementación de NFS de Solaris existe un demonio (biod, *block I/O daemon*) que realiza operaciones de *read-ahead* y *write-behind* para los procesos cliente NFS. La implementación de NFS en Linux es algo diferente: el código de cliente está integrado en la capa del sistema de ficheros virtual y no requiere control adicional mediante el programa biod [113]. NFS mueve los datos en buffers en lugar de en páginas. El uso de estos buffers permite a las operaciones NFS utilizar algunas de las optimizaciones de E/S a disco secuencial de los drivers de dispositivo de disco UNIX.

Además, si el kernel del cliente detecta que la aplicación del cliente está realizando lectura secuencial de datos, intenta hacer prefetching de nuevos bloques de datos.

Se ve claramente con un ejemplo donde se muestran solo las peticiones de un cliente (myr4) al servidor (myr1):

Ejemplo 1

```
20:13:32.716234     myr4 > myr1: 116 read 4096 bytes @ 0
20:13:32.716768     myr4 > myr1: 116 read 4096 bytes @ 4096
20:13:32.716824     myr4 > myr1: 116 read 4096 bytes @ 8192
                ...
20:13:32.725259     myr4 > myr1: 116 read 4096 bytes @ 126976 (**)
20:13:32.726381     myr4 > myr1: 116 read 4096 bytes @ 135168
```

Se producen una serie de lecturas secuenciales sobre un mismo fichero en trozos de 4 kbytes (lo especificado con rsize, un parámetro para especificar el tamaño del buffer de lectura que describiremos a continuación) hasta que el kernel detecta la forma de leer y en la línea marcada (**) se leen 8096 bytes seguidos (un buffer NFS completo). Es decir se leen 8096 bytes empezando en el offset 126976 (en trozos de 4096 bytes). Otro ejemplo más completo se puede ver en [112].

Por tanto, el caching en el lado de los clientes es esencial para un rendimiento aceptable [100]. Incrementa significativamente el rendimiento de NFS en el lado de los clientes y también realza la escalabilidad del servidor. Los clientes acceden más rápido a los ficheros almacenando grandes trozos de datos en una cache local, de rápido

acceso. Si el fichero se lee secuencialmente, el cliente NFS puede anticiparse a los requisitos de datos futuros a través del proceso llamado *read-ahead* y puede almacenar esta información en su cache local para futuras referencias.

El *caching* de grandes cantidades de información en el cliente indica menos peticiones al servidor, por lo tanto la carga del servidor decrece y la escalabilidad aumenta.

Caching en el lado del servidor

Empezamos tratando el tema de las retransmisiones. El cliente retransmite las peticiones repetidamente hasta que recibe una respuesta. Las retransmisiones ocurren por pérdida de paquetes (las peticiones originales o las respuestas) en la red o porque el servidor no puede responder lo bastante rápido. El servidor necesita gestionar esas peticiones duplicadas de forma eficiente y correcta.

Ya hemos comentado que las peticiones NFS se dividen en *Idempotentes* (como READ, LOOKUP, GETATTR) y en n*o idempotentes* (CREATE, REMOVE, RMDIR), el volver a mandar la misma respuesta no es problema con las operaciones idempotentes. Pero no es problema porque cada petición READ, por ejemplo, especifica el offset de comienzo de la lectura. Si hubiera un procedimiento NFS pidiendo al servidor leer los siguientes N bytes de un fichero, esto no iría bien. A menos que el servidor fuera de “estado completo”. Si una réplica se pierde y el cliente repite el READ para los siguientes N bytes, el resultado sería diferente. Esto es porque de los procedimientos READ y WRITE el cliente tiene específicamente el offset de comienzo. Pero es el cliente el que mantiene el estado (el offset actual de cada fichero) no el servidor. Desafortunadamente no todas las operaciones de los sistemas de ficheros son idempotentes. Este concepto de procedimientos de servidor idempotente se aplica a cualquier aplicación basada en UDP, no solo a NFS. Por ejemplo el DNS también lo usa.

Ya que la pérdida de respuestas puede ocurrir siempre con UDP, los servidores necesitan una forma de gestionar las operaciones no idempotentes. Muchos servidores implementan una *cache de respuestas recientes* en la que almacenan las respuestas para las operaciones no idempotentes. Es decir, es completamente necesario gestionar y detectar correctamente las retransmisiones. Para hacer esto, el servidor mantiene una cache de peticiones recientes. La gestión de esta cache ha ido evolucionando con las distintas versiones de NFS. La idea básica es procesar la petición si es idempotente, y si es no idempotente chequear la cache y buscar coincidencias. Si el servidor encuentra la peticiones en la cache y su estado es *in progress* simplemente descarta el duplicado, y si el estado es *done*, el servidor la descarta si se ha completado recientemente. Las entradas de la cache se reciclan en base a la técnica LRU (*Least Recently Used*). Esta cache de retransmisiones ayuda a eliminar algunas operaciones duplicadas, mejorando el rendimiento y la exactitud del servidor [104].

Como hemos comentado, los mecanismos de caching del cliente (atributos y buffer) reducen el número de peticiones que necesitan se enviadas al servidor NFS. Pero, en el servidor también existen políticas de cache adicionales que reducen el tiempo requerido para servir estas peticiones. Los servidores NFS mantienen también las siguientes caches:

La cache de inodos que contiene los atributos de ficheros (el tener estos atributos en memoria en lugar de en disco hace las operaciones de coger y poner atributos mucha más rápidas)

La cache lookup de nombres de directorios o DNLC, que contiene las entradas de directorios leídos recientemente. Así el servidor no tiene que abrir y releer directorios en cada resolución de *pathname*s.

La cache del buffer del servidor, usada para los datos leídos de ficheros. Con las escrituras síncronas, los bloques de ficheros que se escriben a un servidor NFS no se pueden cachear, deben ser escritos antes de que la llamada WRITE del cliente se complete. Sin embargo, el buffer del servidor actúa como una eficiente cache de lectura para los clientes NFS. Los efectos de este caching son más pronunciados en sistemas mapeados por páginas ya que casi toda la memoria del servidor se puede usar como una cache de lectura para bloques de ficheros [104].

Consistencia de la cache

Los datos almacenados en el servidor pueden ser cacheados por múltiples clientes, por lo tanto existe la posibilidad de que los datos cacheados en el cliente sean inconsistentes con los almacenados en el servidor o en otros clientes. El protocolo no proporciona facilidades que garanticen siempre la consistencia, en su lugar se hace un esfuerzo por parte de los clientes por mantener los datos cacheados en sincronía con los del servidor.

Normalmente los esquemas de caching implementados usan dos tiempos: un tiempo de cache y un tiempo de modificación. Cuando el servido cachea datos, también cachea el tiempo de modificación para esos datos (además de otros atributos). Un tiempo de cache asociado con los datos puede variar dependiendo del tipo de datos (fichero o directorio). Si los datos se referencian en el tiempo de cache, entonces se usan los datos cacheados y no se consulta al servidor. Si se excede este tiempo de cache, el cliente contactará con el servidor y verificará que el tiempo de modificación no ha cambiado. Si efectivamente no ha cambiado, entonces se inicializa el tiempo de cache y se usan los datos de la cache. Pero, si el tiempo de modificación es diferente, entonces los datos cacheados son inválidos y hay que obtenerlos de nuevo del servidor. Un cliente que usa el tiempo de modificación como atributo para el caching está haciendo la suposición de que el servidor actualizará el tiempo de modificación si el fichero o el directorio cambian

El tiempo de cache es un compromiso ente consistencia de cache y carga del servidor y de la red. Si es pequeño, entonces la consistencia de cache será alta, pero se consultará el servidor frecuentemente para chequear si el tiempo de modificación ha

cambiado. Si el tiempo de cache se pone a cero, entonces se consultará al servidor siempre que se acceda a la cache. Si el tiempo de cache es grande, entonces el servidor será consultado menos frecuentemente, pero habrá más posibilidad de que el cliente utilice datos inválidos (figura 3.8).

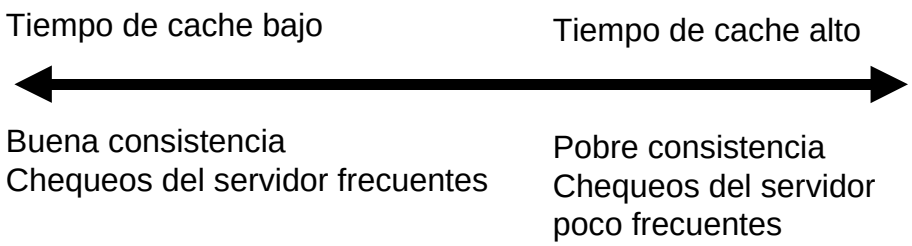


Figura 3.14. Efectos del tiempo de cache

El tiempo de cache debe ser pequeño para servidores que cambian frecuentemente sus datos, y grande si el servidor apenas varía sus datos. Los directorios suelen variar menos que los ficheros, de ahí que existan parámetros para asignar tiempos de cache máximos y mínimos a ficheros y directorios de forma separada

3.4.6. Versiones de NFS

■ NFS versión 2

Fue la primera versión publicada del protocolo NFS. Su implementación empezó en marzo de 1984, se basaba en el kernel UNIX de SunOS, que estaba basado en BSD 4.2. Al poco tiempo de presentarse, apareció la primera versión para PC. Esta versión 2 se recibió muy bien por los consumidores. Hizo posible compartir los datos, backup de ficheros fáciles, y almacenamiento en disco más barato. Ha sido totalmente suplantada por la versión 3 que estudiaremos en detalle, pero sigue siendo todavía la versión más popular de NFS [106].

Tabla 3.1. Procedimientos de la versión 2

NÚMERO	NOMBRE	DESCRIPCIÓN
0	NULL	Procedimiento NULL
1	GETATTR	Toma los atributos del fichero
2	SETATTR	Pone los atributos del fichero
3	ROOT	No implementado
4	LOOKUP	Busca el nombre del fichero
5	READLINK	Lee de un enlace simbólico
6	READ	Lee de un fichero
7	WRITECACHE	No implementado
8	WRITE	Escribe a un fichero
9	CREATE	Crea un fichero
10	REMOVE	Borra un fichero

11	RENAME	Renombra un fichero
12	LINK	Crea un enlace a un fichero
13	SYMLINK	Crea un enlace simbólico
14	MKDIR	Crea un directorio
15	RMDIR	Elimina un directorio
16	READDIR	Lee un directorio
17	STATFS	Toma los atributos de un sistema de ficheros

Los procedimientos RPC que suministra el servidor del protocolo NFS versión 2 aparecen en la tabla 3.1. Más detalles de implementación, argumentos y códigos de error se pueden ver en [105]

■ NFS versión 3

Aunque la versión 2 se ha implementado con éxito en diferentes sistemas operativos, su orientación UNIX y su simplicidad hacen difícil que el protocolo proporcione acceso completo a las características que ofrecen distintos sistemas operativos en los que se puede implementar. Así que los ingenieros de Sun empezaron a recibir muchas peticiones de mejoras del protocolo [105]

En 1992, un grupo de vendedores como IBM, Digital, SunSoft y otros, empezaron el proceso de definición de la nueva revisión del protocolo NFS. El resultado fue NFS versión 3 [114], que ofrece mejoras sustanciales sobre NFS versión 2. Estas mejoras incluyen las siguientes, aunque no están limitadas: mejora de las operaciones de escritura en el lado del cliente, reducción de la carga del servidor debido al incremento de escalabilidad y rendimiento, soporte mejorado para sistemas que utilizan Listas de Control de Acceso (ACLs), soporte para ficheros grandes (multi-gigabytes) en los servidores NFS.

Debido a sus múltiples beneficios, la industria se está convirtiendo a la versión 3 del protocolo NFS. Sin embargo para permitir la compatibilidad hacia atrás con lo ya instalado, es posible implementar NFS para que soporte ambos protocolos concurrentemente. La revisión del protocolo NFS conlleva nuevos requisitos. Las modificaciones son de tres tipos:

1. Se reduce el número de paquetes para un conjunto dado de operaciones de fichero, devolviendo los atributos de fichero en cada operación, esto disminuye el número de llamadas para coger los atributos modificados.
2. Se han tratado los embotellamientos de escritura debido a la definición síncrona de la operación de escritura en la especificación del protocolo original, añadiendo soporte para que el servidor NFS pueda hacer escrituras inseguras. Estas escrituras son aquellas que no han sido asignadas a un almacenamiento estable antes de que termine la operación. Esta especificación define un método para asignar esas escrituras inseguras a un almacenamiento estable de una forma fiable.
3. Se han relajado las limitaciones en los tamaños de las transferencias.

La habilidad para soportar múltiples versiones de un protocolo en RPC permitirá a los implementadores de la versión 3 de NFS definir clientes y servidores que proporciones compatibilidad hacia atrás con la base instalada existente de las implementaciones de NFS versión 2 [115].

Para poder sacar partido de estas ventajas, la versión 3 del protocolo debe estar tanto en los servidores como en los clientes. Como ya se ha mencionado, esta versión permite escrituras asíncronas seguras en el servidor, lo que mejora el rendimiento, permitiendo al servidor guardar en la memoria cache las peticiones de escritura del cliente. El cliente no tiene que esperar que el servidor cometa los cambios en disco, así el tiempo de respuesta es más rápido. El servidor también puede agrupar las peticiones, lo que mejora el tiempo de respuesta en el servidor. Por otro lado, como todas las operaciones devuelven los atributos de los ficheros, se reduce el número de llamadas RPC al servidor, mejorándose también el rendimiento. También se ha mejorado el proceso para verificar los permisos de acceso. Por último, la versión 3 elimina el límite de tamaño de la transferencia de datos de 8 kbytes. Los clientes y servidores negociarán qué tamaño de transferencia van a soportar, en lugar de estar restringidos a los 8 kbytes de la versión 2. En la implementación de Solaris 2.5 el tamaño de trasferencia por defecto es 32 kbytes [103].

El rendimiento de NFS creció un orden de magnitud de 1990 a 1995, de cientos de operaciones por segundo a miles. La versión 3, la cache en el lado del cliente, y el ajuste general han contribuido a este fenómeno.

NFS versión 3 introduce una variedad de mejoras de rendimiento y escalabilidad de NFS como son:

- *Mejoras en el rendimiento de las operaciones de escritura*: las aplicaciones que corren en el cliente pueden escribir periódicamente datos en un fichero, cambiando su contenido. La cantidad de tiempo que una aplicación espera para que sus datos se escriban en un almacenamiento estable en el servidor, es una medida del rendimiento de escritura de un sistema de ficheros distribuido, y un aspecto importante en el rendimiento final. La versión 3 de NFS mejora estos tiempos de escritura con respecto a la versión 2.

- *Mejoras en el rendimiento de las operaciones de lectura:* esto se puede definir como la cantidad de tiempo que las aplicaciones esperan hasta que se hacen disponibles los datos de un fichero, después de una petición de lectura. Tanto la versión 2 como la 3 utilizan *caching* de disco local y *read-ahead*, para mejorar estos tiempos. Además, los clientes NFS también mantienen la cache después de cerrar un fichero. Esto es para que en el caso común en el que un fichero es reabierto, las peticiones de lectura se servirán con datos que ya existen en la cache. Todas estas características reducen el tiempo de espera y mejoran los tiempos de lectura.

- *Peticiones reducidas para los atributos de ficheros:* los clientes deben chequear sus datos de la cache para que no se conviertan en inválidos por los cambios hechos por otra aplicación. Además, los clientes NFS obtienen periódicamente los atributos de los ficheros, que incluyen el tiempo de la

última modificación. La versión 3 devuelve los atributos en todas las operaciones. Esto incrementa la probabilidad de que los atributos en la cache estén actualizados y reduce el número de peticiones de atributos [102].

En definitiva, se modificaron los procedimientos LOOKUP, READ, WRITE, CREATE Y READDIR. Y se crearon nuevos procedimientos:

- ACCESS: proporciona un chequeo de permisos explícito y elimina la suposición de la versión 2 de que el acceso a ficheros se controle mediante bits de modo al estilo UNIX
- MKNOD: soporta la creación de ficheros especiales, lo que evita la sobrecarga de los campos del procedimiento CREATE que se hizo en algunas implementaciones de la versión 2
- READDIRPLUS: extiende la funcionalidad de READDIR devolviendo no sólo los nombres de los ficheros y los identificadores, sino también los manejadores de fichero y los atributos por cada entrada en un directorio
- FSINFO: proporciona información no volátil sobre un sistema de ficheros. La respuesta incluye el tamaño de transferencia de lectura máximo y preferido, el tamaño de transferencia de escritura máximo y preferido, y flags de estado indicando si se soportan enlaces duros o enlaces simbólicos
- FSSTAT: sustituye el procedimiento STATFS de la versión 2, proporciona información volátil sobre el sistema de ficheros que usarán utilidades como el comando df de UNIX. La respuesta incluye el tamaño total y el espacio libre en el sistema de ficheros especificado en bytes, el número total de ficheros y el número de slots de ficheros libres en el sistema de ficheros, y una estimación del tiempo entre modificación de sistemas de ficheros.
- PATHCONF: devuelve la información necesaria sobre las características del sistema de ficheros para soportar la llamada pathconf de POSIX.
- COMMIT: traslada los datos situados en buffers en el servidor a almacenamiento estable. Se usa junto a los nuevos modos de escritura asíncrona.

Más detalles sobre todos estos procedimientos se pueden encontrar en [105].

🖿 Web NFS. Un sistema de ficheros par el web

El Web está basado en el lenguaje HTML y en el simple HTTP, como ya sabemos. Aunque es muy simple de implementar y entender, HTTP es un protocolo caro en términos de sobrecarga de conexiones y transferencia de datos. Cada objeto que transferimos vía HTTP requiere una nueva conexión TCP. Además, debemos transferir el objeto completo cada vez. Cada página HTML puede contener referencias a otros objetos que debemos bajarnos para completar la página entera. Esto requiere conexiones TCP adicionales.

Algunos navegadores como el Netscape han adoptado un modelo de hilos que permite accesos HTTP múltiples concurrentes por página HTML. Mientras que estos hilos ayudan a evitar latencias por las conexiones TCP, incrementan la carga vista por el servidor.

Lo que se necesita son tecnologías de acceso a datos web más eficientes que dejen a los usuarios acceder de forma selectiva, manipular y actualizar datos como algo habitual. Sun Mycrosystem cree que tiene una tecnología en su inventario que puede proporcionar la solución con un pequeño repaso.

Sun considera cualquier uso Web de NFS como una forma de *WebNFS*, tanto su NFS v2 o v3, o NFS con realces, WebNFS. Esto puede desconcertar un poco a los usuarios, ya que no dan una forma específica del protocolo que puede ser etiquetado como WebNFS. Así que nos centraremos sólo en exponer la versión realzada para el Web de NFS, WebNFS, en lugar de usar otras especificaciones.

Si NFS es tan bueno, ¿Cómo que no se ha usado en Internet? NFS es un protocolo eficiente que está optimizado para LANs. Como tal, se publicó originalmente en UDP, que no proporciona mecanismos de control de flujo ni recuperación de errores, aparte de los *timeouts* propios de NFS. Debido a esto, NFS no se ha utilizado sobre Internet. Con la llegada de NFS v3, TCP se convirtió en el protocolo preferido. Ofrece control de flujo, transferencia fiable y otras características que UDP no tiene.

NFS sin modificaciones no es un buen es un protocolo para redes de alta latencia, como se muestra en la figura 3.7. Para mejorar su rendimiento, NFS necesita eliminar la sobrecarga debida al mapeo de puertos, el montaje y la recursión de nombres. Para llevar esto a cabo, WebNFS hace tres suposiciones: el puerto por defecto para NFS es el 2049, se puede exportar un directorio como "public" con un gestor conocido, y los delimitadores de nombres son similares a un URL (*Uniform Resource Locator*) de HTTP.

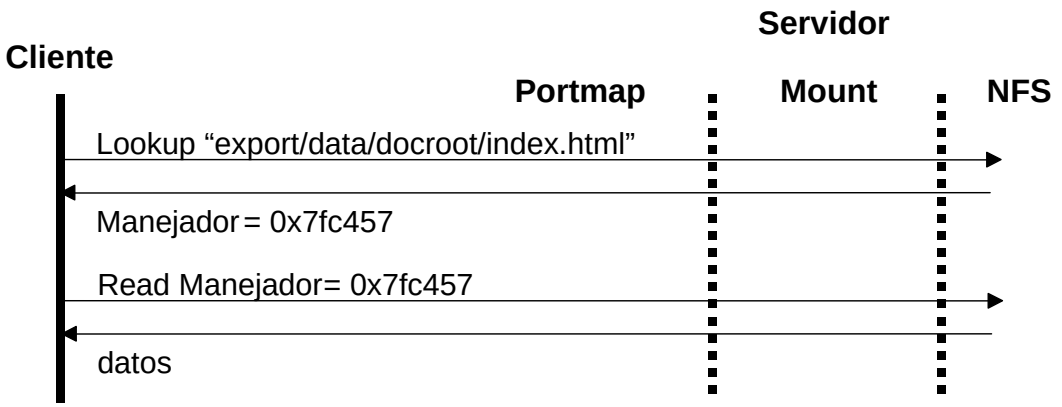


Figura 3.15. Accesos con webNFS

De este modo, WebNFS introduce un nuevo tipo de URL, el URL *nfs*, que se especifican con el formato nfs://server:port/path, similar al usado por HTTP. Como hemos dicho, WebNFS usa el puerto NFS por defecto 2049, a menos que la URL especifique uno. Las etapas del protocolo modificado se ilustran en la siguiente figura.

Aunque el uso de WebNFS proporciona significantes mejoras y beneficios de uso, tiene algunas limitaciones. Estas limitaciones están relacionadas con el hecho de que NFS implementa un sistema de ficheros. Un sistema de ficheros en red implementa las semánticas de un sistema de ficheros en un disco local. Como resultado, muchas características que proporciona HTTP no se pueden soportar directamente por WebNFS.

Hay otra importante limitación: es imposible soportar aplicaciones de servidor sin modificaciones radicales al servidor NFS. Aunque WebNFS tiene la habilidad técnica de convertirse en una tecnología Web importante [116, 117, 118].

Usando WebNFS, un cliente puede conectarse directamente al servidor NFS y obtener un manejador de ficheros inicial mediante la petición LOOKUP relativa al manejador de ficheros público. Este se identifica como un manejador rellenado con ceros (versión 2) o de longitud cero (versión 3). Si la petición LOOKUP incluye un path separado por barras, se le da el nombre de LOOKUP multicomponente. El cliente puede usar una forma especial de LOOKUP multicomponente para recuperar una lista de todos los niveles de seguridad válidos del servidor [105].

■ NFS versión 4

Es un protocolo de sistema de fichero distribuido que hereda características de las versiones 2 y 3 del protocolo NFS. A diferencia de las versiones anteriores, la versión 4 va a soportar el acceso tradicional a los ficheros integrando soporte para *locking file* y para el protocolo *mount*. Además, soporta una fuerte seguridad (y su negociación), operaciones compuestas, cache de cliente e internacionalización. Por supuesto, se ha prestado especial atención para que la versión 4 de NFS opere bien en un entorno de Internet [119].

Cuando se diseñó el protocolo NFS, las máquinas no eran tan potentes como las actuales, las redes más usadas eran las de área local en lugar de las de área amplia, y los mecanismos de seguridad disponibles eran relativamente fáciles de explotar. Además, aunque NFS se diseñó para interoperar entre distintos sistemas operativos, muchas características del protocolo están muy de acuerdo con la semántica de los ficheros UNIX.

Esta historia ha llevado a problemas en el uso de soluciones NFS sobre Internet, donde la seguridad, rendimiento e interoperatibilidad son elementos clave. La versión 4 de NFS se ha diseñado para tratar estos temas, proporcionando las siguientes novedades: acceso mejorado y mejor rendimiento en Internet, fuerte seguridad, con negociación de seguridad construida en el protocolo, interoperatibilidad realzada a través del protocolo y extensibilidad del protocolo

A diferencia de las versiones 2 ó 3, NFS versión 4 puede ser un importante elemento en la estrategia de las empresas para proporcionar mejor soporte para redes globales.

Tabla 3.2. Comparación de las características de NFS versiones 2, 3 y 4.

CARACTERÍSTICA	VENTAJAS	NFS V2	NFS V3	NFS V4
Cache del disco local	Realza el rendimiento del cliente incrementando el espacio de cache disponible	✔	✔	✔
Montaje automático	Hace el sistema de ficheros global accesible a los usuarios de forma continua y transparente	✔	✔	✔
Administración centralizada	Reduce tiempo y esfuerzo en tareas de administración	✔	✔	✔
Espacio de nombre global	Crea nombres de ficheros válidos a través de la red, así los usuarios y las aplicaciones se pueden mover libremente	✔	✔	✔
Determinación del cliente de la versión de protocolo soportada por el servidor	Los clientes y servidores negocian que protocolo usar en base a los que pueden soportar. Son compatibles hacia atrás.		✔	✔
Peticiones reducidas de atributos	Incrementa la escalabilidad y el rendimiento		✔	✔
Peticiones reducidas para información de *lookup*			✔	✔
Escrituras asíncronas	Mejora el rendimiento de las escrituras del cliente			✔
Procedimiento COMPOUND	Mejora el rendimiento en redes de alta latencia			✔
Uso de un puerto bien conocido, 2049, para servicios NFS	Transmisiones más fáciles			✔
Conjunto de caracteres universal UTF-8	Soporta internacionalización con múltiples lenguas en nombres de ficheros y vías			✔
Construido en el protocolo RPCSEC_GSS	Fortalece la seguridad, soporta múltiples mecanismos de seguridad			✔
Gestión volátil de ficheros	Acomoda los entornos de servidor donde la gestión de ficheros persistente no está implementada			✔
Atributos Flexibles	Proporciona extensibilidad para nuevos atributos e indicadores que indican no-soporte			✔
Espacios de nombres del servidor	Todo lo exportado se presenta en el marco de trabajo de un único espacio de nombres del servidor			✔
Locking de ficheros	Soporta compartir *locks*			✔

Con la asistencia del comité continuado de Sun para estándares abiertos y el trabajo con la IETF (*The Internet Engineering Task Force*), el protocolo para NFS versión 4 está disponible de forma abierta, es interoperable, y proporciona excelente servicio en entornos de Internet e intranet. Mostramos algunas características básicas de NFS en la tabla 3.3 .

Como se muestra en la tabla 3.3, NFS versión 4 reúne los requisitos para un rápido, fiable, seguro, y transparente servicio de ficheros sobre Internet. NFS versión 4 implementa las características básicas para mejorar el acceso a Internet y el rendimiento, añade fuertes medidas de seguridad, realza la interoperatibilidad de la plataforma, y proporciona un protocolo extensible que llevará a NFS por las demandas del futuro. Los actuales usuarios de NFS versión 2 y 3 encontrarán que la versión 4 es un importante enlace en su migración hacia un mundo que gira alrededor de Internet e intranet [120].

Uno de los grupos de trabajo activos del IETF, concretamente el del área de transportes, se está dedicando a la especificación y desarrollo de la versión 4 de NFS. El objetivo de este grupo de trabajo es avanzar el estado de la tecnología NFS para producir una especificación de la versión 4 que sea admitida como un RFC estándar de Internet. La primera fase del grupo de trabajo producirá un documento de requisitos que describa las limitaciones y deficiencias de la versión 3 de NFS, las soluciones potenciales para ellas, y un análisis costos/beneficios de las diferentes soluciones. Se ha tenido en cuenta para este documento las experiencias con otros sistemas de ficheros distribuidos tales como DCE/DFS y Coda. Después de la publicación de este documento se volverán a revisar los principios de este grupo de trabajo. Sin embargo, se deja claro que NFS v4 tratará los siguientes asuntos:

- *Acceso mejorado y buen rendimiento en Internet*: se diseñará el protocolo para funcionar bien cuando la latencia sea alta y el ancho de banda bajo, para adaptar la presencia de congestión y escalar un gran número de clientes por servidor.
- *Fuerte seguridad con negociación en el protocolo*: el protocolo puede construirse en el grupo de trabajo ONCRPC soportando el protocolo RPCSEC_GSS. El modelo de permisos necesario para escalar más allá del espacio UID. Además NFS v4 proporciona un mecanismo que permite a los clientes y servidores negociar la seguridad, y exige que clientes y servidores soporten un conjunto mínimo de esquemas de seguridad.
- *Mejor interoperatibilidad a través de la plataforma*: el protocolo publicará un modelo de sistema de fichero que proporcione un conjunto común y útil de características que no favorezca excesivamente a un sistema de ficheros o sistema operativo más que a otro.
- *Diseñado para extensiones de protocolo*: el protocolo se diseñará para aceptar extensiones estándares que no comprometan la compatibilidad hacia atrás [121].

Veamos, brevemente, algunas de las características principales de esta nueva versión:

1. *RPC y Seguridad*: como en las versiones previas, NFS v4 utiliza los mecanismos de XDR (*External Data Representation*) y RPC (*Remote*

Procedure Call). Para conseguir los requisitos de seguridad, se utiliza el marco de trabajo RPCSEG_GSS que extiende la seguridad básica de RPC. Así, se proporcionan varios mecanismos de autentificación, integridad y privacidad a la V4.

2. *Estructura de las operaciones y procedimientos*: una diferencia importante con respecto a las versiones previas es la introducción del procedimiento COMPOUND. Se define en términos de operaciones y esas operaciones se corresponden a los procedimientos NFS tradicionales. Al usar este procedimiento, el cliente es capaz de construir peticiones simples o complejas. Estas peticiones COMPOUND permiten una reducción en el número de RPCs necesarios para operaciones lógicas de sistemas de ficheros. Por ejemplo: sin contacto previo con el servidor, un cliente será capaz de leer datos en un único RPC COMPOUND. Con versiones previas del protocolo, este tipo de peticiones únicas no era posible. El modelo usado por COMPOUND es muy simple. Las operaciones combinadas es una petición COMPOUND se evalúan en orden por el servidor. Una vez que una operación devuelva un resultado fallido, la evaluación acaba y los resultados de todas las operaciones evaluadas se devuelven al cliente. En la V4 se mantiene que el cliente se refiere a un fichero o directorio en el servidor por medio de un gestor de fichero. El procedimiento COMPOUND tiene una forma de pasar el gestor de ficheros de una operación a otra en la secuencia de operaciones.

3. *El modelo de sistema de ficheros*: El modelo de sistema de ficheros general usado por NFS V4 es el mismo que en las versiones previas. El sistema de ficheros del servidor es jerárquico, con los ficheros regulares contenidos tratados como flujos de bytes opacos. La V4 del protocolo no requiere un protocolo a parte para proporcionar el mapeo inicial entre el pathname y el gestor de ficheros. En lugar de usar el anterior protocolo MOUNT para este mapeo, el servidor proporciona un gestor de ficheros ROOT que representa el root lógico o el tope del árbol del sistema de ficheros proporcionado por el servidor. También distingue distintos t*ipos de gestores de ficheros* y distintos t*ipos de atributos.*

4. *OPEN* y *CLOSE:* la V4 del protocolo introduce las operaciones OPEN y CLOSE. La operación OPEN proporciona un único punto donde se pueden combinar las semánticas de lookup, crear y compartir ficheros. La operación CLOSE proporciona la liberación del estado acumulado por OPEN.

5. *Locking de ficheros*: con la V4, el soporte para locking de ficheros parte del protocolo NFS. Ya no se necesitan mecanismos callback de RPC. Esta es una diferencia de las versiones previas del protocolo de locking de ficheros NFS, el NLM (*Network Lock Manager*). El estado asociado con los locks de ficheros se mantienen en el servidor bajo un modelo basado en el arrendamiento.

6. *Caching en el lado del cliente y delegación*: la cache de ficheros, atributos y directorios para la V4 es similar a la de las versiones previas. La información de atributos y directorios se mantiene en la cache durante un tiempo determinado por el cliente. Al final de ese tiempo, el cliente consultará al servidor para ver si los objetos de los sistemas de ficheros han sido actualizados. Para los datos de fichero, el cliente chequea su validez de cache cuando el fichero es abierto. Se envía una consulta al servidor para determinar si se ha cambiado el fichero. Basándose en esta información, el cliente determina si la cache de datos se mantiene o se libera. También, cuando se cierra el fichero, cualquier dato modificado se escribe al servidor. La principal novedad de la versión 4 en el área de caching es la habilidad del servidor para delegar ciertas responsabilidades al cliente. Cuando el servidor concede la delegación de un fichero a un cliente, el cliente tiene aseguradas ciertas semánticas con respecto a la forma de compartir ese fichero con otros clientes. Con la operación OPEN, el servidor puede proporcionarle al cliente una delegación tanto de leer o escribir en el fichero. Si el cliente tiene la delegación de leer, se asegura de que ningún otro cliente va a poder escribir en el fichero durante todo el tiempo que dure la delegación. Si el cliente tiene la delegación de escribir, se asegura que ningún otro cliente va a tener acceso de lectura o escritura sobre ese fichero. Las delegaciones pueden ser retiradas por el servidor. La esencia de una delegación es que permita al cliente ocuparse localmente de operaciones tales como OPEN, CLOSE, LOCK, LOCKUP, READ, WRITE, sin la interacción inmediata del servidor.

Como esta última característica (*Delegation*) es lo único distinto en caching con respecto a las versiones anteriores, voy a exponer cómo actúan las operaciones de caching y prefetching en NFS sin que dependa de las versiones utilizadas. Después volveremos a esa nueva operación y veremos sus implicaciones en nuestro sistema [122].

La versión 4 de NFS puede también beneficiarse del uso de soporte para caching de servidores proxy. Actualmente el caching se hace en el cliente. Los buscadores web también utilizan grandes caches de cliente para páginas web, pero también soporta el uso de cahing de proxy. El protocolo HTTP puede enrutar consultas de páginas web indirectamente a través de un servidor proxy como muestra la siguiente figura. El proxy chequea si la página correspondiente a la petición URL está ya en la cache. Si es así, entonces devuelve la página al cliente sin contactar con el servidor para nada. Una cache de proxy es efectiva si hay páginas que son pedidas por muchos clientes.

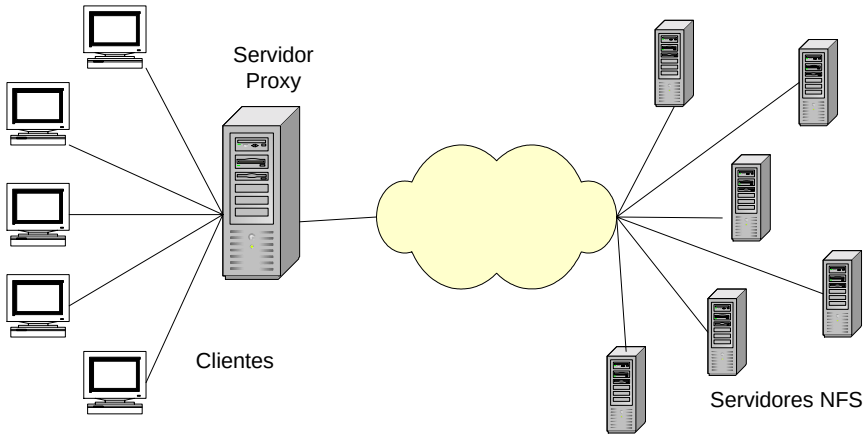


Figura 3.16. Proxy caching

3.4.7. NFS en linux

El NFS de Linux es, principalmente, obra de Rick Sladkey, que escribió el código que corresponde al núcleo y buena parte del código del servidor NFS. Éste último es modificación del servidor *unfsd* que corre en espacio de usuario, escrito originalmente por Mark Shand, y el servidor *hnfs* escrito por Donald Becker.

Ya sabemos que cuando alguien accede a un fichero remoto, el núcleo manda una llamada RPC al programa *nfsd* del nodo remoto. Esta llamada incluye el descriptor de fichero, el nombre del fichero a acceder y los identificadores de usuario y de grupo del demandante. Estos identificadores se usan para chequear permisos de acceso en la máquina remota, con lo que los usuarios de ambas máquinas deberían ser los mismos.

En varias implementaciones de UNIX, las funcionalidades de cliente y servidor NFS se implementan como demonios de nivel de núcleo que se arrancan desde el espacio de usuario al arrancar la máquina. Se trata del programa *nfsd* en el servidor y del programa *biod* en el cliente. Para aumentar el rendimiento, *biod* realiza E/S asíncrona y a veces corren concurrentemente varios servidores de NFS.

La implementación de NFS en Linux es algo diferente: el código de cliente está integrado en la capa de sistema de ficheros virtual (VFS) y no requiere control adicional mediante el programa *biod*. Por otro lado, el código del servidor corre totalmente en el espacio de usuario, por lo que ejecutar varias copias del *nfsd* resulta imposible debido a los problemas de sincronización que originaría.

También hemos visto cómo los volúmenes NFS se montan como los sistemas de ficheros usuales:

```
# mount –t nfs volumen_nfs directorio_local opciones
```

La lista de todas las opciones válidas para mount se encuentra descrita en la página de ayuda *nfs*. Las opciones más interesantes son las siguientes:

- *Rsize=n* y *wsize=n:* Especifican el tamaño de datagrama utilizado por el cliente NFS en las peticiones de lectura y escritura, respectivamente. Por defecto, cada una de ellas vale 1024 octetos, dados los límites del tamaño del datagrama UDP.
- *Timeo=n:* Esta opción establece el tiempo máximo de espera de respuesta a una petición del cliente NFS, en décimas de segundo. Por defecto, este valor es de 0.7 segundos.
- *Actimeo=n:* asigna el mismo valor n a las opciones acregmin, acregmax, acdirmin, acdirmax. Estas opciones establecen el tiempo mínimo y máximo que el sistema va a esperar antes de actualizar desde el servidor los atributos de un fichero o de un directorio.
- *Hard:* marca el montaje del volumen como físico. Es un valor por defecto.
- *Soft:* hace que el montaje sea algo lógico (opuesto al anterior)
- *Intr:* esta opción habilita la posibilidad de que una señal interrumpa una espera por NFS. Es útil para poder abortarla cuando el servidor no responde.

Cuando el cliente realiza una petición al servidor NFS, esperará un tiempo máximo (el que se especifica en la opción *timeo*). Si no hay confirmación tras ese tiempo (que se denomina "de expiración") tiene lugar otra espera, "de expiración menor", en el que la operación se reintenta pero doblando el tiempo de expiración inicial. Tras 60 segundos, se vuelve a la expiración anterior.

Por defecto, la expiración mayor hará que el cliente envíe un mensaje a la consola y empiece de nuevo, con una expiración del doble de tiempo. Este es uno de los aspectos principales de los ajustes del sistema NFS que se muestran en el siguiente capítulo. Potencialmente, esto podría mantenerse eternamente. En este caso se habla de montaje físico o *hard-mount*. La otra variedad, el montaje lógico o *soft-mount*, genera un mensaje de error de E/S al proceso llamante cuando se produce la expiración mayor. El error no se propaga al proceso hasta que hace una llamada a *write()*, por lo que esto, junto con la política de escritura desde la cache, hace que no se sepa realmente si una operación de escritura ha tenido éxito o no, a menos que el volumen esté montado de forma física.

En general, se recomienda el montaje físico salvo en caso de tratarse de información no crítica, como la de servidores de FTP o particiones de noticias. En entornos críticos no debe usarse el montaje lógico a riesgo de perder las conexiones si en un momento se satura o desactiva la red por algún motivo. Una alternativa a usar montajes físicos es aumentar el valor de la opción *timeo*, o bien usar montajes físicos pero permitiendo el envío de señales para interrumpir las esperas en caso de necesidad.

NFS para Linux se está modificando significativamente en los kernels que van saliendo, y el rendimiento se va mejorando. Por ahora, en función del kernel que se tenga, el rendimiento será peor o mejor. En la mayoría de los casos, el rendimiento y la fiabilidad son bastante buenos para el uso diario. Sólo habrá que tener cuidado cuando un servidor está muy cargado. Algunas de las mejoras que se pueden plantar sobre el sistema NFS que tengamos instalado es el ajuste de los parámetros *rsize* y *wsize*. Como ya hemos comentado, estos parámetros controlan los tamaños de buffers para las operaciones de lectura y escritura [123].

El servidor en Linux

Veamos un poco más en detalle la forma de trabajar del servidor (ver diagrama de actividad de la figura 3.12). Como se ve en el diagrama, el servidor recibe y procesa las peticiones. En la actividad Recibe petición, el servidor principalmente identifica el tipo de operación, prepara las estructuras, busca el socket a través del cual realizará esa operación, identifica al cliente, etc.

En la actividad Procesa Petición, realizará las llamadas correspondientes a los procedimientos NFS necesarios

Dos de esos procedimientos son:

- **LOOKUP**: busca el nombre del fichero. Busca un directorio para un nombre específico y devuelve los atributos y el manejador de ficheros para el correspondiente objeto del sistema de ficheros. Como argumentos de entrada necesita el objeto a buscar y, si todo va bien, se obtiene el manejador de fichero para el objeto correspondiente, los atributos para el objeto correspondiente y los atributos para el directorio.

- **READ**: lee de un fichero. Los argumentos que se le pasan son el manejador del fichero del que se va a leer, el offset o posición del fichero en la que se va a empezar a leer, y la cantidad de bytes de datos que se quieren leer. Si todo va bien, se obtienen los atributos del fichero leído, el número de datos devueltos por el *read*, una variable que indica si la lectura del fichero acabó con el final de fichero (muy interesante para nosotros) y los datos en sí leídos

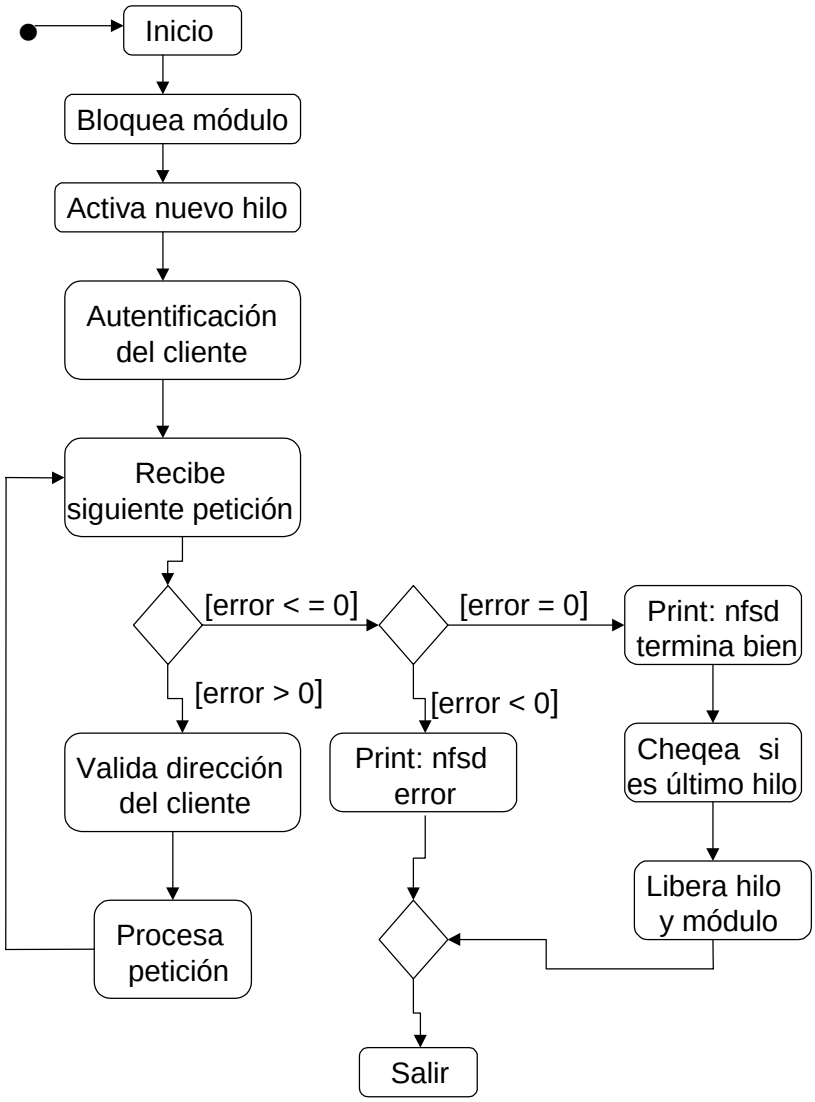


Figura 3.17. Diagrama de actividad del servidor NFS

En LOOKUP tan solo se inicializan las estructuras donde se va a almacenar el manejador y se llama a la operación lookup propiamente dicha, donde se realizan detalladamente todos los pasos necesarios para conseguir el objetivo final.

En READ se reserva el buffer, se comprueba que hay espacio disponible, se llama a la operación read y después se pregunta si se ha leído el final de fichero. Esto se calcula simplemente comparando si el offset especificado más la cantidad de datos a leer igual al tamaño del fichero. Si es así, se pone una variable a TRUE y se acaba la operación, si no es así se sale igualmente.

En el capítulo 5, cuando desarrollemos el diseño del nuevo cliente NFS predictivo profundizaremos un poco más en detalles de implementación tanto del código del cliente como del código del servidor NFS

En la figura 3.13 aparece, de forma resumida, cómo se realiza la comunicación entre el cliente y el servidor cuando el cliente quiere leer tres ficheros cualesquiera f1, f2 y f3. Para leer un fichero antes hay que crear un manejador para ese fichero, y después enviar los READ's necesarios hasta completar la cantidad de bytes que se quieran leer.

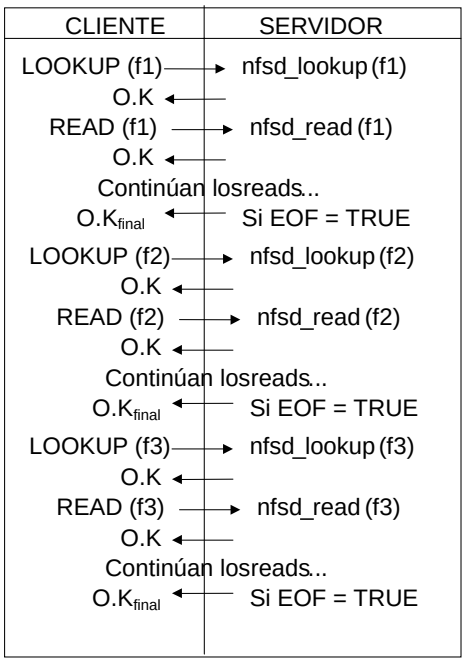


Figura 3.18. Comunicación cliente-servidor para la lectura de ficheros

En READ se reserva el buffer, se comprueba que hay espacio disponible, se llama a la operación read y después se pregunta si se ha leído el final de fichero. Esto se calcula simplemente comparando si el offset especificado más la cantidad de datos a leer igual al tamaño del fichero. Si es así, se pone una variable a TRUE y se acaba la operación, si no es así se sale igualmente.

3.4.8. Variantes de NFS

NFS se ha utilizado como base de otros protocolos. A veces, es más fácil empezar con uno ya existente, probado y extendido, que diseñar e implementar un nuevo protocolo de sistema de ficheros distribuido de la nada. Algunos de los protocolos nacidos a partir de NFS son:

✷ Spritely NFS

Este trabajo se basa en el proyecto *Sprite* de la Universidad de California. Sprite era un sistema operativo distribuido que utilizaba caching en el cliente con un protocolo de consistencia que aseguraba que todos los clientes nunca podrían tener problemas de inconsistencia de datos. *Spritely* NFS tomó prestado de *Sprite* el modelo de caching para proporcionar consistencia de cache a NFS. Aunque este proyecto fue interesante desde el punto de vista de la investigación, sus beneficios eran insuficientes para generar demanda de mercado. Además, la versión 3 de NFS ya cubre algunos de las tareas propuestas en este proyecto [125].

✷ **NQNFS** (***Not Quite NFS***)

Este proyecto tenía los mismos objetivos que el proyecto anterior. Añadió procedimientos a NFS para permitir al servidor mantener estado e indicar los ficheros que estaban abiertos y cacheados en los clientes. Algunas de las características propuestas fueron más tarde adoptadas por la versión 3 de NFS [105]

✷ **TNFS (*Trusted NFS)***

Se desarrolló para soportar los requisitos de acceso del TCSEC (*Trusted Computer System Evaluation Criteria*) del gobierno federal de Estado Unidos para gente con distintos niveles de seguridad. Es un conjunto de requisitos de seguridad para sistemas operativos y software usado en máquinas que acceden a material clasificado. Esta especificación nunca se completó. Incluye un procedimiento ACCESS para permitir acceso de ficheros fiable en el instante de apertura del fichero [105]

✷ **NASD NFS**

Este es un proyecto del laboratorio de datos paralelos de la universidad Carnegie-Mellon. NASD (Network-Attached Secure Disks) intenta proporcionar seguridad, alto ancho de banda, baja latencia de acceso a los datos soportando un camino directo desde el dispositivo de almacenamiento al cliente.

Con una configuración de servidor convencional, el cliente envía una petición de acceso a datos al servidor, éste localiza los datos en uno de sus discos y utiliza el comando adecuado para leer los datos. Los datos pedidos se copian del dispositivo a la memoria del servidor, y después al cliente. La intervención del servidor entre el almacenamiento y envío de datos entre el cliente y el disco incrementa el tiempo de transferencia de los datos.

En este proyecto se observa que el servidor no necesita ningún acceso a los datos que están siendo leídos o escritos al dispositivo. El papel del servidor se limita a situar los datos en un dispositivo concreto, autentificar la petición del cliente, y chequear la autorización del cliente para acceder a los datos.

En una configuración NASD, los ficheros se transfieren directamente entre el cliente y el dispositivo. El acceso directo a los datos reduce la latencia de datos. En lugar de inventar un nuevo protocolo de acceso a ficheros, el equipo del NASD decidió hacer algunas modificaciones a los protocolos NFS y AFS. Un cliente usa el servidor, o el gestor de ficheros, para localizar un fichero y obtener la autorización de acceso a ese fichero. Para leer los atributos de fichero o para transferencia de datos del fichero, el cliente se comunica directamente con el dispositivo, como se puede ver en la siguiente figura [105].

Los discos unidos a la red dan al cliente acceso directo a los dispositivos de almacenamiento. El servidor participa en las operaciones de directorios y control de accesos. Los atributos de fichero y los datos leídos y escritos se transfieren directamente entre el cliente y los dispositivos de almacenamiento mediante un conmutador.

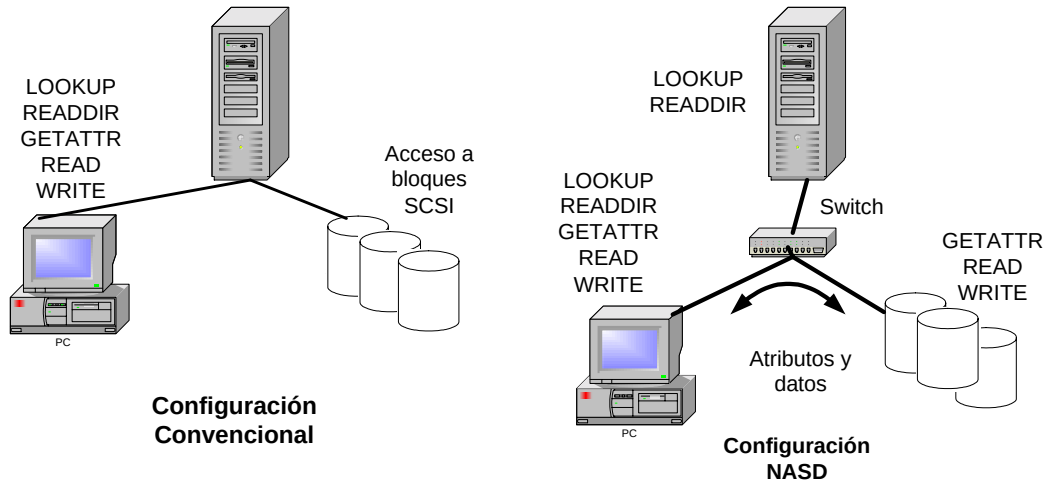


Figura 3.19. Configuración convencional y configuración NASD

Se pretende que el rendimiento del almacenamiento escalable dependa de la eliminación del papel del servidor como router en la red. En lugar de esto, se quiere explotar la habilidad de un dispositivo para inyectar paquetes directamente en la red de los clientes. Con un almacenamiento unido a la red efectivo, la división de datos sobre múltiples dispositivos puede escalar el ancho de banda de almacenamiento [126].

Con este tipo de almacenamiento se reduce la capacidad malgastada, el tiempo para desplegar un nuevo almacenamiento, y las inconveniencias de los backups. También se simplifica la gestión, se incrementa la disponibilidad de los datos y se habilita el compartir datos entre clientes. Los nuevos sistemas NAS se pueden instalar y configurar sin interrumpir la ejecución de las máquinas cliente. Y usuarios NAS de diferentes máquinas pueden ver y compartir la misma colección de ficheros sin un especial esfuerzo [127]

✷ El proyecto Trapeze

Este proyecto de la universidad de Duke tiene como objetivo desarrollar nuevas técnicas para comunicaciones de alta velocidad y acceso rápido a los datos almacenados en clusters de estaciones de trabajo, y demostrar su uso en prototipos de sistemas UNIX mejorados. Se pretende mejorar el soporte del sistema para compartir información de forma fiable y eficiente en las redes de estaciones de trabajo de la próxima generación, particularmente en entornos de carga de trabajo mezclada, en la que el cluster se usa para soportar actividades individuales y compartir información entre grupos, además también par computación a gran escala. La parte más interesante de este proyecto para nosotros es el sub-proyecto Slice, que explora técnicas para construir sistemas de almacenamiento masivo de datos unificados conectados por una red de alta velocidad. Un sistema de almacenamiento en red Slice está configurado como una combinación de módulos del servidor que gestiona funciones específicas del sistema de ficheros, como la gestión de directorios, el almacenamiento de los bloques, el almacenamiento eficiente

de los ficheros pequeños y el caching de red. Slice está diseñado par ser compatibles con clientes NFS estándares, usando un traductor de paquetes a nivel de red para mediar entre cada cliente y un array de servidores presentando una visión de sistemas de ficheros unificado. Más información detallada sobre este proyecto, distintas publicaciones así como la última versión disponible de Slice para instalar se puede encontrar en http://www.cs.duke.edu/ari/trapeze/. Como se puede ver en la siguiente figura, Slice está por encima de NFS. Este módulo intercepta las operaciones de lectura y escritura en los vnodos de los ficheros y los redirecciona a un array de servidores de E/S de bloques.

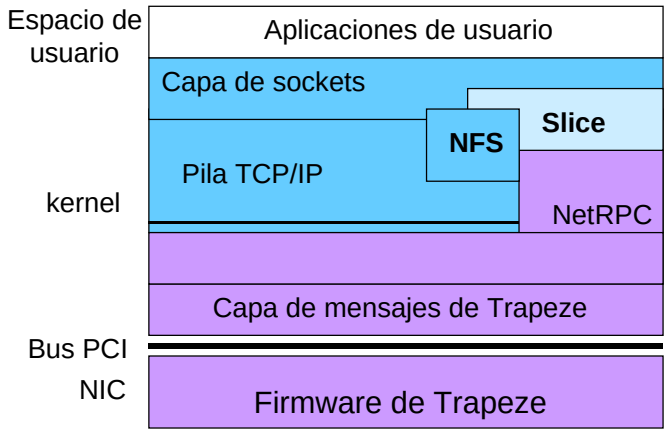


Figura 3.20. Una vista del prototipo Slice/Trapeze

El servicio del E/S de bloques Slice se construye de una colección de PCs, cada uno con un puñado de discos y una interfaz de red de alta velocidad. Se le llama a esta aproximación almacenamiento en red PCAD (*PC-attached Disks)*, indicando una aproximación intermedia entre los discos unidos a la red (NASD) y los servidores de ficheros convencionales o SAD (*Server-Attached disks*) [128].

3.4.9. OTROS SISTEMAS DE FICHEROS DISTRIBUIDOS

★ AFS (*Andrew File System*)

Se desarrolló en la universidad Carnegie-Mellon para proporcionar un sistema de ficheros distribuido que proporcionase acceso a ficheros a un gran número de clientes a través de un campus de una universidad. AFS compartía algunas de las consideraciones de diseño de NFS: soportar semánticas de sistemas de ficheros UNIX y hacerlo dentro del kernel del cliente para que las aplicaciones no tengan que ser modificadas. AFS es notable por algunas características de diseño adicionales. Los clientes cachean los datos de ficheros de forma agresiva y fiable en el servidor para mantener consistencia de cache, minimizando la interacción cliente-servidor. Soporta un espacio de nombres independiente de la posición y compartido con un directorio raíz único. Cada cliente tiene una vista consistente de un espacio de nombres compartido que proporciona acceso a los datos independiente de la posición que pueden ser movido y replicado. Se

usa autentificación kerberos para los usuarios, y el acceso a los ficheros se controla con listas de control de accesos.

AFS evolucionó al DFS (*Distributed File System*) componente del entorno de computación distribuida de OSF. Además de la escalabilidad, los diseñadores de AFS especificaron algunos importantes objetivos. AFS debía ser compatible UNIX, debe proporcionar un espacio de nombres uniforme e independiente de la posición para los ficheros compartidos. Los usuarios pueden acceder a sus ficheros desde cualquier nodo cliente en la red, y los ficheros se pueden mover a otra nueva posición sin alterar el sistema. Debe ser tolerante a fallos, así el fallo de un único servidor o componente de red no convierte el sistema en no disponible. Los fallos deben aislarse muy cerca del punto de fallo y debe proporcionar seguridad [100]

AFS se ha desarrollado en gran cantidad de plataformas y está disponible en muchas versiones de UNIX. Las implementaciones están disponibles también para clientes no UNIX, tales como DOS o Macintosh. AFS también forma los fundamentos básicos del sistema de ficheros Coda, que permite a clientes móviles a datos cacheados estando desconectados de la red [105]. Veamos un poco más de cada uno de estos sistemas de ficheros

AFS, aunque es similar a los sistemas de ficheros UNIX existentes, conlleva bastantes nuevos conceptos y definiciones, como por ejemplo, el concepto de *célula*. Las células son directorios en el espacio de ficheros AFS por debajo del directorio raíz. AFS usa también una estructura jerárquica de ficheros (un árbol). La raíz se llama /afs, y todos los subdirectorios de este directorio raíz se llaman células. Otro concepto importante es el de *volumen.* Un volumen es la unidad básica de almacenamiento AFS y representa una subsección de una partición de disco. Las células se configuran para asignar los directorios home de los usuarios a un volumen. Los volúmenes son parecidos a los directorios UNIX y contienen ficheros, directorios y path a otros volúmenes.

Los servidores proporcionan acceso a los volúmenes AFS y a las bases de datos que controlan los accesos a esos volúmenes. No hay que saber qué máquina es responsable de un volumen dado. AFS automáticamente encuentra el volumen que contiene el directorio o fichero al que se quiere acceder. Los clientes ejecutan el software de cliente AFS que conecta con los servidores. Este enlace proporciona acceso a los volúmenes que se necesitan.

AFS controla la congestión en la red y la sobrecarga del servidor segmentando la red en un número de clusters independientes. A diferencia de NFS, AFS usa servidores dedicados. Cada máquina es o bien un cliente o un servidor pero no ambos.

Se han publicado medidas del rendimiento relativo de estos sistemas de ficheros. En [101] se compara el rendimiento de NFS y AFS usando idénticas configuraciones hardware. Los resultados muestran que para un único servidor, NFS es más rápido para menos de 15 clientes, pero que se deteriora rápidamente para un número mayor. Estos dos sistemas han evolucionado mucho desde estos estudios, pero los factores que afectan a la escalabilidad o han cambiado demasiado.

Como ya hemos comentado un sistema de ficheros muy similar a AFS es Coda. Se ha desarrollado en la universidad de Carnegie Mellon, por lo tanto es un sistema experimental y no un producto comercial. La principal característica que los distingue del resto es la computación móvil. Es la implementación de AFS para PCs

La forma de compartir los datos en Coda es igual a la de AFS, está soportada por un único gran subárbol del sistema de ficheros local de cada máquina. Este subárbol es idéntico en todas las máquinas. Mucha más información detallada sobre este sistema de ficheros se puede encontrar en [129]

★ xFS

En [130] se describe un sistema de ficheros en red sin servidor, los principios básicos, objetivos y propósitos, y se presenta el prototipo del sistema de ficheros xFS como implementación de esta filosofía. Este sistema sin servidor se desarrolló dentro del marco de trabajo del proyecto NOW [131]

Con esta aproximación se distribuyen todos los aspectos del servicio e ficheros (incluyendo almacenamiento de datos, caching de datos y control) a través de todas las máquinas en el sistema. Se destacan los siguientes beneficios:

- Rendimiento mejorado: se elimina la idea de sistema centralizado eliminando los embotellamientos en el servidor y se utilizan los recursos agregados de un gran número de máquinas. Se combinan las caches de todas las máquinas formando una gran cache cooperativa. Cualquier máquina en el sistema puede controlar cualquier fichero
- Disponibilidad mejorada: cuando una máquina falla, el resto de máquinas se encargan de sus responsabilidades. Para conseguir esto hay dos aspectos de diseño muy importantes que son el uso de almacenamiento en disco redundante y el almacenamiento basado en logs, y la independencia de posición para delegar los deberes de las máquinas que fallan al resto
- Escalabilidad mejorada: un objetivo básico es eliminar todos los embotellamientos centralizados del sistema. El rendimiento y la disponibilidad de un sistema sin servidor mejoran conforme más CPUs, DRAM o discos se añadan para proporcionar más recursos que el sistema pueda explotar. Esta arquitectura debería trabajar bien con decenas o cientos de máquinas

El prototipo incluía cache cooperativa, almacenamiento sin servidor, y gestión sin servidor para realizar su objetivo de independencia de posición. En un sistema centralizado típico, el servidor realiza cuatro tareas principales:

- El servidor almacena todos los bloques de datos del sistema en sus discos locales
- El servidor maneja metadatos de posición de disco para indicar donde se guarda cada bloque de datos en disco
- El servidor mantiene una cache central de bloques de datos en su memoria para satisfacer algunos fallos de cliente sin acceder a sus discos
- El servidor gestiona metadatos de consistencia de cache que listan qué clientes en el sistema están cacheando cada bloque. Usa estos metadatos para invalidar datos pasados en caches de clientes

El sistema xFS realiza estas mismas tareas, pero distribuye este trabajo sobre todas las máquinas en el sistema. XFS sustituye la cache del servidor por el caching cooperativo que envía datos a las caches de los clientes bajo el control de los *managers*. De forma similar proporciona almacenamiento en disco escalable, xFS usa división en la red basada en log con *cleaners* distribuidos. Finalmente, para proporcionar control escalable de los metadatos del disco y consistencia de cahe, xFS usa técnicas de gestión sin servidor. Cuatro tipos de entidades (los clientes, los servidores de almacenamiento, los *cleaners* y los *managers*) cooperan para proporcionar servicio de ficheros.

En la tesis referenciada se comparan sobre el mismo hardware xFS, AFS y NFS. Se utiliza NFS como el sistema base por razones prácticas (NFS es más maduro, ampliamente disponible y bien ajustado, permitiendo una fácil comparación y un buen marco de referencia) pero su tiene limitaciones con respecto a la escalabilidad. AFS consigue mejor escalabilidad en comparación con NFS. Los resultados de rendimiento para xFS no fueron tan buenos como se esperaban por lo que este trabajo siguió mejorándose y refinándose. El estudio detallado aparece también en [132]

Siguiendo con el espíritu de xFs, hay otros trabajos interesantes como el sistema de ficheros Frangipani [133] que pretenden proprocionar usuarios con sistemas de ficheros escalables y distribuidos cada vez más potentes. En casi todos estos nuevos sistemas de fichero siempre suele aparecer NFS por encima o por debajo de lo que se pretende implementar, aportando su ya consolidada arquitectura y su disponibilidad y portabilidad en cualquier plataforma.

★ Samba

Samba es un conjunto de aplicaciones UNIX que utilizan el protocolo SMB (*Server Message Block).* Muchos sistemas operativos, incluyendo Windows y OS/2, usan Samba para realizar la comunicación cliente-servidor. Una máquina habilitada Samba puede hacerse pasar por un servidor y ofrecer los siguientes servicios:

- Compartir uno o más sistemas de ficheros
- Compartir impresoras
- Asisitir a los clientes en tareas de *browsing*
- Autentificar clientes en dominios Windows
- Proporcionar resolución de servidor de nombres

Esto proyecto aparece en 1991 de la mano de Andrew Tridgell. Microsoft también ha colaborado con su definición de SMB y el CIFS (*Common internet File System*). Este es un protocolo renombrado por Microsoft para futuras versiones de SMB que se utilizarán en los productos Windows. Utiliza un modelo de espacio de nombres simple. Es un protocolo orientado a sesión [105].

Actualmente Samba gira alrededor de un par de demonios que proporcionan los recursos compartidos o *shares* a los clientes de la red. Estos demonios son el:

- **smbd**: permite compartir fichero se e impresoras en red y proporciona autentificación y autorización para los clientes SMB
- **nmbd**: proporciona servicios de nombres en Internet (WINS) y asiste con el browsing

Samba se mantiene por un grupo de voluntarios bajo la supervisión de Andrew Tridgell. Es considerado un software abierto (*Open Source Software*) y se distribuye bajo la licencia GNU. Samba puede ayudar a que coexistan máquinas Unix y máquinas Windows en la misma red. Samba aparece en UNIX como un conjunto de programas demonio y se puede configurar desde un único ficheeo de propiedades (smb.conf).

Samba está disponible en Internet en www.samba.org, y en [134] aparece definidos en detalle todo los conceptos relacionados con Samba, así como la instalación en sistemas UNIX, la configuración de los clientes Windows, y mucho más. También podemos encontrar información adicional en

En [135] se puede encontrar una comparativa de rendimiento de distintos sistemas de ficheros en red (NFS, Samba y Coda). Aquí se puede comprobar que el uso de NFS no es un capricho ni debido a motivos históricos. Las prestaciones que se han llegado a conseguir con las últimas versiones son envidiables. Además de los buenos resultados, conserva todas las ventajas del sistema ext2, cosa que con Samba no ocurre aunque sí con Coda.

3.4.10. SISTEMAS DE FICHEROS PARALELOS

Existe otro tipo de sistemas de ficheros orientados también a aumentar el rendimiento de las operaciones de E/S y evitar problemas de embotellamientos, especialmente con grandes conjuntos de datos, son los llamados sistemas de ficheros paralelos.

Un buen ejemplo es PVFS (*Paralell Virtual File System*). El objetivo de este proyecto es explorar el diseño, implementación y uso potencial de la E/S paralela como parte de las actividades del laboratorio de investigación de arquitecturas paralelas de la universidad de Clemson. Actualmente está centrado en la E/S paralela en cluster de estaciones de trabajo dedicadas, pilas de PCs y en particular en estaciones de trabajo de Beowulf. Está implementado y disponible (en forma experimental) para otros grupos de investigación. Proporciona lo siguiente:

- Compatibilidad con binarios existentes
- Facilidad de instalación
- División de los ficheros a través de los nodos controlada por el usuario
- Interfases múltiples, incluyendo una interfaz MPI-IO via ROMIO [136]

3.4.11.TRABAJOS DE INVESTIGACIÓN BASADOS EN NFS

Aunque ya está cerca de cumplir los veinte, NFS sigue siendo objeto de continuas modificaciones y adaptaciones a las nuevas tecnologías, dispositivos, máquinas y medios de comunicación. Y esto es así, porque hay que mantener al día un sistema de fichero considerado como un estándar y etiquetado como el más utilizado actualmente debido a sus características básicas que ya hemos resaltado.

Esto queda demostrado por la sucesión de nuevas versiones, cada una de las cuales actualiza el sistema de ficheros y lo acerca más a las necesidades actuales que imponen las nuevas tendencias. Sun Microsystems, su casa comercial, sigue

manteniendo a una gran cantidad de investigadores centrados en esta tema y además, y más importante, se está abriendo y trabajando en unión con universidades y otras entidades para conseguir incorporar las últimas novedades y para convertir NFS en el sistema de ficheros en red universal. Pero además de este NFS "cerrado" que se mete en Solaris y se instala en las máquinas correspondientes, hay que hablar del otro NFS "abierto", el que viene en las distribuciones de Linux, y el que se muestra con su código fuente a todos los investigadores. La necesidad de mejorar NFS en Linux está más que demostrada, y esto queda reflejado en los numerosos e interesantes trabajos, proyectos y tesis que siguen desarrollándose teniendo como elemento principal el sistema de ficheros NFS.

Por eso no quería acabar este capítulo sin resaltar algunos de estos trabajos, que como el que se presenta en esta tesis doctoral quieren mejorar algun aspecto concreto para conseguir mejores prestaciones, sea cual sea la base de los experimentos, los bancos de pruebas utilizados o la arquitectura donde se desarrollan.

Existen dos trabajos muy interesantes en el Instituto de Investigación de Kanpur (India). Uno orientado a extender NFS para conseguir acceso transparente a dispositivos remotos en el servidor, para que el cliente pueda acceder no solo a distintos ficheros de datos, sino también a cualquier tipo de dispositivo que exista en el servidor [137]. Esto vimos que era una diferencia importante con Samba, pues bien ese trabajo va orientado a eliminarla. Otro de los trabajos está orientado a diseñar e implementar un gateway de FTP a NFS totalmente portable, que proporcione acceso a ficheros en servidores FTP remotos haciéndolos aparecer como parte del árbol de sistema de ficheros local. Simplemente montando el servidor en un directorio local, el usuario será capaz de acceder a los ficheros en servidores FTP como si fueran locales [138].

3.5.SISTEMA OPERATIVO DE LA PLATAFORMA

El sistema operativo utilizado en el cluster ha sido Linux, ya que se plantea como el sistema operativo ideal para diseñar supercomputadores paralelos de bajo coste.

3.5.1.Fundamentos y características principales de Linux

La vida de linux hasta el momento, aunque breve, ha sido bastante intensa. No podía ser de otra manera para un sistema que, contra todo pronóstico, ha conseguido desarrollarse, sobrevivir y ganar aceptación hasta convertirse en el fenómeno que es hoy en día. La historia de Linux es un buen ejemplo para reflejar los profundos cambios experimentados en el ámbito informático. El desarrollo de linux no hubiera sido posible si n el trabajo ni el timpo dedicados al sistema por parte de miles de programadores que, antes o después, aceptaron la invitación para colaborar con el proyecto linux a través de la red, y capitaneados por Linus Torvalds. La evolución del sistema ha sido y sigue siendo constante. A lo largo de esta evolución, la cantidad de colaboradores del proyecto Linux ha crecido de forma considerable. Además, se ha producido un espectacular incremento del número de usuarios del sistema, debido principalmente a dos fenómenos sociales: el auge de la filosofía del software libre (o de código abierto), lo que implica que todo el mundo tiene derecho a usar, copiar y modificar los programas sin ningún costo, y la popularidad de Internet [139]. El software Linux se desarrolla bajo condiciones distribuidas y abiertas, cualquiera puede verse involucrado si lo desea.

Pero Linux también tiene sus inconvenientes. Un sistema de programación tipo Unix, con sus comandos y configuraciones es difícil de seguir, la documentación brilla por su ausencia o es difícil de entender, por lo que está bastante lejos de ser fácil de usar [140].

Algunas de las características principales de Linux son las siguientes:

- *Multitarea*: todos los procesos se ejecutan de forma independiente unos de otros
- *Acceso multiusuario*: permite que un número variable de usuarios trabaje en el sistema al mismo tiempo
- *Multiproceso*: si corre en arquitecturas multiprocesador, el sistema puede distribuir algunas aplicaciones a través de los procesadores
- *Independiente de la arquitectura*: corre en distintas plataformas hardware
- *Demanda de carga ejecutable*: sólo las partes de un programa que se está ejecutando se carga en memoria. Se utiliza esto en lugar del intercambio de páginas, donde la memoria completa para un proceso se escribe a disco, lo que hace que se utilice la memoria de forma más eficiente.
- *Cache dinámica*: ajusta de forma dinámica el tamaño de la memoria cache en uso en función del uso actual de la memoria.
- *Librerías compartidas*: hay un número determinado de librerías estándar usadas por más de un proceso al mismo tiempo. Así se carga en memoria sólo una vez el código de esas librerías, en lugar de una vez por proceso.
- *Varios formatos para ficheros ejecutables*: se pueden ejecutar bajo Linux programas que corren en distintos entornos.
- *Modo protegido de memoria*: usa mecanismos de protección de la memoria del procesador para prevenir que los procesos accedan a la memoria asignada al núcleo o a otros sistemas
- *Sistemas de ficheros diferentes*: soporta una amplia variedad de sistemas de ficheros, el más común ext2, y otros como vfat, iso, nfs, ups, hpfs, etc.
- *Soporta PPP, SLIP y TCP/IP*: Linux puede integrarse en redes locales UNIX. En principio se pueden usar todos los servicios de red.

3.5.2. El núcleo (kernel) de Linux

La primera versión del núcleo estuvo disponible en noviembre del 91, y aunque Linus Torvalds haya intentado mantener el núcleo organizado en una forma fácil de seguir y aclarando constantemente las resacas de versiones anteriores, el núcleo de Linux no es un buen modelo de programación estructurada. Hay números mágicos en lugar de declaraciones constantes en ficheros cabecera, funciones expandidas en línea en lugar de llamadas a funciones, instrucción goto, instrucciones en ensamblador en lugar de en código C, y otras muchas características poco elegantes. Pero muchas de ellas fueron incluidas a posta, ya que muchas partes del núcleo están bastante limitadas en tiempo y el código del programa se ha optimizado para un buen comportamiento en tiempo de ejecución, en lugar de para facilidad de lectura.

Linux se implementó en la clásica arquitectura de núcleo monolítica. Pero no es una colección caótica de código de programa. Muchos componentes del núcleo son accesibles sólo mediante interfaces bien definidos. Un buen ejemplo es VFS, que representa una interfaz abstracta a todas las operaciones orientadas a ficheros con las que nos tenido que tratar. El código completo en la versión 2.4 para la arquitectura Intel consiste de más de 470.000 líneas de código C y de más de 800 líneas en ensamblador

El núcleo es un proporcionador de servicios desde el punto de vista de un proceso en ejecución. El punto de vista interno de un sistema Linux en ejecución es otra cosa. Hay que distinguir proceso de hilo. Un hilo es una clase de hebra o rama en el curso de un programa que puede ser procesado en paralelo con otros hilos. Los hilos se alternan en la CPU de la misma forma que lo hacen los procesos. Son más rápidos de crear ya que no requieren un espacio de direcciones propio [140]. La primera etapa que un programador debe conocer cuando esté en el espacio del kernel es la de crear o bifurcar un proceso. Para crear un nuevo hilo se llamará a la función kernel_thread.

Se podría contar mucho más, pero no es el objeto de este trabajo exponer las profundidades del núcleo en detalle, sino dar algunos conceptos básicos para que se pueda entender con facilidad el trabajo realizado. En los capítulos siguientes se mencionarán algunos detalles más.

3.5.3. Distribución y versión utilizadas

Existen distintas opiniones sobre la calidad de las distintas distribuciones. Las más usadas son *Red Hat*, *Debian* y *Slackware*. El elegir entre una u otra es cuestión de gustos. Nosotros hemos utilizado la distribución *Red Hat* [141], versión 7.1, y el kernel 2.4.8. Además utilizamos la versión 3 de NFS.

3.6.RED DE INTERCONEXIÓN UTILIZADA

3.6.1. Interfaces de red utilizados en clusters

La interfaz de red es un elemento crítico en un cluster. Por un lado es el principal cuello de botella de la mayor parte de los clusters. Por otro lado, no siempre está soportado por el kernel. La Ethernet tradicional funciona razonablemente bien, pero tiene una velocidad baja. Además, el alto número de colisiones hace que escale mal. Otra opción es la Fast Ethernet, que tienen una tasa de transferencia mayor, habrá menos colisiones, pero seguirá habiendo problemas de escalabilidad. A partir de ocho nodos empiezan los problemas y a partir de 16, la red se satura.

Las tarjetas Myricom son actualmente las mejores del mercado para clusters con Linux, aunque la tecnología sea un blanco móvil y la velocidad para quedarse obsoletas es bastante elevada. Alcanzan velocidades de transmisión de 1,28 Gbps en cada dirección y una latencia entre 5 y 18 microsegundos, un orden de magnitud más rápida que Fast Ethernet. Estos dos parámetros permiten disminuir las colisiones hasta en dos órdenes de magnitud, por lo que estas redes escalan hasta varios miles de nodos. Existen versiones en cobre y fibra óptica que más adelante detallaremos. Podemos conectar las máquinas con Myricom (que es nuestra plataforma) tanto en conexiones punto a punto, como con Hubs y switches. Esta opción es más cara, pero no presenta colisiones.

Las tarjetas Giganet son otra opción, son tecnológicamente inferiores a Myricom, pero están apoyadas por Intel. No son válidas para WAN y usan su propia familia de protocolos (no usa IP).

Por otro lado están los supercomputadores clase mosix se basan en un conjunto de parches al kernel, que se instalan de forma automática mediante scripts y que nos permiten montar un cluster que a todos los efectos se va a comportar como un sistema multiprocesador. Ofrecen la ventaja de un espacio de procesos común en el que los procesos migran de un nodo a otro según la carga de las máquinas y la memoria libre, empleando para ello un algoritmo de planificación adaptativo. La columna vertebral de este proceso se denomina migración, por la que un proceso pasa de un nodo a otro de forma transparente al usuario. Las migraciones de un nodo a otro se realizan automáticamente, siguiendo un criterio de máximo uso de los recursos disponibles. El protocolo de migración es descentralizado, por lo que la caída de un nodo no compromete el cluster completo. Mosix es transparente al usuario. El programador que desarrolla una aplicación para Mosix y el usuario que la ejecuta no ven el cluster subyacente. Emplear un supercomputador Mosix puede ser útil para algunas tareas, pero también puede ser un completo error para otras. Las aplicaciones lentar por realizar un uso intensivo de lecturas y escrituras de disco no obtienen ninguna mejora por usar Mosix. Existe un sistema de ficheros de Mosix, pero está en fase beta y tampoco gana nada [53, 74, 75]

3.6.2. Descripción del cluster

El cluster consta de ocho estaciones (myrinet1, myrinet2, ..., myrinet8) y dos conmutadores, como se muestra en la figura

Las estaciones son Pentium II a 350 MHz, con 728 Mbytes de RAM. Cada máquina está equipada con una tarjeta Myrinet, pero no todas tienen la misma:

- Cuatro tienen una tarjeta Myricom M2F-PCI32c (puerto LAN) con procesador LANAI4.3 y 1024 Kbytes de memoria
- Dos máquinas tienen una Myricom M2F-PCI32b (puerto LAN) con procesador LANAI4.1 y 512 Kbytes de memoria
- Una tiene una Myricom M2M-PCI32c (puerto SAN) con procesador LANAI4.3 y 1024 Kbytes de memoria
- Una tiene una Myricom M2M-PCI32b (puerto SAN) con procesador LANAI4.1 y 512 Kbytes de memoria

Los dos conmutadores Myrinet del cluster tienen ocho puertos. En uno de ellos (M2F-SW8) los ocho puertos son puertos LAN, mientras que en el otro (M2FM-SW8) hay cuatro puertos LAN y cuatro puertos SAN. Un puerto de una tarjeta de una máquina del cluster debe conectarse a un puerto del conmutador del mismo tipo (LAN a LAN, SAN a SAN) mediante cables LAN o SAN específicos. Así, las estaciones y conmutadores pueden intercambiarse para formar diferentes topologías, como veremos en el capítulo de resultados, pero siempre evitando la incompatibilidad entre puertos LAN y SAN. En nuestro cluster, inicialmente, cada conmutador está conectado directamente a cuatro estaciones, y los dos conmutadores están conectados por un enlace simple.

Cada terminal de la plataforma dispone también de una tarjeta Fast-Ethernet para bus PCI. Estas tarjetas se encuentran conectadas a un concentrador conectado también a la red general. Esto permite el acceso al *cluster* desde terminales exteriores.

En la configuración software del cluster, la myrinet será el servidor NFS, y el resto serán los de clientes. Entre los clientes se va a paralelizar el algoritmo de compresión de vídeo MPEG-2, por lo que existe una necesidad real de velocidad en la transmisión, que quedará satisfecha con la utilización de esta red de gigabits por segundo. En el capítulo siguiente plantearemos los principales problemas detectados en este tipo de entornos para aplicaciones multimedia.

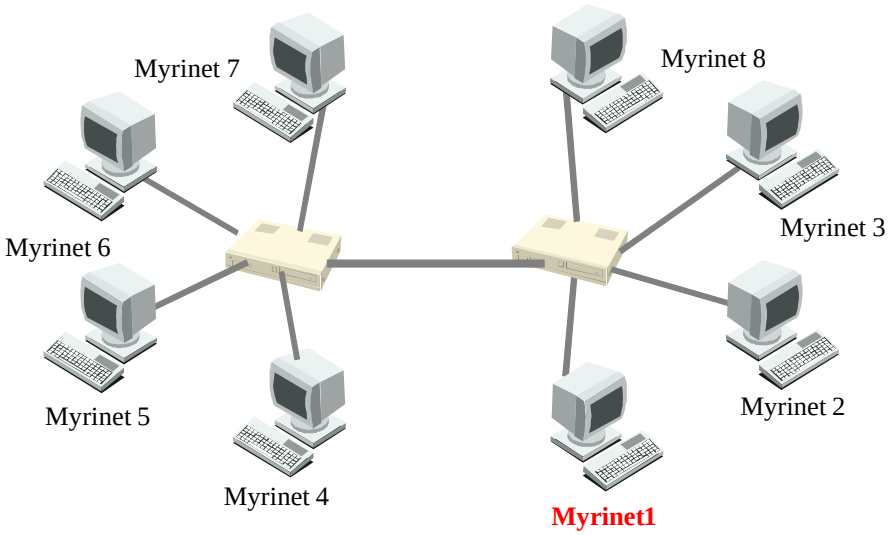


Figura 3.21. Cluster myrinet del RAAP

CAPÍTULO 4

EVALUACIÓN INICIAL

4.1. INTRODUCCIÓN

Como ya se ha comentado en el capítulo 3, NFS es el sistema de ficheros estándar, más utilizado en todo el mundo. Bien es verdad que desde que apareció en los años ochenta han ido surgiendo distintas propuestas alternativas para intentar conseguir nuevos objetivos o para intentar hacer lo que NFS, en su versión inicial, dejaba al descubierto.

También hemos comentado ya, cual ha sido la motivación de este trabajo, evaluar los clusters y sus aspectos fundamentales, y, mejorar y optimizar un elemento clave en los cluster de alta velocidad, como es la entrada/salida. Pero además, soportando tráfico multimedia, con todo lo que ello conlleva de requisitos computacionales, ancho de banda, capacidad de almacenamiento, etc., y utilizados como plataforma ideal para el procesamiento paralelo.

Uno de los principales elementos a tener en cuenta es el sistema de ficheros en red utilizado. Esa capa software que permite a las distintas computadoras conectadas a la red compartir la información almacenada en una única ubicación y que permite a todas las máquinas acceder a esa fuente virtual de información, como si de algo particular se tratase.

Vamos, por tanto, a evaluar el comportamiento de NFS realizando un estudio exhaustivo y detallado donde se variarán todos los factores que, en mayor o menor parte, pueden afectar a los resultados finales. Lo analizaremos hasta sacar conclusiones claras que nos permitan mejorar todo lo posible el sistema y llegar a una propuesta formal de mejora.

Vamos a analizar cuáles serán las pruebas que se van a hacer, los parámetros NFS de interés, las posibles implicaciones de la propia red y los distintos patrones de carga que se podrían presentar con el tráfico multimedia. Con todo esto pretendemos detectar y exponer los posibles problemas que pueda presentar este sistema de ficheros.

Iniciamos esta exposición de resultados trasladándonos a los orígenes. Todo empezó, precisamente, en la capa superior de la arquitectura que estamos presentando y evaluando hoy. Todo empezó con la paralelización de las aplicaciones multimedia. Primero, analizamos este tipo de tráfico y su repercusión en sistemas de multicomputación [143, 144]. Después, descubrimos el enorme potencial y uso de los clusters como plataformas ideales de investigación del procesamiento paralelo. Con ésto, empezamos a desarrollar y evaluar técnicas de división de la carga entre los distintos procesadores de un cluster [145]. Aquí fue donde empezaron a aparecer los primeros problemas de desfases de tiempos entre procesadores que realizaban las mismas tareas. Y aquí fue donde empezamos a estudiar la importancia de compartir la información multimedia, y de utilizar sistemas de ficheros que fuesen capaces de solventar esta necesidad de compartir, permitiendo la transferencia de datos a través de la red, en una arquitectura cliente-servidor, de forma totalmente transparente al usuario.

Empezamos utilizando NFS en el cluster ATM de la HKUST (*Hong-Kong University of Science and Technology*) y lo seguimos utilizando en el cluster Myrinet de la UCLM. Empezamos ajustando y probando ese sistema de ficheros [146] para comprobar posibles mejoras en su rendimiento. Desde entonces hasta ahora, nos planteamos una propuesta seria de mejorar el sistema de fichero en red NFS. Antes, teníamos que conocerlo perfectamente y probar cómo le afectaban otros cambios en el cluster, como la topología de red utilizada [147] o los patrones de carga representativos de distintas aplicaciones multimedia [148]. Con todo esto, presentamos los principales problemas detectados, debidos precisamente a los efectos de ciertos factores como el *caching*, implementados en NFS, y a la sobrecarga del servidor, ya que la red no iba a plantear ningún problema. De toda esta evaluación [149] va a salir una propuesta clara y precisa de mejora que detallaremos en los siguientes capítulos.

También hay que destacar, que la aplicación multimedia utilizada, el codificador de vídeo MPEG-2, también ha sido el causante de otros interesantes trabajos en nuestro departamento[150, 151].

4.2. PUNTO DE PARTIDA

Claro está que no pretendemos contar en este capítulo todos y cada uno de los experimentos realizados desde el principio (sería una tesis demasiado larga), de ahí que en el apartado anterior se hayan referenciado las publicaciones aparecidas con estos trabajos para que se puedan consultar en caso de interés. Para este capítulo hemos de fijar un punto de partida, y para ello hemos elegido un punto de inflexión clave en nuestro trabajo, como es el estudio y optimización del sistema de ficheros en red. Este va a ser el estado inicial desde donde partimos, con el objetivo de exponer claramente los principales problemas detectado y, ya en el siguiente capítulo, desarrollar detenidamente la mejora implementada.

Antes de iniciar las pruebas expuestas en este capítulo, centradas en la evaluación de NFS, realizamos una gran cantidad de experimentos variando la interfaz paralela (MPICH, LAM, BIP), la longitud de la secuencia de vídeo, y otros aspectos software [146, 142]. Tras estas pruebas, nos dimos cuenta que había un problema presente en la lectura de la información, así que nos planteamos la actualización exhaustiva del sistema (sistema operativo, drivers para la correcta utilización de

Myrinet, versión de MPICH, etc.). Después de estas actualizaciones previas, nos plantearemos el banco de pruebas que se va a ejecutar sobre una topología de red básica, que también cambiaremos para encontrar la posible relación entre los primeros resultados obtenidos y la propia red. Una vez realizadas estas pruebas y expuestos los resultados, realizaremos unas segundas pruebas variando también la forma en que se realizan las operaciones de lectura y escritura, con el objetivo de cubrir el máximo número de casos, simulando el mayor número de aplicaciones multimedia posibles. Y por último, pero no por ello menos importante, realizaremos un estudio exhaustivo sobre el efecto de los mecanismos de *caching* y *prefetching* presentes tanto en el servidor como en los clientes NFS. El estudio de estos mecanismos y la gran importancia y repercusión en las operaciones de E/S, será el punto de partida para proponer la optimización del sistema NFS. A continuación, presentamos esquemáticamente el estudio experimental que se va a llevar a cabo en este capítulo.

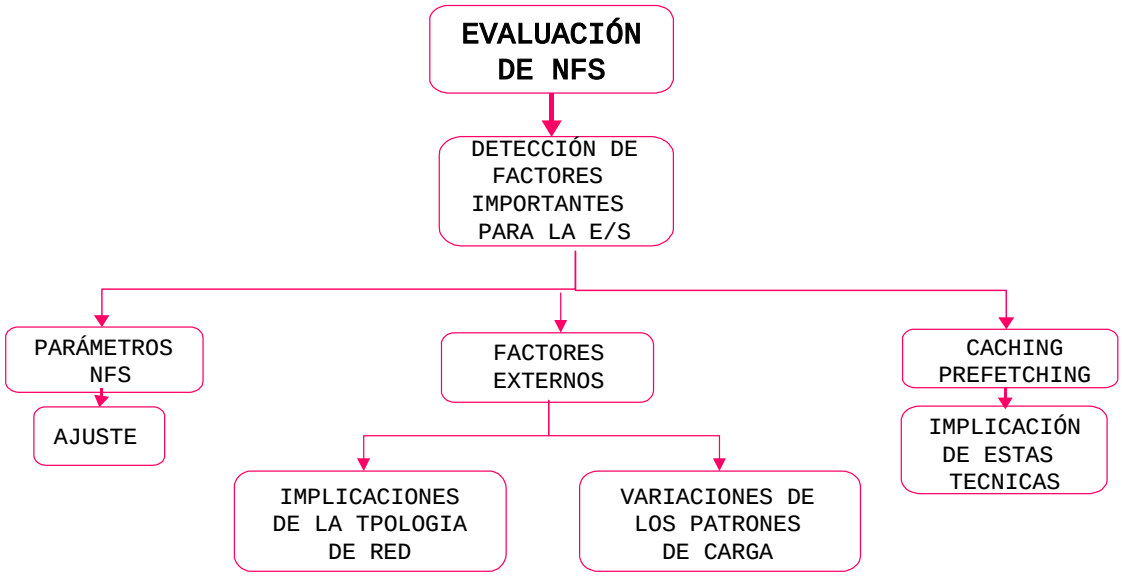


Figura 4.1. Desarrollo de los experimentos

4.3.PLANTEAMIENTOS PREVIOS

Nuestro objetivo es evaluar el comportamiento de NFS, y por lo tanto estaremos interesados en los tiempos de entrada/salida (lectura/escritura) obtenidos por los procesadores. Este va a ser nuestro parámetro de interés principal y el que querremos reducir con las distintas mejoras al sistema NFS. Vamos a utilizar el algoritmo paralelo espacial de división horizontal de datos (el método 2 mencionado en el capítulo 3) y esto lo hamos así porque el algoritmo espacial es la opción más utilizada, mucho más eficiente y realista que la versión temporal [153, 145]. Si representamos esquemáticamente los pasos que siguen los procesadores en el algoritmo de codificación básico de MPEG-2 (ver figura 4.2) podríamos definir los siguientes parámetros:

$$Tiempo\ de\ lectura = \sum_{i=1}^{i=n} (r2 - r1)$$

$$Tiempo\ de\ escritura = \sum_{i=1}^{i=n} (w2 - w1)$$

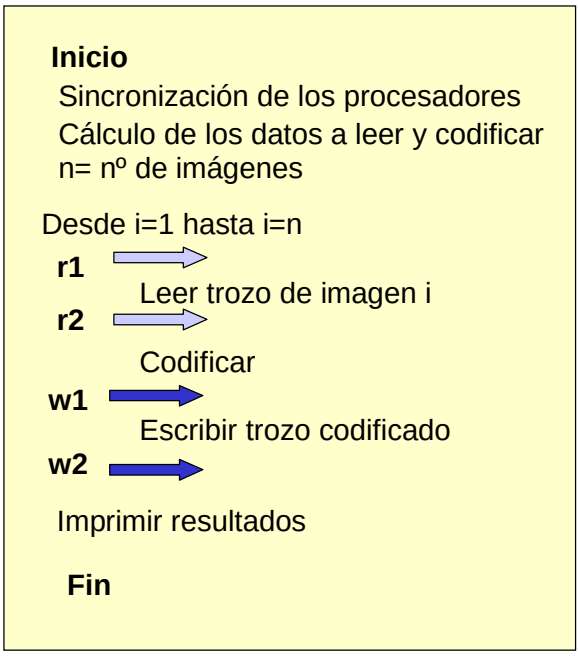


Figura 4.2. Cálculo de tiempos en el codificador

Probaremos distintos tamaños de paquetes entre clientes y servidores, es decir, distintos valores de rsize y wsize, dos parámetros de NFS, con los que veremos los resultados que se obtienen en cada caso. Empezaremos utilizando una versión inicial del codificador paralelo MPEG-2, donde cada una de los procesadores calcula el trozo que debe leer y codificar, a continuación, para cada una de las imágenes de la secuencia, lee el trozo que le corresponde, lo codifica y escribe los datos codificados en un fichero independiente para cada imagen en el servidor. Los datos se leen *con solapamiento* como ya se describió en el capítulo 3. A este patrón de carga lo llamaremos *patrón 1*: lectura solapada de segmentos de datos – escritura de datos.

En nuestras pruebas hemos utilizado la secuencia *Composed*, representativa de distintas formas de movimiento. Es una mezcla o composición de distintas secuencias. Como el reparto de datos entre los procesadores se hace de forma estática, es más conveniente utilizar una secuencia variada y evitar así beneficiar o perjudicar a determinados procesadores con el reparto de los datos, ya que van a existir dependencias entre los resultados y el contenido de la secuencia. La secuencia *Composed* original consta de 400 imágenes, que será precisamente la longitud de la secuencia utilizada.

Cada experimento se ha repetido tres veces y se ha tomado la media de las tres ejecuciones. Durante las pruebas, no había ninguna otra operación realizándose en las máquinas y se ha utilizado *cache fría*, es decir, se ha limpiado la cache entre ejecuciones, tanto en el cliente como en el servidor, para obtener datos fiables y evitar dependencias con las ejecuciones anteriores.

Por último, vamos a variar también, el número de procesadores que utilizaremos en las distintas pruebas. Disponemos, como ya hemos visto, de ocho máquinas. Una de

ellas es el servidor NFS (myrinet 1) y el resto son clientes (ver figura 4.3). Hemos podido comprobar en experimentos anteriores que si utilizamos el servidor en el proceso de codificación, éste ofrecerá unos valores de lectura y escritura muy por debajo del resto de procesadores. Por lo que, por el momento, solo va a realizar su papel de servidor NFS. El resto de maquinas (desde la myrinet 2 a la myrinet 8) se encargarán del proceso de codificación en paralelo. A esta configuración de partida la llamaremos *Topología A.*

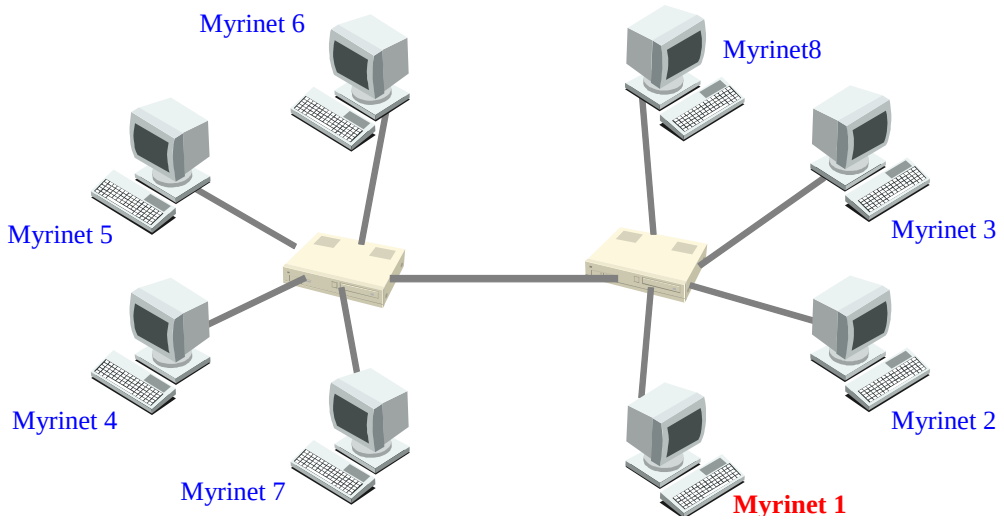


Figura 4.3. Topología A

4.4. DETECCIÓN DE FACTORES CRÍTICOS PARA LA ENTRADA/SALIDA

Para realizar un buen banco de pruebas que evalúe la entrada/salida en NFS, lo primero que tenemos que hacer es buscar cuáles son los factores que van a afectar a las operaciones de E/S. En este apartado vamos a centrarnos en explicar el alcance de esos factores, de su posible ajuste para obtener mejores resultados, y de exponer cual es su implicación directa en los resultados.

4.4.1. Parámetros NFS

Cuando decidimos utilizar las posibilidades que ofrece NFS, lo primero que tuvimos que hacer es, como ya se ha comentado, exportar el sistema de ficheros en el servidor y montarlo en los clientes. La operación de montaje se puede realizar en línea o mediante el fichero /etc/fstab (que es lo más usual). En el arranque de cada máquina cliente se montarán todos los sistemas de ficheros especificados en este fichero fstab. Por otro lado, a la hora de especificar el montaje para un sistema de ficheros NFS, existen distintos parámetros opcionales. La mayoría de ellos tiene unos valores por defecto que se adjudican inmediatamente sino se especifica lo contrario.

Algunos de ellos tienen una implicación directa en las operaciones de entrada/salida (lectura/escritura) de datos, como los siguientes:

♦ *rsize = n*. Es la cantidad de bytes que utiliza NFS para leer ficheros de un servidor NFS. El valor por omisión depende del núcleo, actualmente es 1024 bytes.

♦ *wsize = n.* Es la cantidad de bytes que NFS usa para escribir ficheros en un servidor NFS. El valor por omisión depende del núcleo, actualmente es 1024 bytes.

♦ *timeo = n.* Es el límite de tiempo, en décimas de segundo, usado antes de la primera retransmisión después de una expiración RPC. El valor por omisión es 7 décimas de segundo. Después de la primera expiración, el tiempo de expiración se duplica, así como después de cada expiración consecutiva hasta que se alcanza un tiempo de expiración máximo de 60 segundos o la cantidad de retransmisiones es muy grande. En ese momento, si el sistema de ficheros está montado en forma rígida (*hard*), cada cascada de expiraciones comienza con valores iniciales dos veces más grandes que la cascada anterior, duplicándose en cada retransmisión. La expiración máxima siempre es 60 segundos.

♦ *hard*. Si una operación de ficheros NFS tiene una expiración mayor, entonces imprime el mensaje *"server not responding"* (servidor no responde) en la consola y sigue intentándolo indefinidamente. Este es el comportamiento por omisión.

♦ *soft.* Si una operación de ficheros NFS tiene una expiración mayor, entonces devuelve un error de E/S al programa llamador. El valor por omisión es seguir intentando la operación NFS indefinidamente.

♦ *intr*. Si una operación de ficheros NFS tiene una expiración mayor y está montado de modo rígido, entonces permite interrumpir la operación y devolver EINTR al programa llamador. El comportamiento por omisión no permite que las operaciones de ficheros se interrumpan.

♦ *actimeo=n:* asigna el mismo valor n a las opciones *acregmin, acregmax, acdirmin, acdirmax*. Estas opciones establecen el tiempo mínimo y máximo que el sistema va a esperar antes de actualizar desde el servidor los atributos de un fichero o de un directorio

En general se recomienda el montaje físico salvo en caso de que se tenga información crítica, como la de servidores de FTP o particiones de noticias. En entornos críticos (por ejemplo, estaciones de trabajo X con dependencia de servidores de aplicaciones X Windows) no debe usarse el montaje lógico a riesgo de perder las conexiones si en un momento se satura o desactiva la red por algún motivo [113].

Los clientes de nuestra red utilizan inicialmente las siguientes opciones de montaje:

rsize = 1024, wsize = 1024, hard, intr, actimeo=600

4.4.2. Ajuste de parámetros

Empezaremos nuestros experimentos probando distintos valores de rsize y wsize, ya que utilizar valores más grandes mejora el rendimiento [113, 152]. Utilizaremos valores de 1024, 2048 y 4096 bytes con distinto número de procesadores. También hemos utilizado el valor de 8192, pero los resultados son casi idénticos a los obtenidos con 4096 bytes.

Presentaremos los resultados de dos formas distintas. Por un lado, en esta sección, mostraremos los valores medios para cada una de las pruebas, es decir, la media de los valores obtenidos por todos los procesadores. Estos valores nos serán útiles en la comparativa de los distintos casos y nos ayudarán a elegir los mejores valores para los parámetros en cada caso. Después presentaremos los valores medios por procesador. Es decir los valores individuales de los procesadores para ver así, las posibles diferencias entre ellos y ver cómo afecta la variabilidad de estos valores en los resultados finales. En la figura 4.4 representamos los tiempos medios de lectura para los distintos casos y en la tabla 4.1 aparecen todos los resultados.

Tabla 4.1. Tiempos de lectura

# PROC	RSIZE = 1024	RSIZE = 2048	RSIZE = 4096
1	70,5608	49,5706	45,0015
2	39,9867	27,8077	33,8151
3	29,8573	21,3875	29,75
4	21,9507	15,9964	24,442
5	21,7152	14,8032	21,4065
6	23,5668	17,2938	25,7739
7	27,2001	21,4557	36,3655

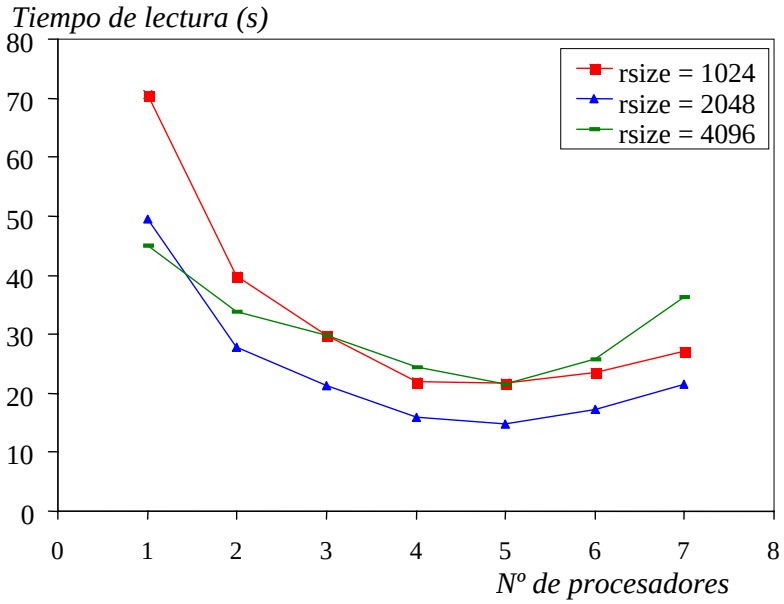


Figura 4.4. Tiempos de lectura para distinto número de procesadores

Podemos apreciar la importante diferencia entre el caso secuencial y el caso paralelo. Las ventajas de aumentar los valores de rsize están más que justificados para el caso secuencial (1 procesador), pero vemos que para los casos paralelos ya no está tan claro. Totalmente lógico, si tenemos un solo cliente leyendo del servidor, antes terminará cuanto más grandes sean los trozos que está leyendo. No hay más peticiones en el servidor, y por tanto tiene dedicación exclusiva.

En la figura 4.4 podemos ver que algo no va bien. A partir de 5 procesadores los tiempos de lectura no disminuyen, sino todo lo contrario. Con 6 y 7 procesadores los valores obtenidos para el tiempo de lectura son mayores que los valores obtenidos con 5 procesadores. El tiempo de lectura es crucial para obtener buenos resultados finales, ya que antes de procesar los datos correspondientes, o en este caso, codificar el trozo correspondiente de cada imagen, los clientes deben tomar la información del servidor de vídeo. Podemos ver que existe un problema de escalabilidad, que habrá que resolver para permitir obtener mejores resultados con el número máximo de procesadores posible.

Con respecto a los efectos del incremento de los valores de rsize podemos ver en la siguiente gráfica más claramente lo que está pasando.

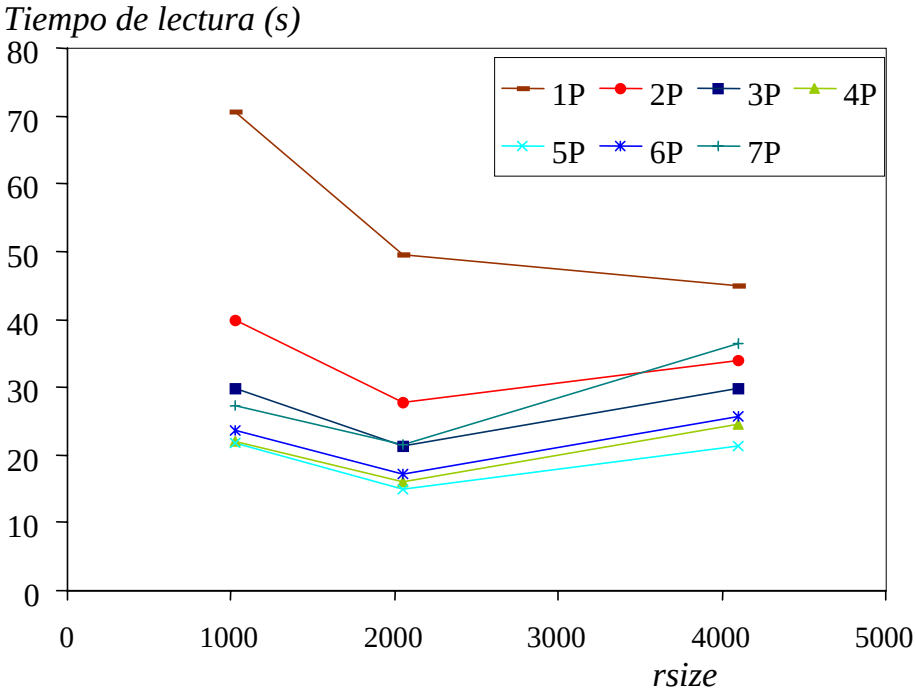


Figura 4.5. Tiempos de lectura en función de rsize

Si representamos estos tiempos en función del tamaño de bloque de datos utilizado podemos ver que el aumento de rsize de 2048 bytes a 4096 bytes conlleva un aumento en el tiempo de lectura para todos los casos paralelos. Tan solo en el caso secuencial es conveniente utilizar el mayor de los tamaños. También se puede ver en la figura anterior que los resultados de 5 procesadores son muy similares a los resultados con 4 procesadores y que los tiempos obtenidos con 6 y 7 procesadores están por encima de los obtenidos con 5 procesadores.

Veamos qué ocurre también con los tiempos de escritura de los trozos codificados en el servidor. La figura 4.6 muestra estos tiempos. Cada uno de los procesadores escribe su trozo codificado en un fichero distinto en una determinada ubicación en el servidor. Al final del proceso, el servidor concatenará la información almacenada en estos ficheros para formar un único flujo codificado de salida.

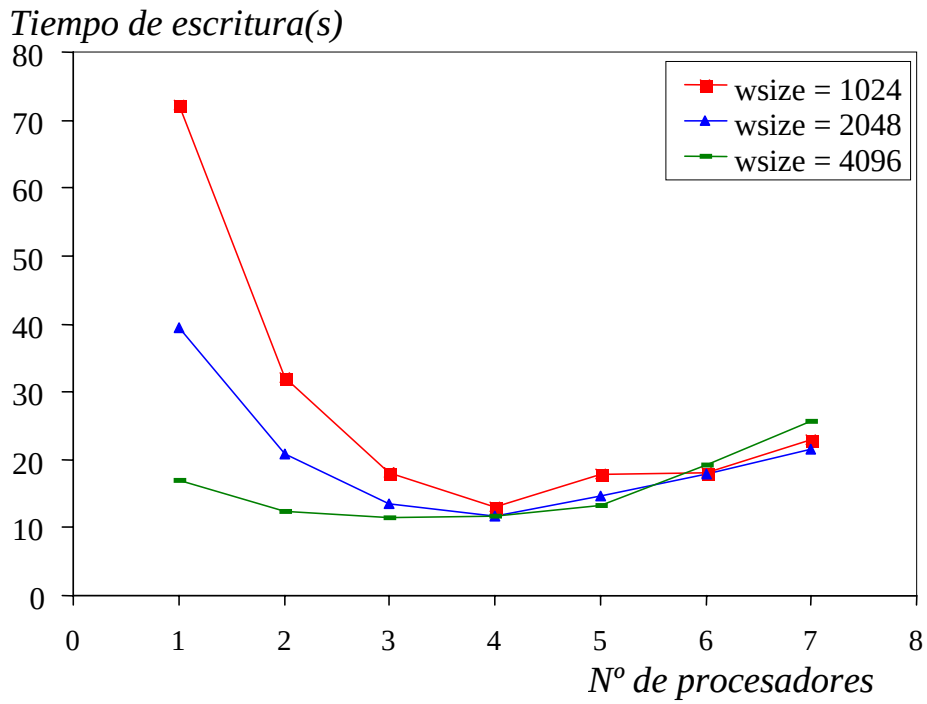


Figura 4.6. Tiempos de escritura

Podemos resaltar varias cosas de esta gráfica:

1. Para el caso secuencial, el aumento del tamaño de bloque implica una disminución muy importante en los tiempos.
2. A partir de 3 procesadores los tiempos son muy similares para todos los valores de wsize.
3. Aparece también el problema de escalabilidad a partir de 5 procesadores, ya que no existe una disminución progresiva de los tiempos de escritura.
4. Se tarda menos en escribir que en leer. La cantidad de datos que se va a escribir es menor que la cantidad de datos que se va a leer. Todos los procesadores escriben la misma cantidad de información, cosa que no ocurre con la lectura de datos.
5. Hay que tener en cuenta la independencia de ficheros, es decir, las peticiones de lectura van dirigidas a los mismos ficheros, y casi todas al mismo tiempo. A la hora de escribir, cada procesador crea su propio fichero, uno por cada trozo de imagen codificada

Otro parámetro que afecta bastante a la E/S es timeo. Este parámetro fija el tiempo que espera un cliente hasta que le responde el servidor. En los experimentos anteriores se han producido retransmisiones, como se mostrará más adelante. Estas retransmisiones son, en parte, las causantes de los elevados tiempos de lectura obtenidos y del consecuente problema de escalabilidad detectado. Ya hemos comentado que, por

defecto, tiene un valor de 7 décimas de segundo. Este valor nos parecía enorme para las actuales redes de alta velocidad. De ahí que nos planteásemos repetir las pruebas anteriores con valores más pequeños de timeo. Al estar definido como valor entero, el valor más pequeño era 1 décima de segundo. Realizamos estas pruebas, pero los resultados obtenidos han sido prácticamente idénticos a los anteriores. Para intentar descubrir lo que estaba pasando, medimos, en cada procesador los tiempos invertidos en leer cada una de las imágenes de forma individual, y saber realmente cual era el tiempo invertido en esto, mejor dicho, cúales eran los tiempos más probables invertidos en estas lecturas. Este estudio lo realizamos para el caso de 7 procesadores, ya que hay más posibilidad de que existan retransmisiones cuanto más cargado está el servidor.

Vamos a representar mediante una función de distribución acumulativa (CDF) esos tiempos de respuesta por parte del servidor, y así poder hablar de valores concretos. La tabla 4.2 muestra los datos obtenidos y la figura 4.7 refleja esta medida.

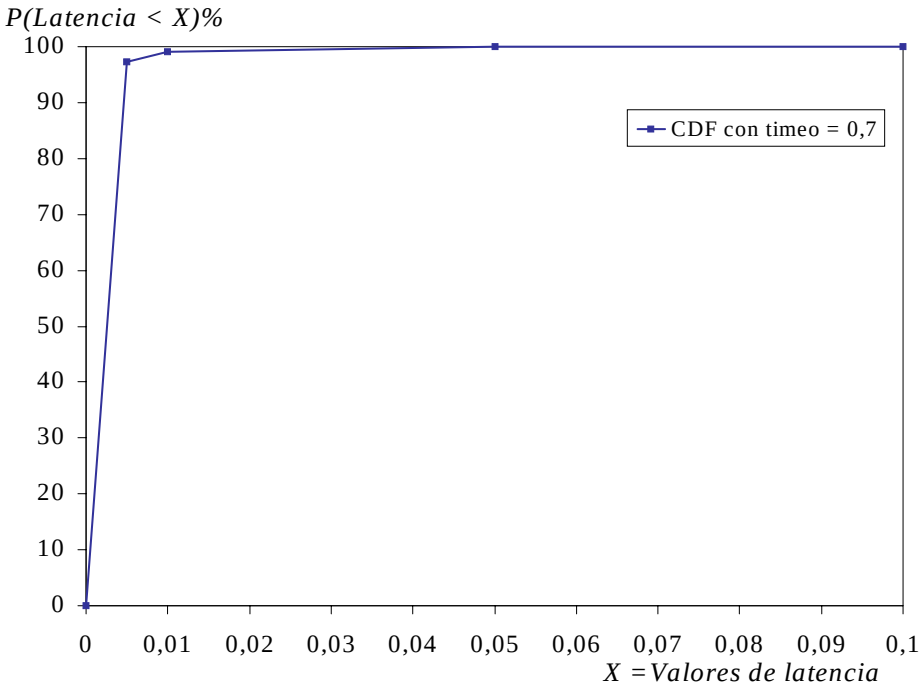


Figura 4.7. Función de distribución acumulativa

El 97% de los tiempos de respuesta son menores de 0,005 segundos, y el 99% son menores de 0,01 s (como se puede ver en la tabla 4.2). En nuestro intento de optimizar el sistema de ficheros también intentamos cambiar la definición de timeo en el código fuente del servidor NFS de un valor entero a un valor real. En la tabla 4.2 y en la gráfica 4.7 está más que demostrado la inutilidad de asignar 0,1 a este parámetro. Pues bien, nuestro intento de definición de timeo como valor real fracasó ya que también era necesario retocar otros códigos fuente como el del comando mount para que todo funcionase sin problemas. Por lo tanto decidimos no invertir más tiempo en este trabajo de definiciones de parámetro y plantear una solución mucho más general que evite estas lentísimas retransmisiones.

Tabla 4.2. Valores obtenidos con la CDF

T. RESPUESTA	PROBABILIDAD %
< 0,001	95,6320731
< 0,005	97,3515902
< 0,01	99,1227703
<0,05	99,9645669
< 0,1	99,9876975

Para terminar este estudio preliminar, vamos a representar la ganancia que supone un aumento en el número de procesadores para el tiempo de lectura y para el tiempo de escritura. En la siguiente figura representamos los valores correspondientes.

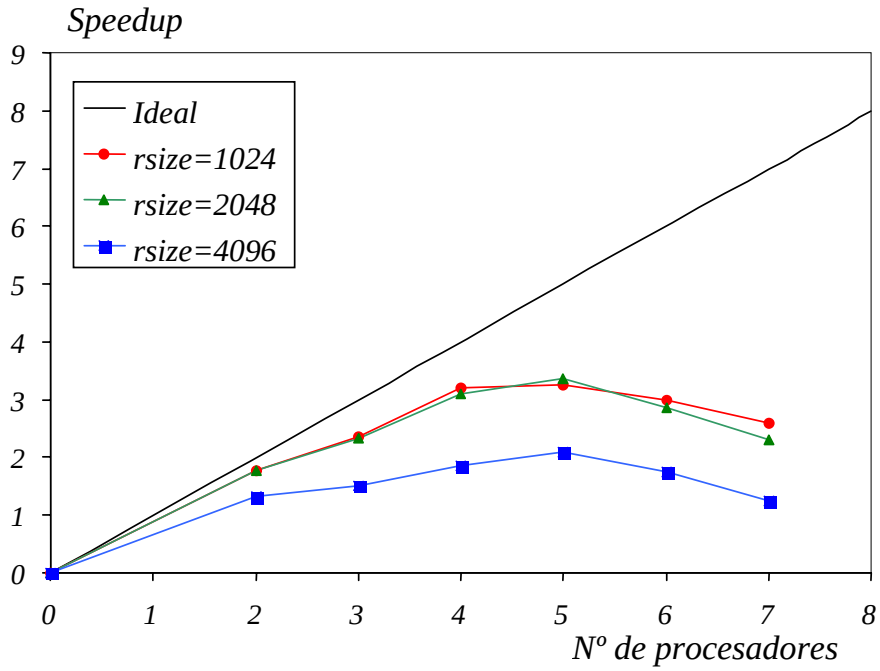


Figura 4.8. *Speedup* para los distintos valores de rsize

Podemos ver como a partir de 5 procesadores las curvas caen, y, definitivamente, con la utilización de rsize=4096 bytes se obtienen los peores resultados. Por lo tanto, sirva esta gráfica para plantear la necesidad de propuestas de mejora en el sistema de entrada/salida utilizado.

Veamos que ocurre ahora con la ganancia para los tiempos de escritura, con los distintos valores de wsize utilizados.

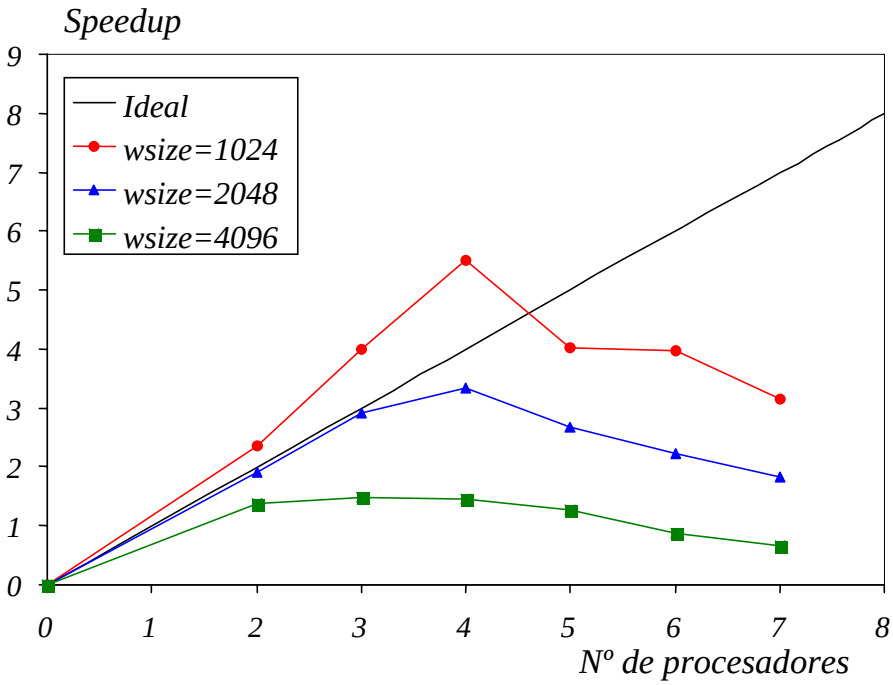


Figura 4.9. *Speedup* para los distintos valores de wsize

Podemos ver en esta gráfica la eficacia del paralelismo, sobre todo para el caso de wsize = 1024 bytes, donde se llega a un *speedup superlineal*. Hasta cuatro procesadores, el aumento de velocidad que supone un aumento en el número de procesadores, es bastante importante. A partir de 5 procesadores ese crecimiento desaparece y la disminución en el tiempo de escritura con el aumento en el número de procesadores, desaparece. Por lo tanto el problema de escalabilidad se hace presente a partir de 5 procesadores. Con los otros dos casos, wsize = 2048 y 4096 bytes, la ganancia no es tan espectacular. El utilizar paquetes más grandes para escritura mantendrá al servidor ocupado más tiempo con el mismo procesador, por lo que habrá que buscar el equilibrio entre el tamaño de paquete utilizado y el número de procesadores que participen. El problema de escalabilidad sigue estando presente a partir de cinco procesadores.

Pero, para no culpar a priori al sistema de ficheros empleado, vamos a realizar en las siguientes secciones, distintas pruebas variando otros factores hasta ahora fijos en todas las pruebas. Realizaremos una serie de experimentos con distintas topologías de red, para encontrar posibles responsabilidades por parte de la red utilizada en los resultados obtenidos. Y también vamos a realizar pruebas con otros patrones de carga, ejemplo de los requisitos exigidos por otras aplicaciones multimedia.

4.4.3. CAMBIOS EN LA TOPOLOGÍA DE RED UTILIZADA

Antes de continuar el estudio centrándonos en otros parámetros importantes, vamos a plantear otras topologías de red para comprobar si es la propia red la responsable de los problemas de escalabilidad detectados. Sobre esas nuevas topologías realizaremos también las mismas pruebas que se hicieron sobre la configuración

anterior para ver si es posible que existan cuellos de botella en el cable que une los conmutadores o en el propio conmutador. Como ya se explicó en el capítulo 3, tenemos dos conmutadores, uno de ellos con 8 puertos LAN y otro con 4 puertos LAN y 4 SAN. Con respecto a las máquinas, 6 de ellas se pueden conectar a los puertos LAN y 2 a los puertos SAN. Con todo esto planteamos las dos nuevas configuración posibles:

Configuración B: un solo conmutador activo y seis estaciones conectadas a él (ver figura 4.10). Como la myrinet 1 actúa como servidora de vídeo, tan solo tendremos 5 máquinas codificando. Vamos a ver lo que ocurre al tener un solo conmutador. Eliminamos el cable que une los conmutadores. Señalar que con esta configuración se tiene el máximo número de conexiones punto a punto en el conmutador, vamos a ver cómo responde el propio conmutador ante esta situación

Configuración C: otra posibilidad que nos permitiría utilizar más máquinas y evitar el posible colapso de peticiones entre los conmutadores (si lo hubiera) consiste en aprovechar el puerto que nos queda libre y poner otro cable conector entre los dos conmutadores, para agilizar el tráfico entre ellos (ver figura 4.11). Así tendríamos en funcionamiento las ocho máquinas y los dos conmutadores, y podremos realizar pruebas más interesantes con el máximo número de procesadores codificando, y por supuesto leyendo y escribiendo en el servidor NFS.

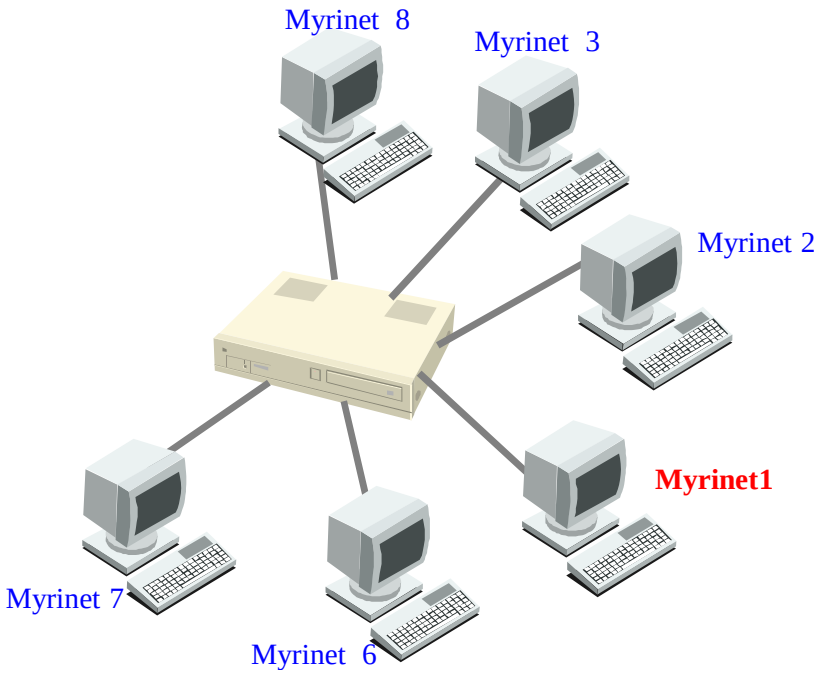


Figura 4.10. Topología B

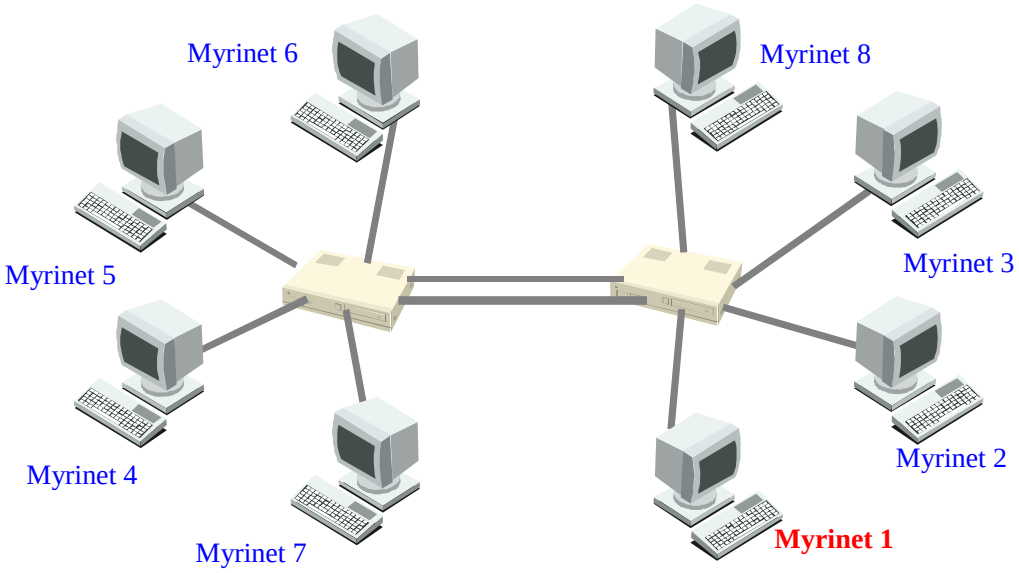


Figura 4.11. Topología C

Empezamos evaluando la configuración B. En la siguiente gráfica comparamos los valores de tiempos de lectura obtenidos con las configuraciones A y B, para así poder comprobar si existen mejoras apreciables con la nueva topología.

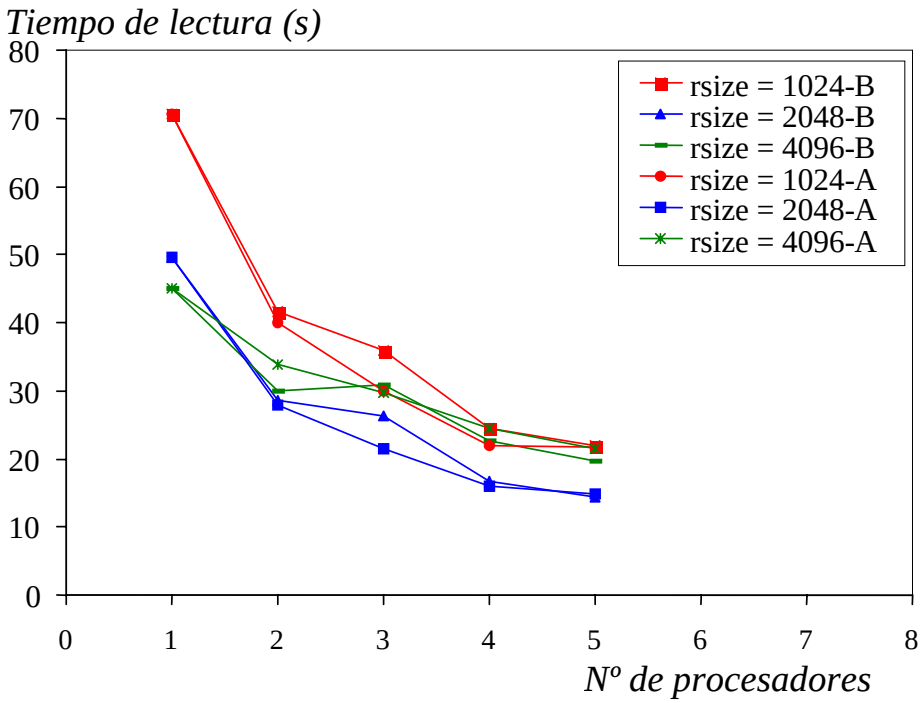


Figura 4.12. Tiempos de lectura para las topologías A y B

Como podemos ver, para cada uno de los valores de rsize los datos obtenidos con cada una de las dos configuraciones son bastante similares, lo que indica que no ha habido ninguna mejora considerable al forzar la comunicación sólo dentro de un mismo conmutador.

Por otro lado, si el propio conmutador fuese el cuello de botella ahora tendríamos peores resultados ya que hemos aumentado el número de conexiones dentro del mismo. Por otro lado, ante la posibilidad de que el cable que une los conmutadores fuese el cuello de botella, ponemos dos cables para agilizar las comunicaciones. Los resultados de estos experimentos los vemos en la figura 4.13.

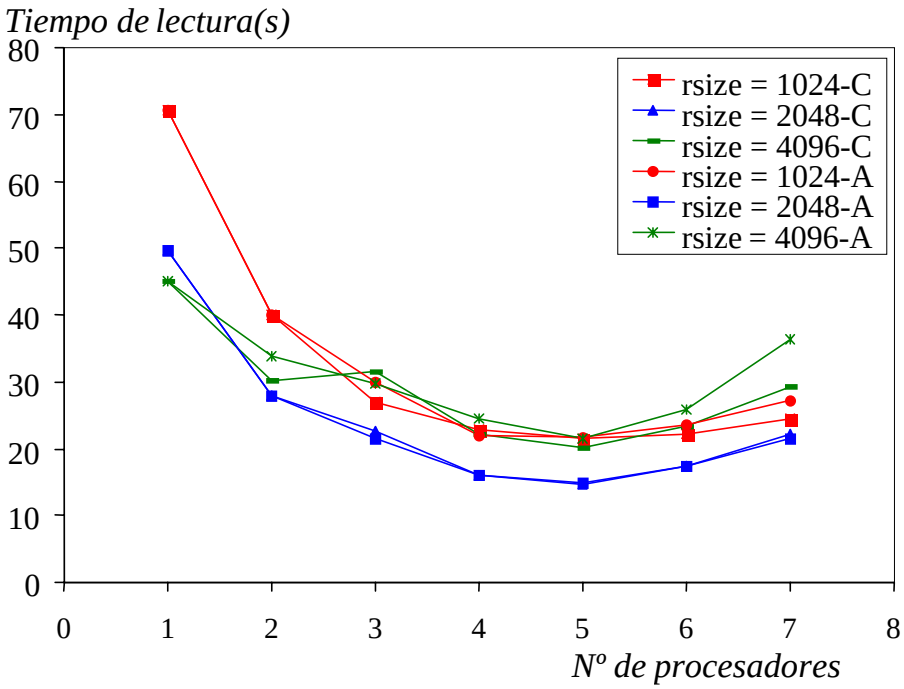


Figura 4.13. Tiempos de lectura para las topologías A y C

Los resultados para ambas configuraciones son bastante similares, el añadir un nuevo cable conector entre los conmutadores no ha supuesto ningún cambio importante. Por lo tanto, podemos decir que la causa de los problemas detectados tampoco está en la conexión de los conmutadores de la red.

Debido a la similitud de resultados, no representamos los tiempos de escritura de forma detallada, porque va a ocurrir algo similar a los tiempos de lectura, solo representaremos el *speedup* que nos dará una idea de la ganancia en tiempo al aumentar el número de procesadores para las nuevas topologías de red

Vamos a representar ahora los *speedups* para los tiempos de lectura y escritura obtenidos con las nuevas configuraciones. Los representamos en la figura 4.14. y 4.15. En la figura 4.14 se puede apreciar claramente la similitud en los resultados obtenidos al variar las topologías de red para todos los valores de rsize. Los peores resultados se obtienen utilizando el mayor tamaño de bloque de datos, 4096 bytes, y por supuesto el problema de escalabilidad aparece a partir de 5 procesadores.

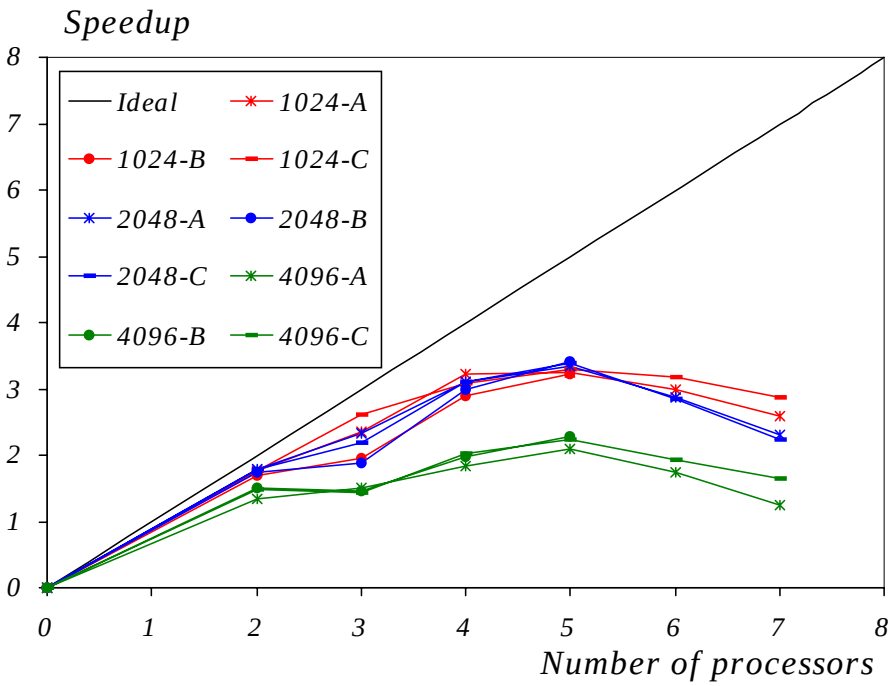


Figura 4.14. *Speedups* obtenidos con las tres configuraciones para distintos valores de rsize

En la siguiente figura representamos el *speedup* conseguido para los tiempos de escritura con distintos valores de wsize.

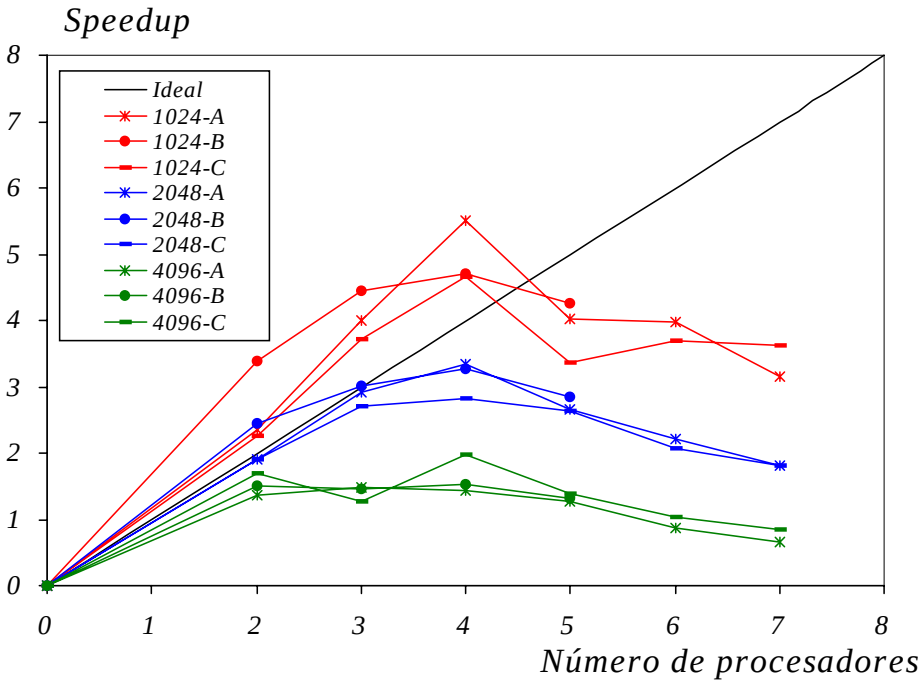


Figura 4.15. *Speedups* obtenidos con las tres configuraciones para distintos valores de wsize

Podemos ver que ocurre lo mismo que para los tiempos de lectura. Los cambios en la topología de la red no tienen ningún efecto importante en los tiempos de E/S obtenidos. Los problemas detectados no se deben a cuestiones físicas de la red o de la arquitectura propia de los conmutadores, sino que están originados por el propio sistema NFS. Por lo que, a partir de ahora, todas las pruebas las realizaremos sobre la topología A.

4.4.4. PRUEBAS CON DISTINTOS PATRONES DE CARGA

Nos proponemos ahora evaluar nuestro sistema de ficheros utilizando nuevos patrones de E/S, distintos al utilizado en todos los experimentos anteriores. Implementaremos otras formas de lectura y escritura para simular los requisitos de lectura/escritura de otras aplicaciones multimedia y realizar pruebas con los patrones de carga que éstas plantearían. Veremos qué ocurre con otros patrones diferentes de E/S sobre la configuración A y rsize y wsize = 4096 bytes (por ser el caso con peores resultados). Ahora fijamos los parámetros NFS (rsize y wsize) y la topología de red, para variar otro factor muy interesante como es la forma en que se van a realizar las peticiones de lectura y escritura, lo que hemos llamado patrones de carga del servidor NFS.

Existen actualmente multitud de aplicaciones multimedia de solo lectura en redes de alta velocidad cuyo interés crece día a día, como el Vídeo bajo Demanda o todas aquellas destinadas a proporcionar una fuente de información audiovisual al usuario (museos virtuales, librerías digitales, etc.). Estas aplicaciones conllevan una lectura masiva y concurrente de datos del servidor, y por lo general, distintos clientes quieren leer los mismos datos. En todas ellas existen distintos usuarios pidiendo información a distintos servidores. Precisamente el auge reciente de este tipo de aplicaciones ha hecho que exista un aumento en los trabajos de investigación dedicados a alcanzar un diseño optimizado para servidores y "proporcionadores" de servicios de red multimedia. También está aumentando el número de aplicaciones multimedia controladas por el usuario, es decir, aplicaciones donde el usuario quiere tener el control de visión y presentación de resultados, y por lo tanto quiere tener acceso transparente a la información.

Basándonos en todas estas aplicaciones, planteamos los siguientes patrones de carga posibles:

- *Patrón 1:* lectura solapada de segmentos de datos - escritura de datos
- *Patrón 2:* lectura de ficheros completos - escritura de datos
- *Patrón 3:* Lectura de ficheros completos - no escritura
- *Patrón 4:* lectura solapada de segmentos de datos - no escritura
- *Patrón 5:* lectura no solapada de segmentos de datos - escritura de datos
- *Patrón 6:* lectura no solapada de segmentos de datos - no escritura de datos

Hemos definido estos patrones teniendo en cuenta los requisitos de E/S de la mayoría de las aplicaciones multimedia actuales. El patrón 1 ha sido el utilizado en todas las pruebas anteriores, y consiste en la lectura con solapamiento de datos por parte de los distintos clientes, es decir, habrá datos dentro del mismo fichero solicitados por más de un cliente. Una vez realizado el procesamiento oportuno con esos datos, los

clientes escriben nuevos datos en el servidor. Hemos considerado las operaciones de escritura como una forma de cargar más al servidor. Lo que ocurre es que el servidor está ocupado aún más que si solo hubiese operaciones de lectura.

Con el patrón 2 (lectura de ficheros completos - escritura de datos) estamos forzando el sistema para que todos los clientes lean la misma cantidad de información y realicen las mismas peticiones de lectura al servidor. Si tenemos un servidor de bases de datos, por ejemplo, éste pude recibir peticiones de gestión y mantenimiento de las bases de datos almacenadas que impliquen lecturas de ficheros completos de datos y las consecuentes inserciones y actualizaciones de datos. Será interesante ver que ocurre en el servidor cuando todos los clientes quieren leer exactamente los mismos ficheros.

El patrón 3 (lectura de ficheros completos - no escritura) es un claro ejemplo de lo que ocurriría en sistemas de Vídeo bajo Demanda, donde distintos clientes pueden estar pidiendo las mismas películas al mismo tiempo. En este caso solo hay peticiones de lectura. También es similar a lo que ocurre con un servidor web, que continuamente está recibiendo las peticiones de distintos clientes que pueden estar queriendo acceder a exactamente la misma información.

Con el patrón 4 (lectura solapada de segmentos de datos - no escritura) se quieren representar todas aquellas aplicaciones que no conllevan tampoco operaciones de escritura pero en la que los distintos clientes van a leer datos solapados de los mismos ficheros. Habrá por tanto distintos clientes pidiendo los mismos datos almacenados dentro de distintos ficheros. Esto sería un ejemplo de simples consultas a un servidor de bases de datos o también de un servidor web.

Hemos querido completar el estudio con los patrones 5 (lectura no solapada de segmentos de datos - escritura de datos) y 6 (lectura no solapada de segmentos de datos - no escritura de datos) para representar otra forma de leer datos de un fichero de forma exclusiva, es decir, aquí no va a haber más de dos clientes pidiendo los mismos datos, solicitarán leer los mismos ficheros pero no los mismos trozos de datos dentro de ellos. Con el patrón 5 también hay escritura de datos y con el patrón 6 solo hay operaciones de lectura. Seguirán este patrón todas aquellas aplicaciones multimedia paralelas de procesamiento de imágenes en los que sea necesario solo una determinada porción de la imagen para realizar determinadas operaciones.

Justificada, por tanto, la distinta elección de patrones, vamos a mostrar los datos obtenidos con ellos. En la figura 4.16 representamos los tiempos de lectura obtenidos.

De esta figura, lo que más nos impresiona, antes de entrar en detalle, son los elevados resultados que se obtienen para el patrón 2 (se leen los ficheros completos y se escriben los datos codificados en el servidor). El patrón 3 se acerca bastante al 2 pero al no tener el servidor la sobrecarga de la escritura no se disparan tanto los resultados. Vemos claramente como con el patrón 2 el servidor se colapsa y tan solo con tres

procesadores ya aparecen los problemas. Además, los tiempos obtenidos con 7 procesadores son más del doble de los obtenidos con la versión secuencial.

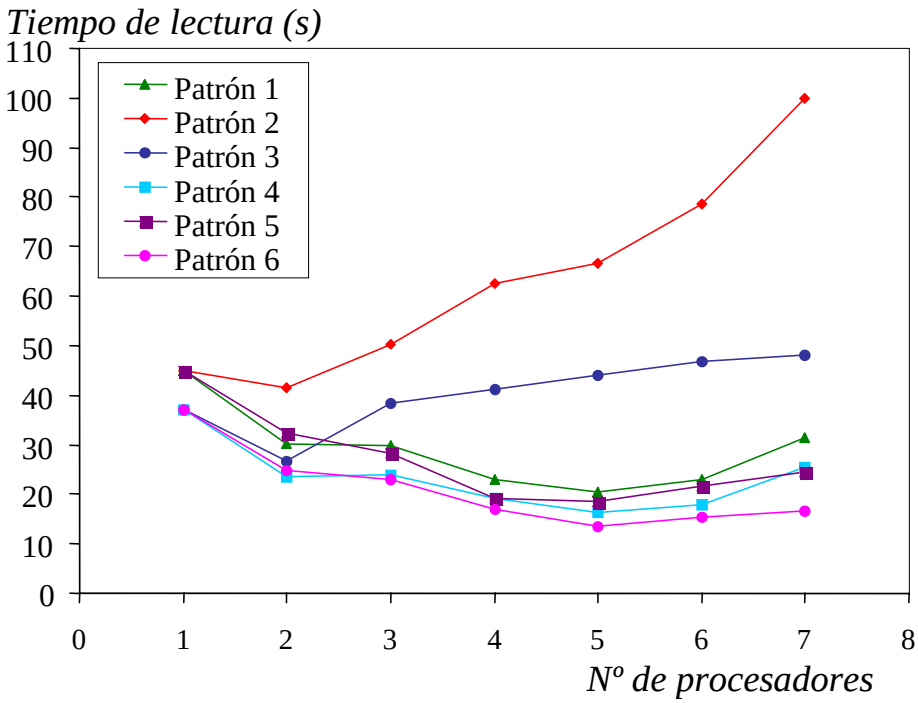


Figura 4.16. Tiempos de lectura para los distintos patrones de E/S

Los patrones 1 y 4 ofrecen tiempos de lectura muy similares. El patrón 4 solo difiere del patrón 1 en que también se realizan operaciones de escritura en el servidor, con lo que se carga un poco más al mismo. En ambos casos existe el problema de escalabilidad casi de forma idéntica. A partir de 5 procesadores los resultados que se obtienen son peores que los anteriores.

Si comparamos los patrones 1 y 5 (ya que con el 1 la lectura es solapada y con el 5 no) podemos ver que los resultados son muy similares y que no hay importantes diferencias, por tanto, los problemas de escalabilidad detectados en los apartados anteriores con el patrón 1 no se deben a la lectura solapada de datos. Quizás lo único que podríamos señalar es la leve diferencia entre los tiempos de lectura para el caso de 7 procesadores. En el caso de lectura no solapada los resultados son un poco menores, debido a que se lee menos información.

En cuanto a los patrones 4 y 6 en los que no existen operaciones de escritura, vemos que ocurre lo mismo que con los patrones 1 y 5, salvo que los tiempos obtenidos son un poco menores al no tener operaciones de escritura. Los problemas de escalabilidad aparecen a partir de 5 procesadores, y tan solo hay una leve diferencia en el caso de 7 procesadores.

Pero lo más destacable es la gran diferencia que existe entre los patrones 1 y 2. Vamos a ver para algunos casos (2, 4 y 7 procesadores) el tiempo medio de lectura invertido por procesador y los datos leídos por procesador:

Tabla 4.3. Datos para el patrón 1

#PROC	T.LECTURA/PROC	DATOS LEÍDOS/PROC
2	30,2133	345600
4	22,8671	207360
7	31,4402	148112

Tabla 4.4. Datos para el patrón 2

#PROC	T.LECTURA/PROC	DATOS LEÍDOS/PROC
2	41,55	622080
4	62,5266	622080
7	99,818	622080

Con el patrón 2 todos leen siempre la misma cantidad de información, sea cual sea el número de procesadores. Si evaluamos la relación que existe entre los tiempos obtenidos con los patrones 1 y 2 podemos ver que:

- *Con 2 procesadores*: el tiempo en leer con el patrón 2 es un 27,3% mayor que el tiempo en leer con el patrón 1, pero la cantidad a leer con el patrón 2 es un 44,44% mayor que la que se lee con el patrón 1. Además, si con el patrón 1 se tardan en leer los 345600 bytes, 30,2133 segundos, en leer 622 080 bytes se tardarían 54,3839 segundos, que es más de los 41,55 segundos invertidos.

- *Con 4 procesadores:* el tiempo en leer con el patrón 2 es un 63,4282% mayor que el tiempo en leer con el patrón 1, pero la cantidad a leer con el patrón 2 es un 66,67% mayor que la que se lee con el patrón 1. Además, si con el patrón 1 se tardan en leer los 207 360 bytes, 22,8071 segundos, en leer 622 080 bytes se tardarían 68,6013 segundos, que es más de los 62,5266 segundos invertidos

- *Con* 7 *procesadores:* el tiempo en leer con el patrón 2 es un 68,5025% mayor que el tiempo en leer con el patrón 1, pero la cantidad a leer con el patrón 2 es un 76,191% mayor que la que se lee con el patrón 1. Además, si con el patrón 1 se tardan en leer los 148 112 bytes, 31,4402 segundos, en leer 622 080 bytes se tardarían 132,051 segundos, que es más de los 99,818 segundos invertidos

Por lo tanto, si comparamos el patrón 2 con el 1, vemos que si el tiempo invertido fuera proporcional a la cantidad de información leída, los tiempos obtenidos con el patrón 2 aún deberían ser mayores. Y es que estamos comparando los tiempos obtenidos con el patrón 2 con un patrón, el 1, en el que ya existen problemas de

escalabilidad, y los tiempos, por tanto, no son, ni mucho menos, los ideales. Así, la diferencia entre el patrón 1 y el 2 se debe claramente a que con el patrón 2 los procesadores leen más cantidad de información.

Existe menos diferencia entre los patrones P3 y P4 (cuando ya no hay operaciones de escritura). Repitamos el cálculo anterior para demostrar matemáticamente lo que está pasando:

Tabla 4.5. Datos para el patrón 3

#PROC	T.LECTURA/PROC	DATOS LEÍDOS/PROC
2	26,56	622080
4	41,1175	622080
7	48,133	622080

Tabla 4.6. Datos para el patrón 4

#PROC	T.LECTURA/PROC	DATOS LEÍDOS/PROC
2	23,685	345600
4	19,3125	207360
7	25,4714	148112

Para ambos patrones, ahora no hay operaciones de escritura. Con el patrón 4 existe paralelismo de datos pero con el patrón 3 no (ya que todos leen siempre la misma cantidad de información, sea cual sea el número de procesadores). Si evaluamos la relación que existe entre los tiempos obtenidos con los patrones 3 y 4 podemos ver que:

- *Con 2 procesadores*: el tiempo en leer con el patrón 3 es un 26,56% mayor que el tiempo en leer con el patrón 4, pero la cantidad a leer con el patrón 3 es un 44,44% mayor que la que se lee con el patrón 4. Además, si con el patrón 4 se tardan en leer los 345600 bytes, 23,685 segundos, en leer 622 080 bytes se tardarían 42,633 segundos, que es más de los 41,55 segundos invertidos.

- *Con 4 procesadores:* el tiempo en leer con el patrón 3 es un 53,53% mayor que el tiempo en leer con el patrón 4, pero la cantidad a leer con el patrón 3 es un 66,67% mayor que la que se lee con el patrón 4. Además, si con el patrón 4 se tardan en leer los 207 360 bytes, 19,3125 segundos, en leer 622 080 bytes se tardarían 57,9375 segundos, que es más de los 41,1175 segundos invertidos

- *Con 7 procesadores:* el tiempo en leer con el patrón 3 es un 47,0812% mayor que el tiempo en leer con el patrón 4, pero la cantidad a leer con el patrón 3 es un 76,191% mayor que la que se lee con el patrón 4. Además, si con el patrón 4 se tardan en leer los 148 112 bytes, 25,4714 segundos, en leer 622 080 bytes se tardarían 106,98 segundos, que es mucho más de los 48,133 segundos invertidos

En esta comparación, podemos ver que los resultados obtenidos con el patrón 3 están más cerca de los obtenidos con el patrón 4. No existe tanta diferencia como al comparar los patrones 1 y 2. Por lo tanto hay algo más que afecta a los resultados. En la siguiente sección volveremos a esta gráfica de los patrones de carga para estudiar el efecto de las técnicas de *caching* y *prefetching* presentes en NFS, y sobre todo el efecto que va a tener el *read-ahead* del cliente.

Ahora vamos a representar los tiempos de escritura invertidos por los distintos patrones. Representaremos estos tiempos para los patrones 1 (lectura solapada de segmento de datos – escritura de datos), 2 (lectura de ficheros completos – escritura de datos) y 5 (lectura no solapada de segmento de datos – escritura de datos).

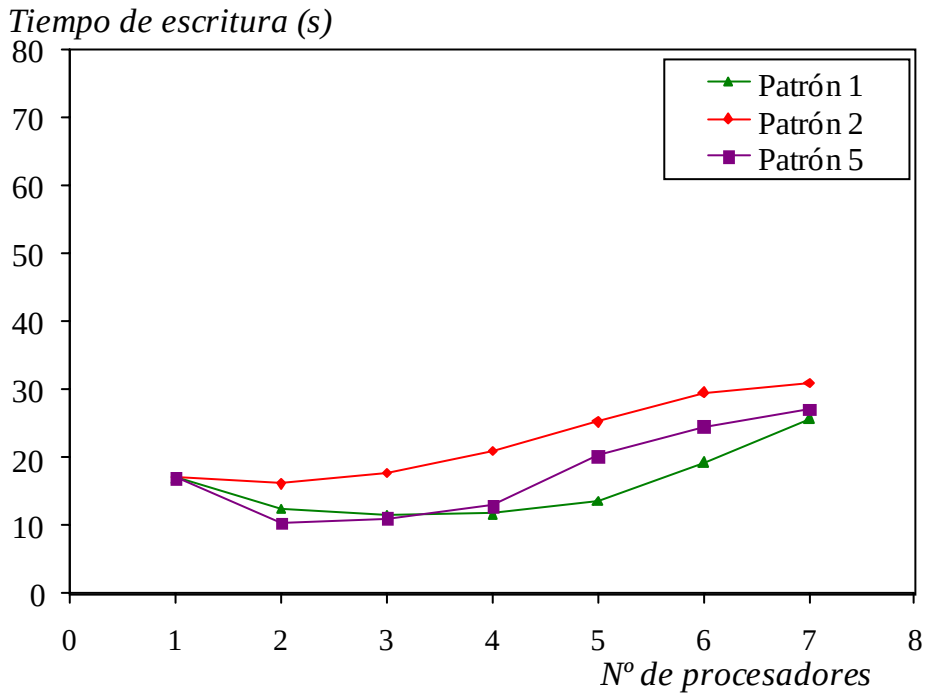


Figura 4.17. Tiempos de escritura para los distintos patrones de E/S

Como podemos ver en la anterior figura, los tiempos de escritura son bastante similares sea cual sea el patrón de carga empleado. Bien es verdad, que el que peores resultados ofrece es el patrón 2, que es precisamente aquél en el que el servidor está más cargado por existir un número mayor de peticiones de lectura.

Representamos ahora el *speedup* conseguido para todos los patrones planteados. En primer lugar para los tiempos de lectura (figura 4.18). Como era de esperar, las curvas para los patrones 2 y 3 caen rápidamente, mientras que para el resto caen a partir de 5 procesadores.

A continuación representamos el *speedup* para los tiempos de escritura (figura 4.19). La ganancia con el aumento de procesadores es bastante pequeña, y como se esperaba, los peores valores son los obtenidos con el patrón 2.

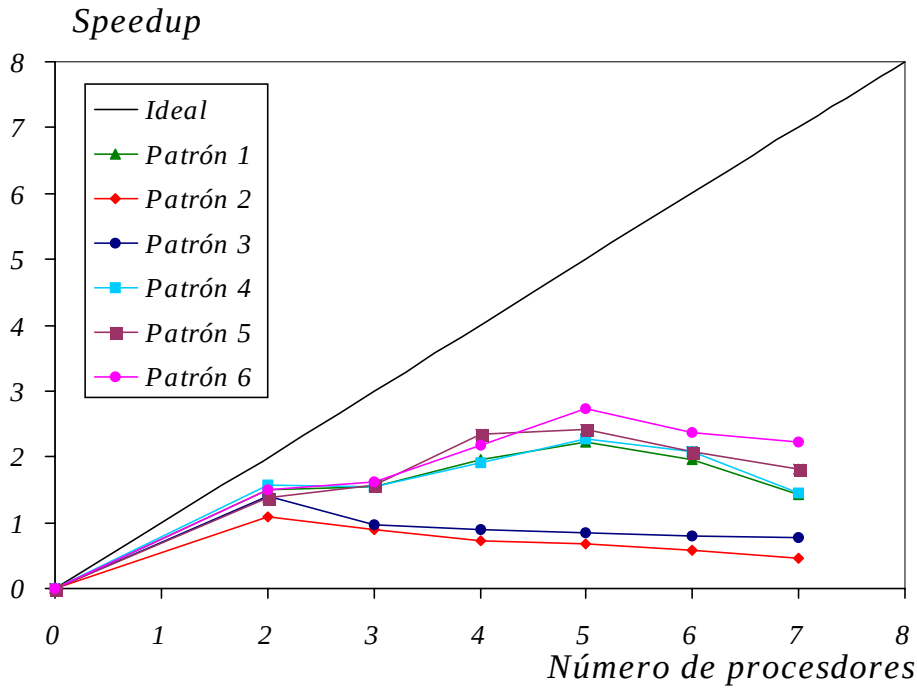


Figura 4.18. *Speedup* de los tiempos de lectura para los diferentes patrones de carga

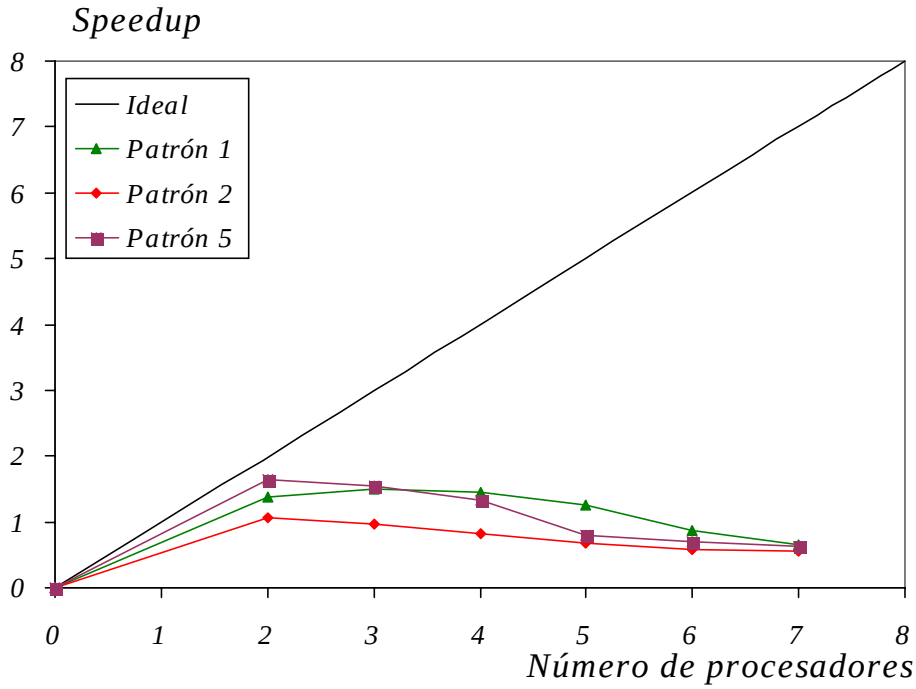


Figura 4.19. *Speedup* de los tiempos de escritura para los diferentes patrones de carga

De estas gráficas se puede deducir que existe un problema claro de escalabilidad en el servidor NFS. Además, si probamos distintas formas de acceder a la información, el caso más problemático y con el que peores tiempos de lectura y escritura se obtienen es aquel en el que los distintos clientes quieren leer todos los ficheros completos de datos y escribir el resultado de la codificación en el servidor.

El problema está servido, aunque todavía nos queda un factor importantísimo por estudiar: el efecto de la cache. En la siguiente sección vamos a hacer un estudio detallado de las implicaciones de estas técnicas utilizadas en NFS, pensadas, por supuesto, para mejorar el acceso a la información, para reducir el número de operaciones de E/S y para reducir el numero de peticiones por parte de los clientes, en definitiva el tráfico por la red. Pero vamos ver si este es el efecto que se consigue con el procesamiento paralelo. Veremos a fondo, tanto en el servidor como en los clientes, el uso que se está haciendo de la cache, y plantearemos un estudio detallado de los tiempos obtenidos por procesador, en lugar de las medias totales representadas anteriormente. Con estos tiempos explicaremos los resultados obtenidos en las gráficas anteriores y los problemas detectados. Además, será precisamente en las técnicas de *caching* y *prefetching* presentes en NFS, donde situemos el origen del planteamiento de nuestra aportación y mejora del sistema NFS.

4.4.5. TÉCNICAS DE *CACHING* Y *PREFETCHING* EN NFS

ya vimos en el capítulo 3 en qué consistía el *caching* y *prefetching* y, también, n las técnicas de *caching* y *prefetching* existentes en NFS. En este apartado lo que queremos estudiar es el efecto de tales técnicas.

El término *caching* se refiere al almacenamiento temporal de datos de fichero en un dispositivo de almacenamiento local de rápido acceso llamado cache [102]. Después del *caching*, el *prefetching* es la siguiente etapa lógica para incrementar el rendimiento de los sistemas de E/S, y consiste en adelantar los datos a la cache del cliente antes de que se necesiten para así reducir el número de peticiones al servidor. El *prefetching* en el cliente NFS se consigue con el procedimiento de *read-ahead*, trayendo más datos de los específicamente pedido, pero hay que leer una gran cantidad de datos de forma secuencial para que este mecanismo se desencadene.

También comentamos en el capítulo 3, las distintas versiones de NFS y las características y objetivos que se seguían con cada una de ellas y con cada revisión del protocolo. Con respecto al *caching* las únicas diferencias importantes entre protocolos son dos:

- *La versión 3 difiere de la versión 2 en la posibilidad de realizar escrituras asíncronas. Solo con la escritura asíncrona se cachean las peticiones de escritura. Con la síncrona, no.*

- *La versión 4 difiere de las anteriores en la definición de un nuevo procedimiento llamado COMPOUND que combina operaciones individuales en una única petición RPC.*

La versión utilizada en estas pruebas experimentales ha sido la versión 3, ya que la versión 4 aún está en desarrollo. Las políticas de cache en NFS aseguran un rendimiento aceptable, pero son, por supuesto, mejorables.

Vamos a plantear ahora los resultados obtenidos en nuestros experimentos sobre los tiempos de E/S por procesador. Es decir, vamos a ver los tiempos resultantes (de lectura y escritura) en cada uno de los procesadores, en lugar de representar las medias totales.

Usaremos la topología A, y rsize=wsize=4096, y empezaremos con el patrón 1 (lectura de segmentos de datos – escritura de datos). Nuestro propósito es estudiar el comportamiento de NFS a lo largo del tiempo, tanto desde el lado del servidor como del de los clientes para ver cual es el funcionamiento interno del sistema e intentar descubrir las causas de los primeros problemas de escalabilidad detectados, explorando en sus caches y explicando el proceso de comunicación cliente-servidor.

Cuando un cliente (por ejemplo, la myrinet2) quiere leer ciertos datos de un fichero (por ejemplo, el fichero comp0.Y) , se desencadenan las siguientes peticiones desde dicho cliente y las siguientes respuestas desde el servidor (hemos filtrado los mensajes de la red con *tcpdump*):

Ejemplo 2:

```
19:35:20.929695 < myr2 > myr1: lookup “comp0.Y”
19:35:20.933382 < myr1 > myr2: reply ok  lookup
19:35:20.933597 < myr2 > myr1: read 4096 bytes @ 0
19:35:20.998633 < myr1 > myr2: reply ok  read
                    ...
```

En la primera columna aparece el instante de tiempo en el que se realiza la petición o la respuesta. A continuación aparece el autor de la petición/respuesta y en la tercera columna a quien va dirigida esa petición/respuesta. La última parte corresponde a la descripción de la operación realizada. Como podemos ver en la secuencia anterior, cualquier lectura de datos de un fichero implica definir un gestor de ese fichero (mediante la operación lookup) y una serie de peticiones de lectura (read) de ese fichero. Por lo tanto, la operación de leer un trozo de datos de un fichero se traduce en una serie de *micro-reads* del tamaño de bloque establecido hasta completar toda la operación.

Cuando tenemos varias máquinas realizando peticiones al servidor, se van a entremezclar todas las peticiones provenientes de todas las máquinas y se van a producir los adelantamientos, solapamientos y retardos que vamos a detectar a continuación.

Con el patrón 1, no todos los procesadores leen la misma cantidad de información. En efecto, se leen segmentos de datos solapados, por lo tanto, vamos a calcular los trozos exactos que lee cada procesador de cada una de las imágenes de la secuencia.

Trabajamos con una secuencia PAL CCIR 601, YUV, lo que quiere decir que cada imagen tiene tres componentes, o tres frames (la luminancia y las dos crominancias) y que el tamaño del marco de luminancia es 720x576 bytes y el de las crominancias 360x288 bytes (la mitad del tamaño de la luminancia en ambas dimensiones). Por lo tanto, una imagen completa tiene un total de 622 080 bytes (unos 600 Kbytes).

Como se explicó en el capítulo 3, los procesadores a los que les corresponden el primero y último trozo de la imagen leen siempre menos bytes (solo añaden un *slice* a su trozo a codificar) que los procesadores a los que le corresponden algún trozo intermedio, leen dos *slices* más de datos sobre su trozo básico a codificar, uno por encima y otro por debajo (ver figura 4.20)

Figura 4.20. División horizontal de datos

Veamos, para algunas de las ejecuciones realizadas, qué está ocurriendo y cuales son los datos que se están utilizando:

- *Ejecución con dos procesadores*. Es el caso de paralelismo más simple, cada procesador va a codificar la mitad de cada frame y va a leer la mitad más una franja de datos de cada frame. En la siguiente tabla representamos la cantidad de bytes por imagen que leen cada uno de los procesadores y el tiempo de lectura invertido.

Tabla 4.7. Bytes/imagen leídos y Tiempo de lectura para cada procesador

PROCESADOR	BYTES/IMAGEN	T. LECTURA
P0	345600	25,1407
P1	345600	42,4895

En este caso los dos procesadores que participan en la codificación leen exactamente la misma cantidad de información. Entonces, ¿Por qué esas diferencias en el tiempo de lectura?. Como vemos este estudio va a ser mucho más interesante que el de los valores medios, porque las medias, precisamente, ocultan lo que está pasando.

Antes de nada, comentar el hecho de que no sólo se han realizado las pruebas utilizando siempre las mismas máquinas, sino que ante lo asombroso de algunos resultados cambiamos las máquinas y las situaciones se repetían. Es decir, el hecho de que, como en este caso, una máquina tarde mucho más que la otra en leer, no se va a deber a que una esté más cerca o más lejos de servidor. Por lo tanto ese factor lo descartamos. Si mostrásemos todos los casos probados, el estudio se haría interminable y nos desviaría del hilo conductor que intentamos seguir. Por ello nombraremos a los procesadores Pi, ..., Pj, en lugar de con sus nombres verdaderos, myrinet2, ...myrinet8.

Para ver lo que está pasando, vamos a representar los tiempos de lectura obtenidos para cada una de las imágenes, el tiempo entre peticiones de lectura y los instantes de petición de ambos procesadores. En la siguiente figura mostramos los tiempos de lectura para cada una de las 400 imágenes que forman la secuencia

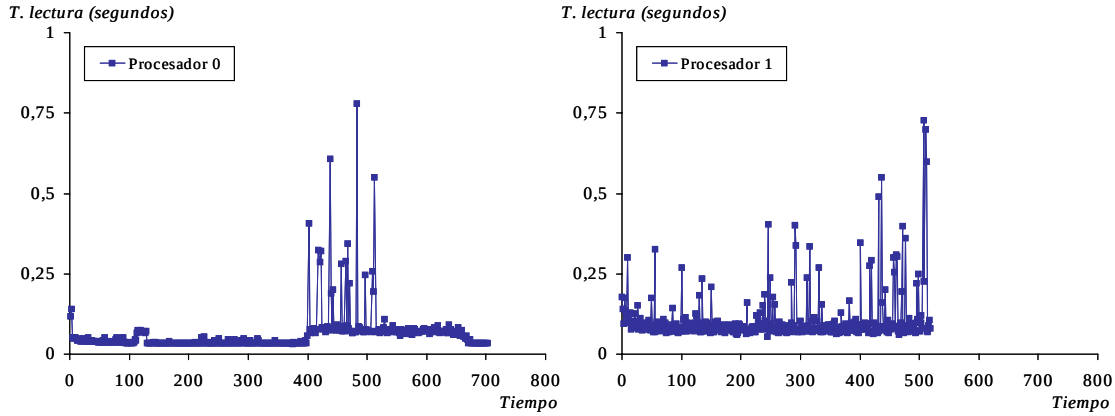


Figura 4.21. Tiempos de lectura para cada procesador

Hay dos diferencias fundamentales en el comportamiento de los procesadores:

1. Al principio, los tiempos de lectura de las imágenes son menores y muy similares para P0 y más variables y mayores para P1. Después los tiempos para P0 también aumentan produciéndose una especie de “escalón” en los tiempos de lectura
2. P0 acaba bastante después que P1

El primer punto se corresponde con la tabla 4.7. Si los tiempos de lectura para cada imagen son menores, al final, la suma de todos ellos seguirá siendo menor, que es lo que

pasa con P0. En el caso de P1 los tiempos de lectura para cada imagen son mayores y al final se obtiene un tiempo de lectura total mayor. Si representamos el histograma de estos tiempos de lectura de cada una de las imágenes para P0 y P1 podemos ver más claramente estos tiempos.

Con el histograma obtenemos una representación de la distribución de una variable cuantitativa que muestra la concentración relativa de los datos a lo largo de diferentes intervalos o secciones de la escala en la que están medidos dichos datos, es decir se resume una única variable numérica (en este caso el tiempo de lectura) y se muestra el número de casos que se encuentran dentro de cada intervalo.

Como podemos ver en la figura 4.22, los histogramas para P0 y P1 son ligeramente diferentes. Para empezar las escalas son distintas y de ahí las importantes diferencias. Para P0, la mayoría de los datos son menores de 0,075 segundos, el tiempo de lectura medio por imagen es precisamente 0,063 segundos. Mientras para P1 el intervalo es un poco más abierto, de ahí que la media por imagen sea 0,106 segundos.

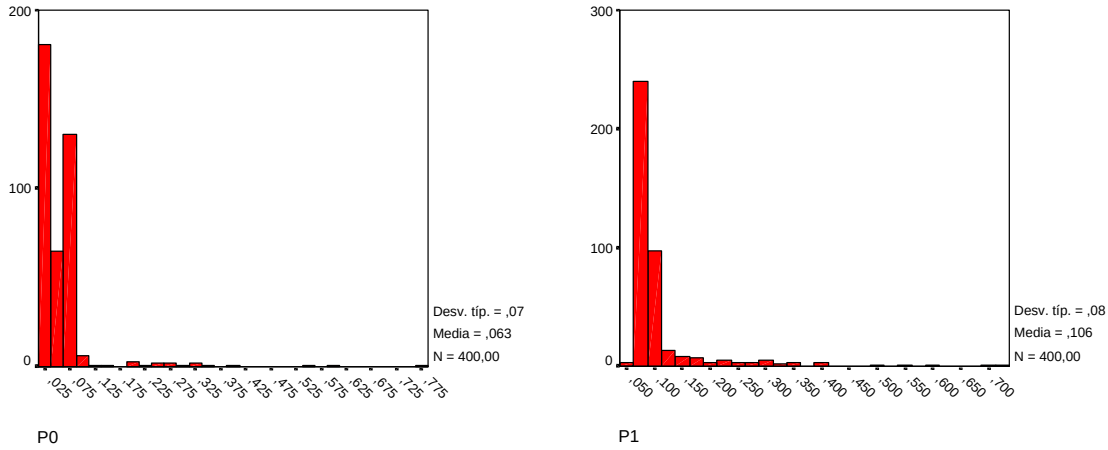


Figura 4.22. Histograma de los tiempos de lectura para P0 y P1

Esto aclara aún más lo que ya se expone en las gráficas de la figura 4.21. Los tiempos de lectura del procesador 1 son mayores que los tiempos de lectura del procesador 0. ¿A qué se deberá esto?

Para intentar explicarlo vamos a representar, en primer lugar, los instantes de petición de cada uno de los procesadores.

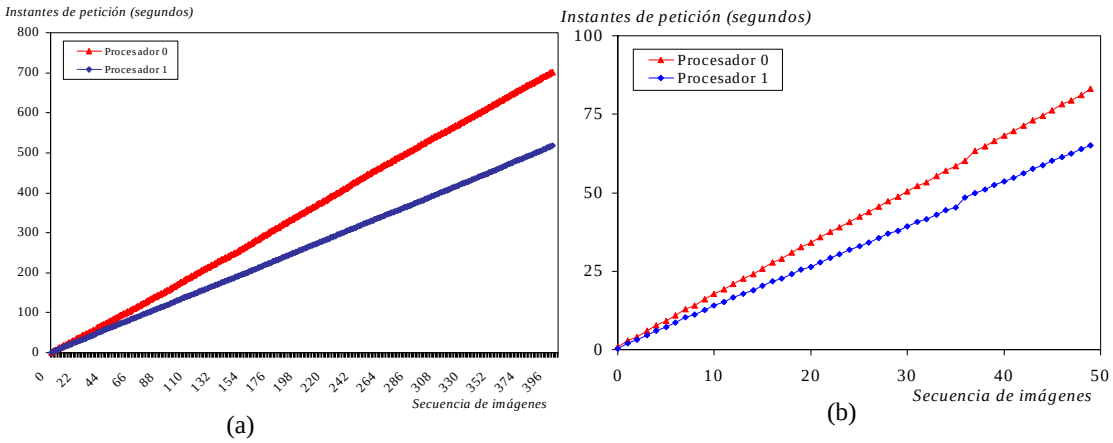


Figura 4.23. Instantes de petición. (a) Secuencia completa, (b) 50 imágenes

En estas gráficas queremos mostrar los instantes de tiempo en los que cada uno de los procesadores pide leer su trozo de datos. En la figura 4.23.a representamos estos instantes para cada una de las 400 imágenes de la secuencia, y en la figura 4.23.b sólo para las primeras 50 imágenes. Como vemos en la gráfica de la secuencia completa existe un desfase claro entre los procesadores. Curiosamente el procesador 1 siempre pide antes las imágenes, incluso al principio de toda la secuencia donde parece que van a la par, existen diferencias claras. Para esto hemos representado, en la gráfica b, un pequeño zoom de lo ocurrido y se ve claramente que P1 siempre va por delante.

¿Qué implican estos adelantamientos? Pues que P1 va a sufrir los retardos que suponen los accesos a disco por parte del servidor, tanto para los atributos de los ficheros como para los datos. Por el contrario, P0 se va a ver claramente beneficiado de los efectos de la cache del servidor, en la que, como ya hemos visto, se guardan todos los atributos, los directorios seguidos y los datos de los ficheros. El que primero solicita la información de un nuevo fichero sufre los accesos a disco. Esa petición implica que ese fichero se carga en la cache y que las posteriores peticiones se sirvan directamente de la cache sin tener que acceder a disco.

Pero, ¿A qué se debe el escalón que aparece en la gráfica 4.21 para los tiempos de lectura de P0? Para explicar este fenómeno hemos realizado un exhaustivo estudio de la memoria y los buffers utilizados en distintas situaciones.

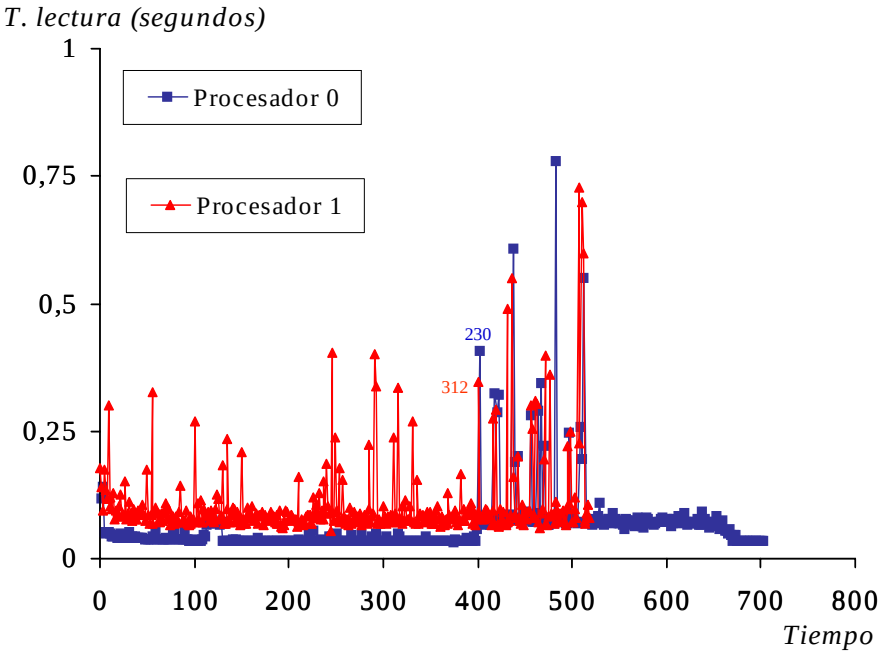


Figura 4.24. Tiempo de lectura de los procesadores 0 y 1

El cambio en los tiempos de lectura para el procesador 0 se produce justo al leer la imagen 230. En ese mismo instante el procesador 1 va a leer la imagen 312. Por lo tanto hay un desfase entre los procesadores de 82 imágenes, es decir, de unos 49 Megabytes de datos. Así, podíamos pensar que la cantidad de datos que cabe realmente en la cache del servidor está en torno a los 50 MB, y por tanto, cuando el procesador 0 quiere leer la imagen 230, ya no está, porque debido al adelantamiento de P1, esa imagen ya fue sacada de la cache para traer las nuevas imágenes de P1.

La manera de confirmar esta hipótesis es representar la cantidad de memoria y el tamaño de los buffers utilizados por el servidor (la myrinet1) durante todo el proceso, durante todo el tiempo en el que se está ejecutando el codificador y los procesadores 0 y 1 están realizando la compresión. En la siguiente gráfica (figura 4.24) representamos estos valores, que han sido tomados con la ejecución, en el servidor, del comando top. Los datos están expresados en Kilobytes.

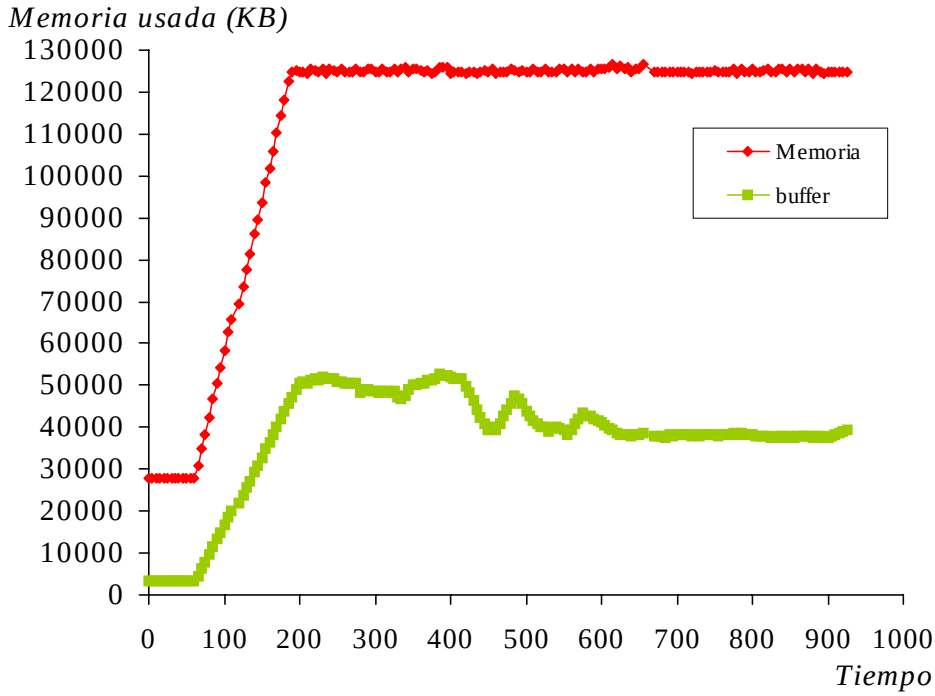


Figura 4.25. Memoria utilizada

Se han tomado los datos de la memoria total utilizada y del tamaño del *buffer-cache* utilizado desde, justo antes de empezar el programa de codificación, hasta el momento en el que el programa ha acabado. Con respecto a la memoria total, se han utilizado en la codificación unos 98 MB (el servidor dispone de 128 MB de RAM) y, con respecto al tamaño de los buffers, se ha alcanzado hasta un valor máximo de 49 168 KB, es decir, antes de empezar ya se tenían 3180 KB y durante el proceso de codificación se ha llegado hasta un valor de 52 348 KB. La diferencia en tamaño nos da el tamaño de la cache del servidor que, justamente se aproxima a los 50 MB que esperábamos. Además, la tendencia es la siguiente: al principio empieza aumentando de tamaño (ya que ambos procesadores están pidiendo imágenes), después hay algunas variaciones (debidas al desfase entre los procesadores y la necesidad de salida de ciertas imágenes para que se produzca al entrada de otras) y, finalmente hay una leve bajada que se mantiene hasta el final del proceso (esto se va a deber a que P1 ya ha terminado y queda P0 como único proceso solicitando imágenes, algunas de las cuales estarán aún en la cache).

Pero, para confirmar que efectivamente el tamaño de la cache de datos del servidor ronda alrededor de los 50 MB, hemos realizado este mismo estudio en otra situación distinta. Hemos ejecutado el codificador con dos procesadores después de que la plataforma se haya estando utilizando para otros programas, y por tanto, de partida, tanto la memoria total como los buffers ya estaban llenos antes de empezar.

Podemos ver en la figura 4.26, que la memoria total estaba casi llena, pero se ha seguido completando hasta llegar a 127 MB. El buffer también estaba lleno (91 MB), pero se ha empezado a vaciar hasta que se ha estabilizado al llegar a los 49 480 KB. A

partir de ahí, podemos ver claramente que la situación es idéntica a la de la gráfica 4.24. En esta ejecución también existe una diferencia similar a la anterior para los procesadores 0 y 1. El procesador 0 sufre un aumento en sus tiempos de lectura (un escalón) al leer la imagen 202. En ese instante el procesador 1 está leyendo la imagen 277. Por lo tanto, hay 75 imágenes de diferencia entre ellas, o lo que es lo mismo, 45 MB.

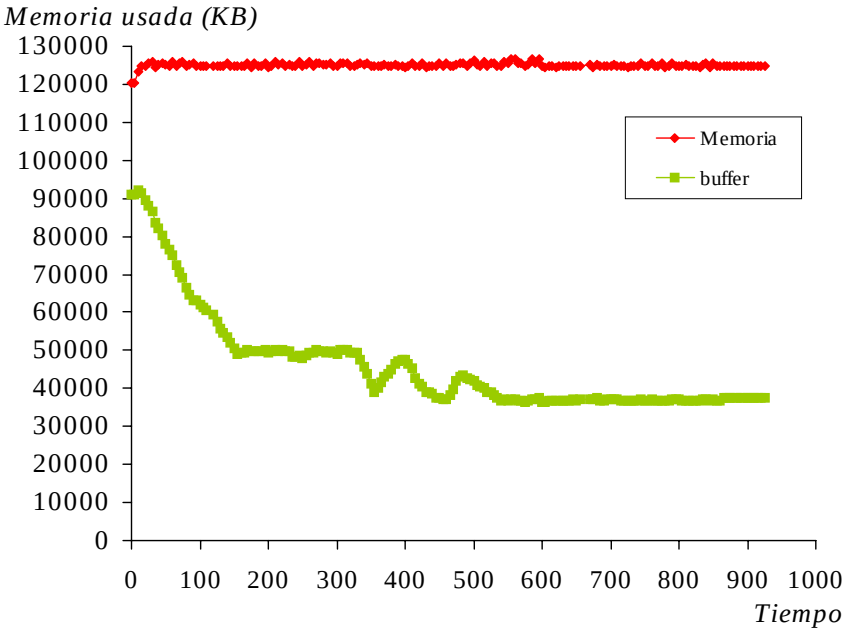


Figura 4.26. Memoria utilizada

Por lo tanto, podemos decir que el escalón que sufre P0 en sus tiempos de lectura, se produce cuando el buffer tiene un tamaño cercano a los 50MB. El servidor tiene 128 MB de memoria total, y de ahí unos 50 MB se van a dedicar a cachear los datos pedidos por los clientes. Sabemos que en la cache del servidor también se van a guardar otros elementos como inodos, nombres de directorio y retransmisiones, y como sabemos que la mayoría del espacio de la cache está reservado para datos pedidos por los clientes, podemos decir que el tamaño de la cache estará alrededor de los 64 MB.

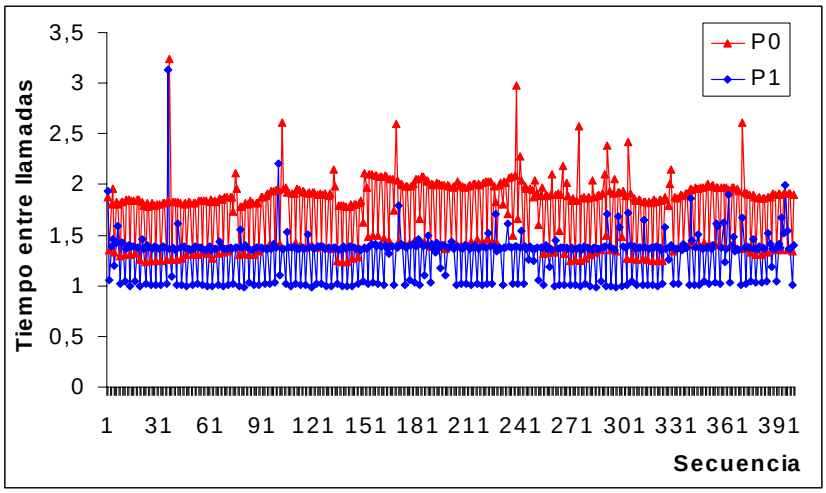


Figura 4.27. Tiempo entre peticiones de lectura

Aún nos queda otra diferencia importante entre P0 y P1 por explicar: si P0 tarda menos en leer, ¿Por qué acaba después que P1? Para responder a esta pregunta vamos a representar el tiempo entre lecturas, es decir, el tiempo transcurrido entre las peticiones de lectura (figura 4.27). En ese tiempo los procesadores codifican su trozo de datos y escriben el trozo codificado en el servidor. Por lo tanto se tendrá un tiempo de procesamiento y un tiempo de escritura.

Como vemos P0 presenta unos tiempos entre peticiones de lectura mayores que los de P1. En la gráfica se puede seguir el patrón de imágenes I, P y B, siendo las I las que menor tiempo de codificación conllevan y las B las que requieren mayor tiempo de codificación.

Veamos los tiempos de los que estamos hablando:

Tabla 4.8. Tiempos entre llamadas

	P0	P1
Σ T. entre llamadas	701,5479	518,0213
T. Escribir	12,3439	12,2972
T. Procesamiento	689,204	505,7241

De la gráfica anterior podemos deducir que el tiempo de escritura es bastante despreciable frente al tiempo de procesamiento, y además, para ambos procesadores es bastante similar, luego no puede ser la causa de las diferencias entre ambos.

Por ello, nos vamos a centrar en el tiempo de procesamiento que como vemos sí es bastante mayor para P0 que para P1.

Para saber exactamente los tiempos invertidos, representaremos también los histogramas correspondientes para este tiempo entre llamadas.

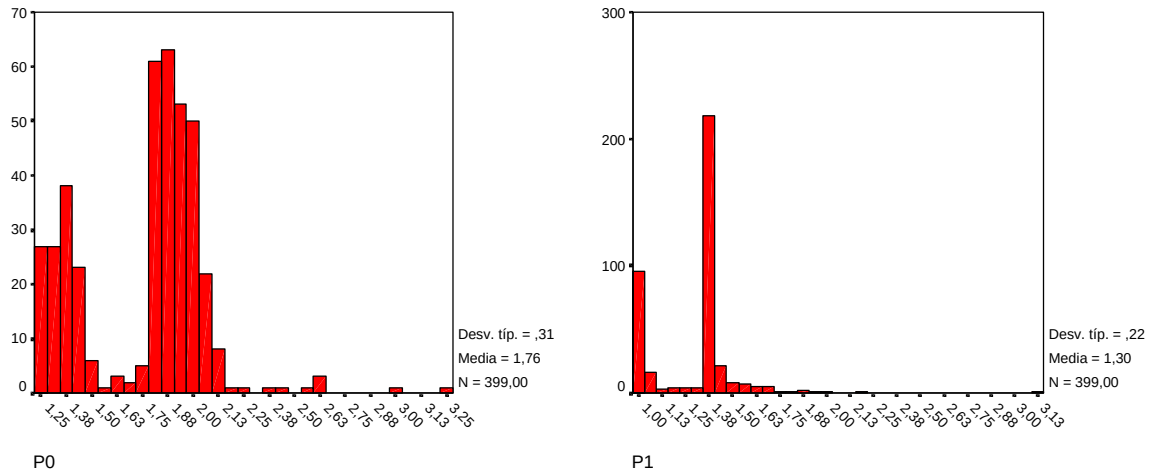


Figura 4.28. Histogramas para los tiempos entre llamadas

Podemos ver como existen tres intervalos de valores bien diferenciados para cada uno de los procesadores, esto se debe a lo que hemos comentado antes acerca de las imágenes I, P y B. Pero lo más destacable de las figuras anteriores es que, para el procesador 0, los valores de cada uno de esos tres intervalos están más dispersos, es decir, son intervalos más grandes mientras que para P1 la mayoría de los valores se centran en unos pocos valores. La media para P0 es ligeramente mayor que para P1 al igual que la desviación típica.

¿Pero por qué tarda P0 más tiempo en procesar sus datos? Pues debido a la inevitable dependencia con el contenido de la imagen. Para ver esta dependencia, hacemos el estudio detallado para 4 y 7 procesadores y veremos cómo se va haciendo cada vez menor.

Por lo tanto podemos concluir de este primer estudio con dos procesadores que en el caso de carga paralela mínima, donde participan dos procesadores en la codificación, hemos detectado un importante efecto de la cache del servidor que está perjudicando a un procesador (el que se adelanta) y beneficiando a otro (el que va detrás).

Por otro lado, el que un procesador acaba antes o después va a depender más del proceso de codificación en sí que del propio sistema de E/S. Recordemos que ambos procesadores tenían exactamente la misma cantidad de información que leer, pero los resultados de tiempos de lectura han sido bastante diferentes.

Veamos ahora otros dos casos también muy interesantes, el caso de carga media (4 procesadores) y el de carga máxima (7 procesadores). En estos casos los procesadores no van a tener la misma cantidad de información que leer, debido a la lectura solapada de datos. Veamos que ocurre ahora:

- *Ejecución con cuatro procesadores*. Planteemos un estudio similar al anterior, aunque más reducido, ya que las causas de los problemas se van a repetir. En este caso cada frame de la secuencia se divide en cuatro trozos iguales a codificar por cada uno de los procesadores (P0, P1, P2 Y P3). Los procesadores a los que le corresponden los trozos primero y último (P0 y P3) van a leer el trozo base a codificar más una franja de datos, mientras que los procesadores a los que les corresponde los trozos de en medio (P1 y P2) leerán los trozos base a codificar más dos franjas de datos.

En la siguiente tabla se muestran los segmentos de datos leídos por los procesadores y el tiempo invertido en la lectura.

Tabla 4.9. Bytes/imagen y Tiempo de lectura para cada procesador

PROCESADOR	BYTES/IMAGEN	T. LECTURA
P0	190080	19,5238
P1	224640	18,5785
P2	224640	18,7752
P3	190080	40,8905

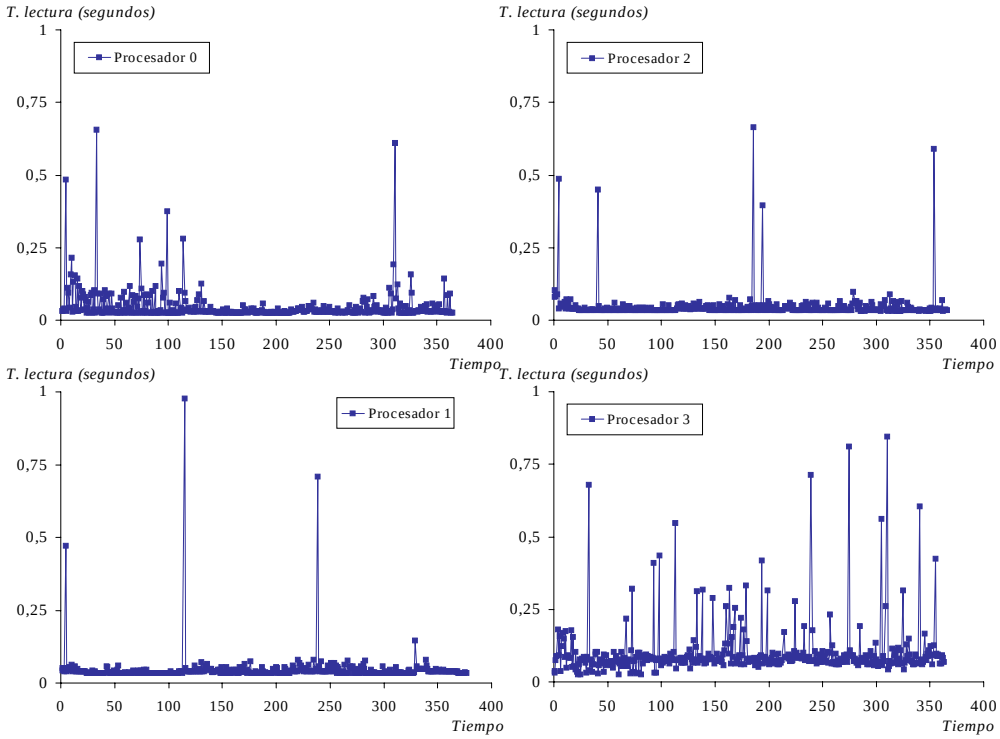


Figura 4.29. Tiempos de lectura para cada procesador

De la tabla 4.9 podemos deducir que, como en el caso anterior, la cantidad de datos leídos no implica directamente el tiempo invertido en leer. Se supone que si los procesadores P0 y P3 tienen exactamente la misma cantidad de datos que leer, deberían tardar más o menos el mismo tiempo en hacerlo, pero resulta que P3 invierte más del doble en leer que P0. Además, los tiempos de lectura de P0 y P3 deberían ser menores que los tiempos de lectura de P1 y P2, ya que estos dos últimos tienen más información que leer. Pero esto tampoco es lo que ocurre. P1 y P2 tardan menos en leer que los otros dos procesadores. Por lo tanto, algo está afectando y provocando estos resultados finales en los tiempos de lectura. Vamos a representar algunas gráficas que seguro nos aclararán la situación. Empezamos representando los tiempos de lectura/imagen para cada procesador (figura 4.29).

Representemos también los histogramas de cada uno de ellos que nos aclararán aún más lo que está pasando (figura 4.30).

Los tiempos de lectura individuales para cada una de las imágenes de la secuencia son muy similares para los procesadores P1 y P2, los valores de tiempo de lectura son muy regulares y existen un par de retransmisiones en cada caso. No ocurre lo mismo para P0 y P3, para los que los tiempos de lectura son mayores y más variables, sobre todo para P3. Además, en este caso todos los procesadores acaban casi al mismo tiempo. Lo que indica que se reduce la dependencia con el contenido de la imagen.

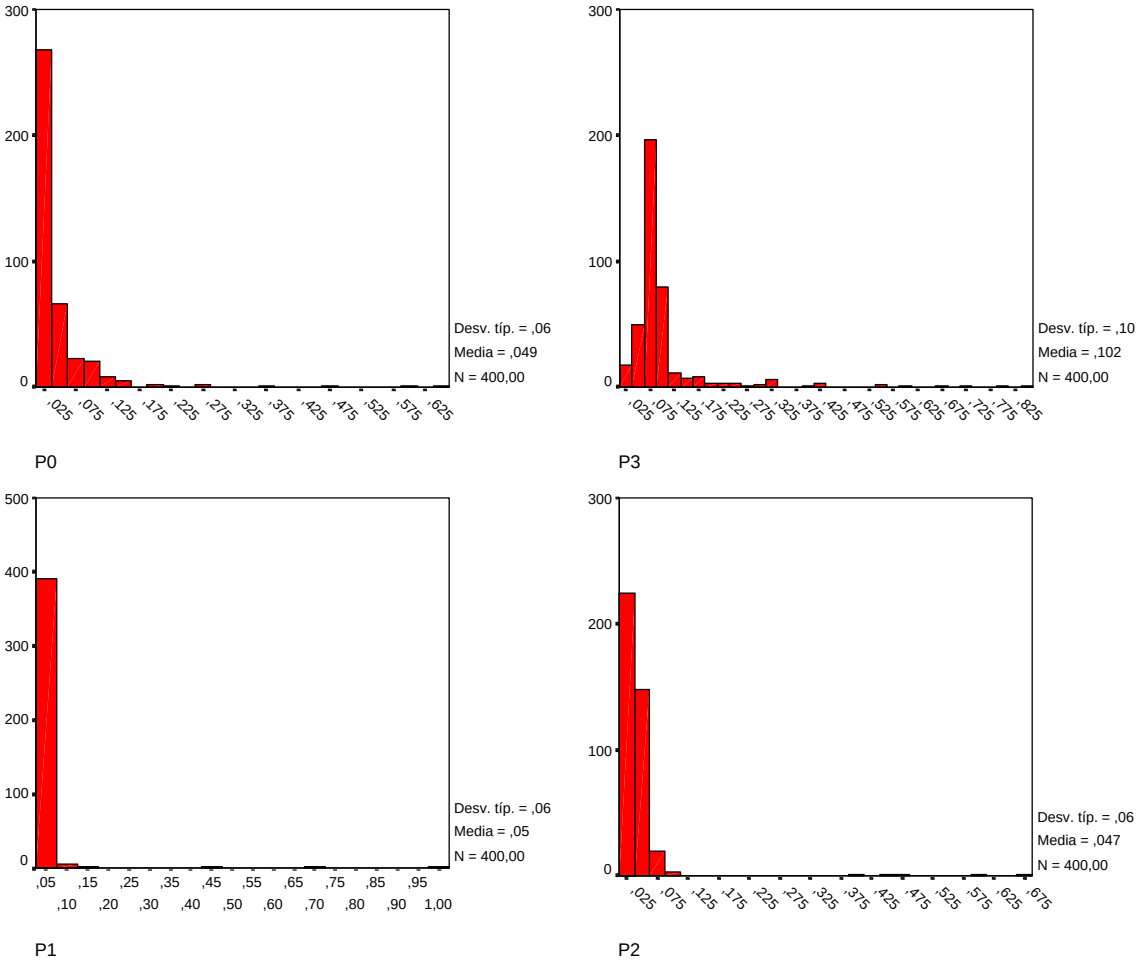


Figura 4.30. Histogramas para los tiempos de lectura de los distintos procesadores

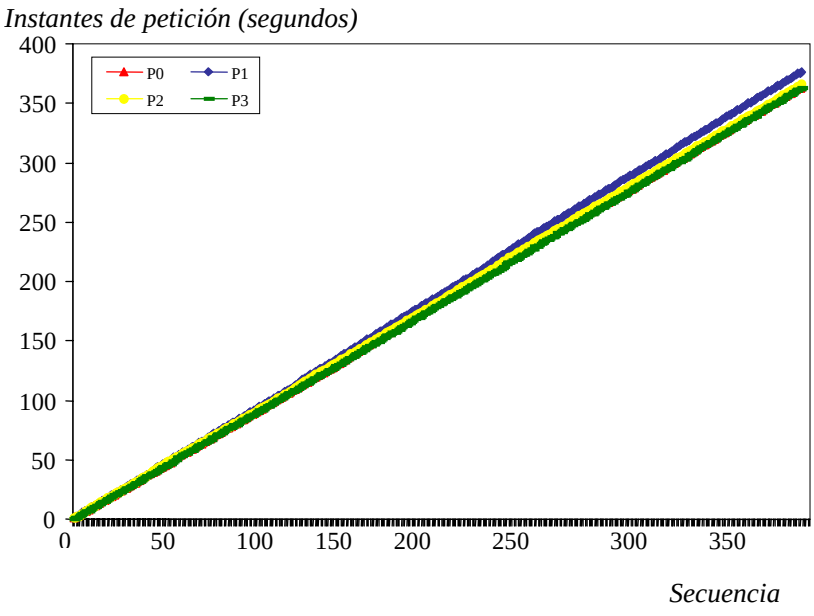


Figura 4.31. Instantes de petición para cuatro procesadores

Podemos ver cómo para la gráfica del procesador 3, los valores individuales de tiempos de lectura se mueven en un intervalo un poco más amplio que para el resto de procesadores, de ahí que la media por imagen sea mayor para P3 que para el resto. Además, la media y la desviación típica también son mayores.

Siguiendo el orden del estudio anterior, vamos a representar los instantes de petición (figura 4.31). P3 va a ser el que se adelante y, por eso, para cada imagen individual se van a tener unos tiempos de lectura mayores. P0 aparece como un caso intermedio en el que, al principio, los tiempos de lectura son mayores y luego ya son más bajos.

En este caso no existen los desfases que observamos en el caso de dos procesadores, pero podemos ver que va a ser P3 el que primero pide los datos de cada imagen. Existe solapamiento de peticiones entre P3 y P0, y con leve desfase aparecen las peticiones de P1 y P2.

Vamos a representar también el tiempo entre peticiones de lectura para completar el estudio, teniendo en cuenta, que los cuatro procesadores acaban casi al mismo tiempo.

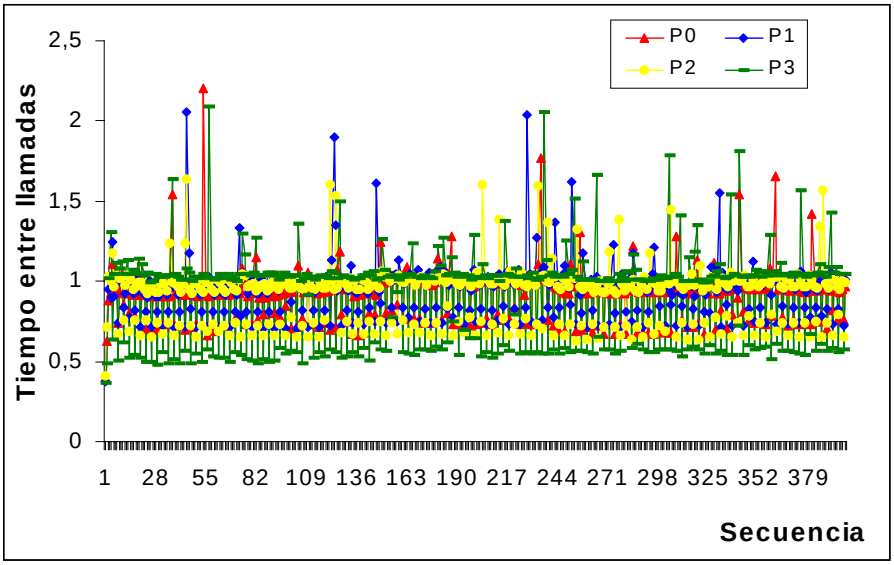


Figura 4.32. Tiempo entre peticiones de lectura para los distintos procesadores

En el caso de cuatro procesadores no existen las diferencias que existían para el caso de dos procesadores. Aquí los cuatro patrones están más o menos solapados, incluso si los mostramos dos a dos no se aprecian tantas diferencias.

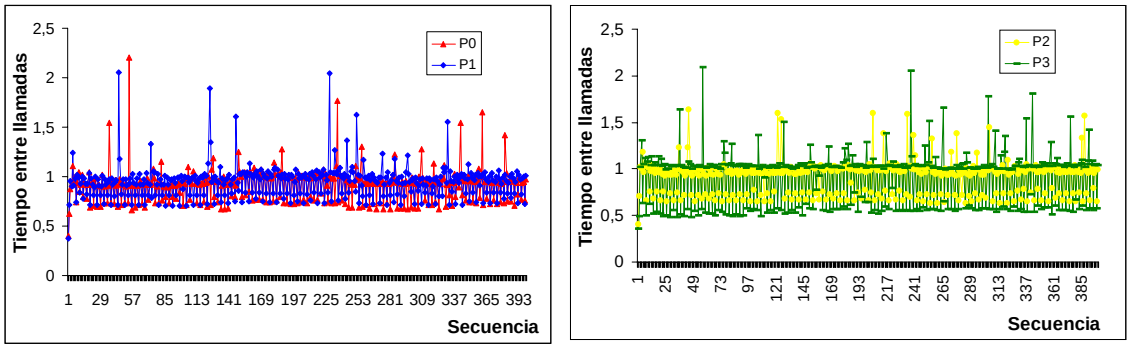


Figura 4.33. Tiempo entre peticiones de lectura

Las diferencias son muy pequeñas en este caso, lo que explica que los cuatro procesadores acaben más o menos al mismo tiempo. Como en el caso anterior vamos a representar los histogramas de los tiempos entre llamadas.

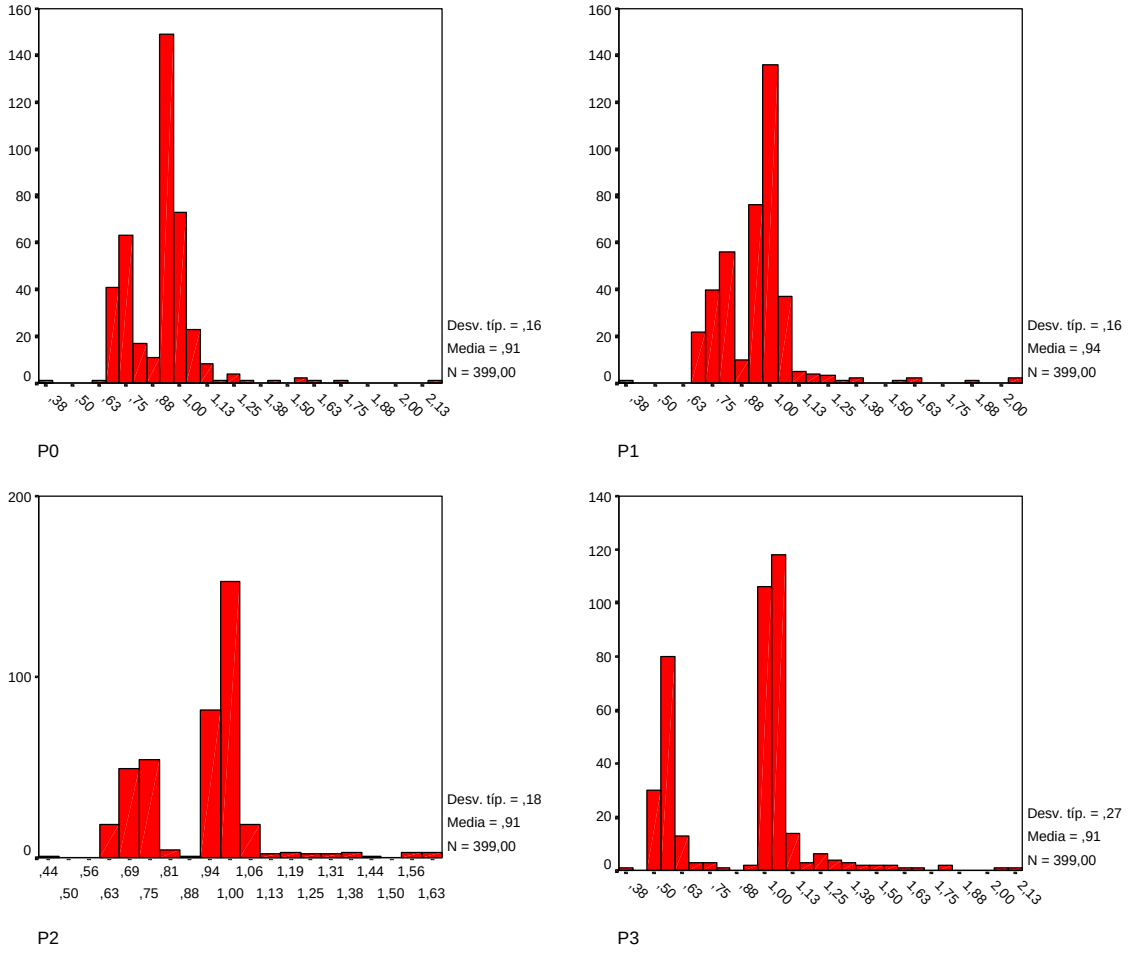


Figura 4.34. Histogramas de los tiempos entre peticiones de lectura

En la figura 4.29 podíamos ver cómo era el procesador P1 el que acababa el último. Esto se puede justificar perfectamente con el histograma representado ahora. La media de P1 es la mayor y los valores para las imágenes B son ligeramente mayores que

el resto. Como en el caso anterior de dos procesadores, vamos a desglosar este tiempo entre peticiones de lectura en sus componentes: tiempo de procesamiento y tiempo de escritura.

Tabla 4.10. Tiempos entre llamadas

	P0	P1	P2	P3
Σ T. entre llamadas	362,6379	375,3152	365,0285	362,5711
T. Escribir	11,2072	12,4484	12,4011	10,8311
T. Procesamiento	351,4307	362,8668	352,6274	351,74

El tiempo de escritura vuelve a ser bastante despreciable frente al tiempo de procesamiento, y además, para los cuatro procesadores es bastante similar, luego no puede ser la causa de las diferencias entre ambos. Por ello, nos vamos a centrar en el tiempo de procesamiento que como vemos sí es un poco mayor para P1 (el que acaba el último) que para el resto.

Por lo tanto, en este caso de cuatro procesadores aunque a cada procesador le corresponda una segmento de datos distinto a leer, vemos que los tiempos de lectura obtenidos no se corresponden con esas cantidades leídas, y que, por tanto, los resultados se ven influenciados mucho más por el comportamiento del sistema de ficheros que por la cantidad de datos asignada. El procesador que primero pide los datos es el que sufre los fallos de cache y los consiguientes accesos a disco.

- *Ejecución con siete procesadores*. En este caso cada imagen de la secuencia se divide en siete segmentos iguales a codificar por cada uno de los procesadores (P0, P1, P2, P3, P4, P5 y P6). Los procesadores a los que le corresponden los trozos primero y último (P0 y P6) van a leer el trozo base a codificar más una franja de datos, mientras que los procesadores a los que les corresponde los trozos de en medio (P1, P2, P3, P4 y P5) leerán los trozos base a codificar más dos franjas de datos. En la siguiente tabla se muestran los bytes/imagen leídos por cada procesador y el tiempo invertido en la lectura.

Tabla 4.11. Bytes/imagen leídos y tiempos de lectura invertidos

PROCESADOR	BYTES/IMAGEN	T. DE LECTURA
P0	123430	15,6521
P1	157990	19,7516
P2	157990	22,522
P3	157990	22,007
P4	157990	44,767
P5	157990	47,0105
P6	123430	48,3713

En la tabla anterior podemos ver que para los tiempos de lectura existen diferencias considerables entre los distintos procesadores. Tres de los siete procesadores (P4, P5 y P6) devuelven un tiempo de lectura mucho mayor (más del doble) que el resto, y, además, uno de esos tres procesadores tienen que leer un trozo menor. Vamos a representar estos tiempos de lectura. Como en los casos anteriores, nos apoyaremos en

gráficos para explicar los datos obtenidos y reflejar las distintas situaciones por la que pasa cada procesador.

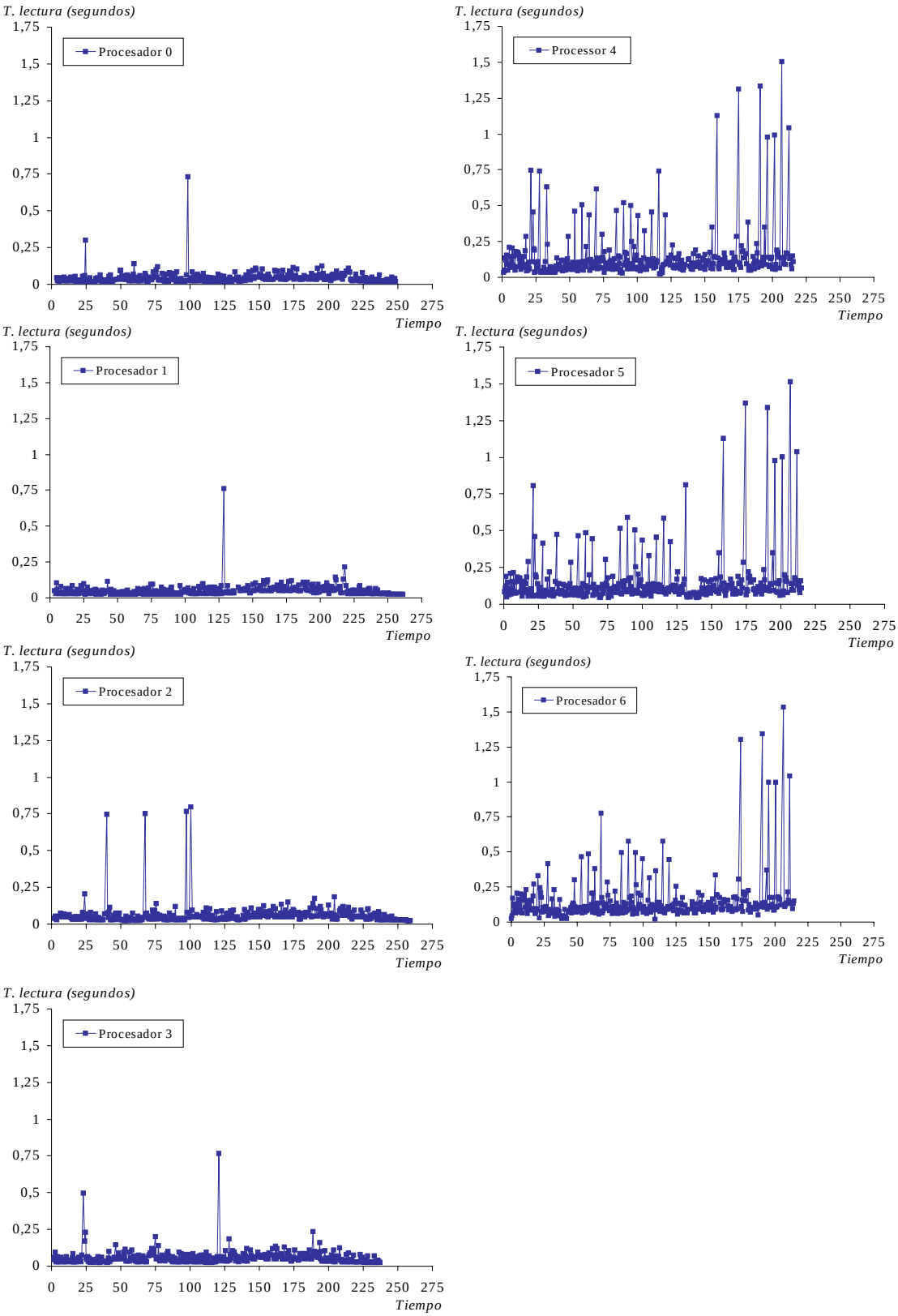


Figura 4.35. Tiempos de lectura para cada procesador

Hemos tenido que cambiar el eje vertical en las gráficas anteriores, ya que para siete procesadores existen muchas más retransmisiones, e incluso se ha llagado hasta

más de 1,4 segundos al expirar el tiempo de espera dos veces. Precisamente son los procesadores P4, P5 y P6 los que han sufrido estas retransmisiones, lo que lleva también a aumentar los tiempos totales de lectura. Ya hemos comentado lo inadecuado del valor por defecto para timeo, y será un problema que tendremos que solucionar. Los tiempos de lectura ofrecidos pon los procesadores P0, P1, P2 y P3 son menores que los obtenidos con el resto de los procesadores para cada una de las imágenes en particular, de ahí las diferencias finales

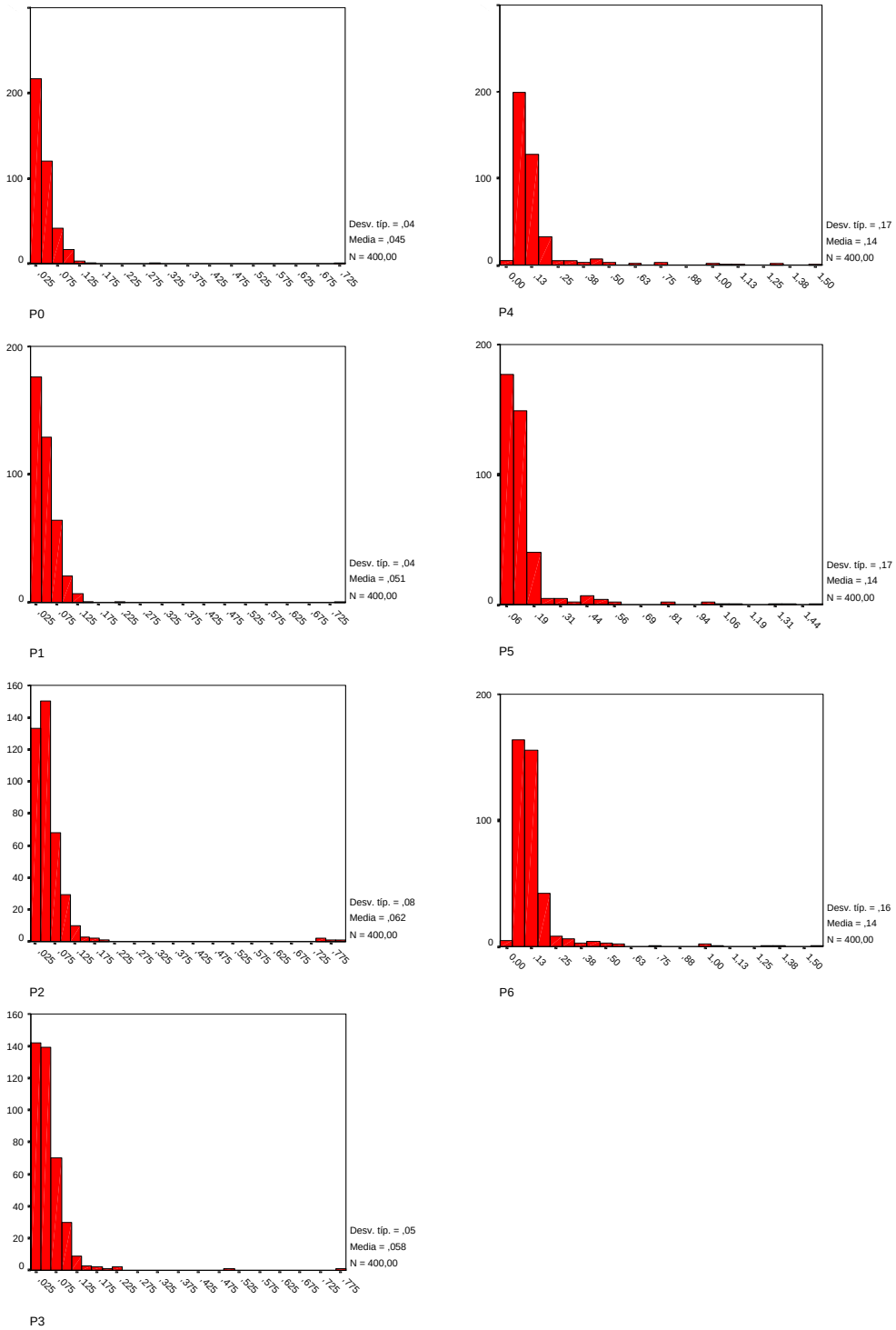


Figura 4.36. Histogramas de los tiempos de lectura para los 7 procesadores

En la figura anterior representemos los histogramas correspondientes a estos tiempos de lectura para ver el número de imágenes dentro de cada intervalo de valores.

Hay varias cosas interesantes que resaltar. En primer lugar, los valores medios de latencia y la desviación típicas son mayores para los procesadores P4, P5 y P6, como era de esperar. Además, para los procesadores P4, P5 y P6, es necesario representar valores mucho más altos debido a las retransmisiones. Por lo tanto en este caso, aparte de que los intervalos para P4, P5 y P6 son mayores, existe el fenómeno crítico de las retransmisiones que hinchan bastante los valores de latencia obtenidos. Vamos a representar los instantes de petición para cada uno de los procesadores para completar la explicación de estas diferencias en los tiempos.

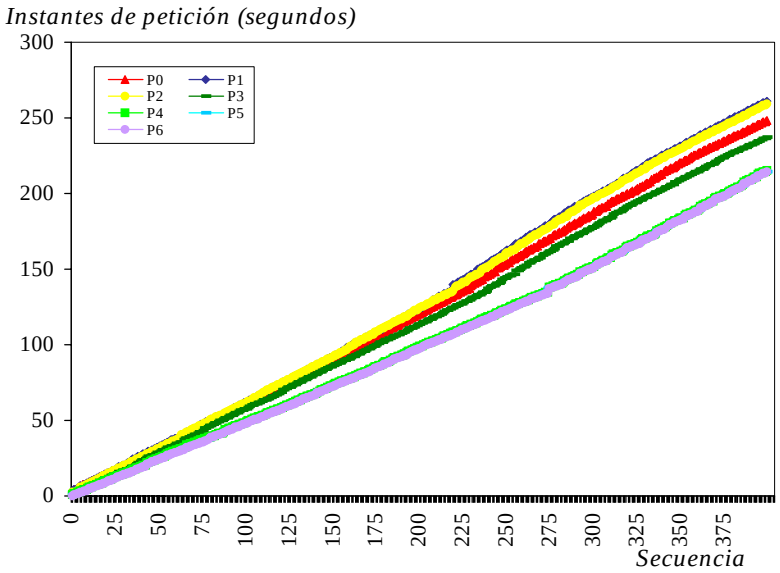


Figura 4.37. Instantes de petición

De nuevo vemos el desfase existente entre los procesadores P4, P5 y P6 que son los que se adelantan, y los procesadores P0, P1, P2 y P3 que van a pedir siempre después y se van a beneficiar de los efectos de la cache del servidor, y por lo tanto, van a conseguir unos tiempos de lectura menores. En la gráfica 4.35 podemos ver también que ahora no todos los procesadores acaban al mismo tiempo, lo que nos indica que el tiempo entre llamadas no va a ser tan similar como en el caso de cuatro procesadores anterior.

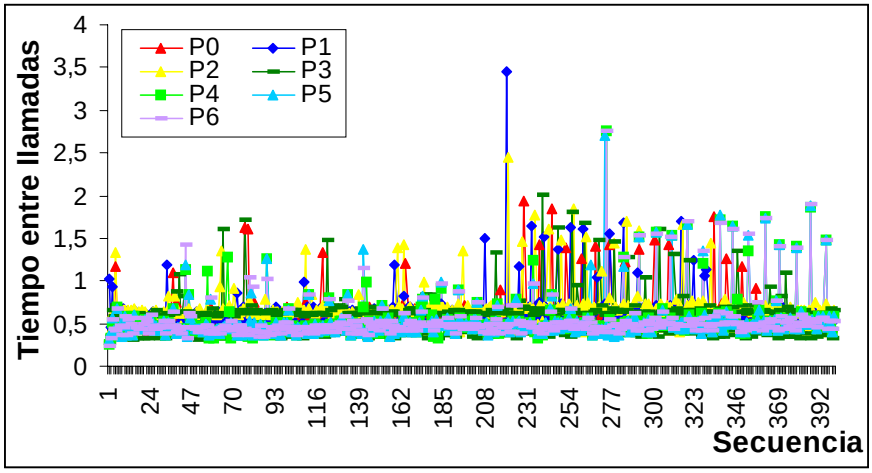


Figura 4.38. Tiempo entre peticiones de lectura para todos los procesadores

La anterior figura muestra precisamente estos tiempos entre llamadas. Representaremos, por un lado, en una misma gráfica los datos de todos los procesadores

y, por otro, en otra gráfica los datos para los procesadores P0 y P6 que son los que ofrecen los tiempos menor y mayor, respectivamente, y, además, P0 acaba después que P6.

Como es bastante difícil sacar conclusiones sobre esta gráfica debido a la cantidad de datos representados, veamos que ocurre si solo mostramos los datos para los procesadores 0 y 6

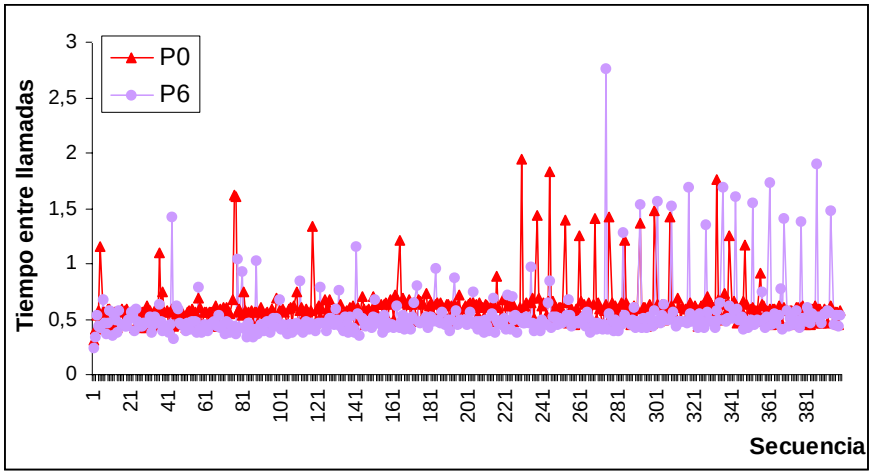


Figura 4.39. Tiempo entre peticiones de lectura para dos procesadores

En este caso sí se ve más claramente que el procesador 0 sigue un patrón con tiempos ligeramente superiores a los tiempos del procesador 6, lo que demuestra que efectivamente el procesador 0 acabe después.

Tabla 4.12. Tiempos entre llamadas

	P0	P1	P2	P3	P4	P5	P6
Σ T. entre llamadas	244,4588	258,129	256,8803	235,6581	214,1064	214,2739	214,0828
T. Escribir	26,9572	25,2018	27,1056	25,7353	24,4161	25,6489	24,5077
T. Procesamiento	217,5016	233,0234	229,7747	209,9228	189,6903	188,625	189,5751

Efectivamente, queda demostrado el hecho de que si unos procesadores acaban antes (P4, P5 y P6) que otros, se debe a un menor tiempo de procesamiento, que tendrá que ver, por tanto, con el contenido de la secuencia.

Veamos, por último, los histogramas de los tiempos entre llamadas para saber los valores más característicos para este tiempo de procesamiento (ver figura 4.40). Debemos fijarnos con atención en las diferencias en las escalas tanto en el eje vertical como en el horizontal, y así, entenderemos la causa de las diferencias ente los procesadores. Además, el valor medio obtenido de estos tiempos entre llamadas es mayor para los procesadores P0, P1, P2 y P3, como era de esperar.

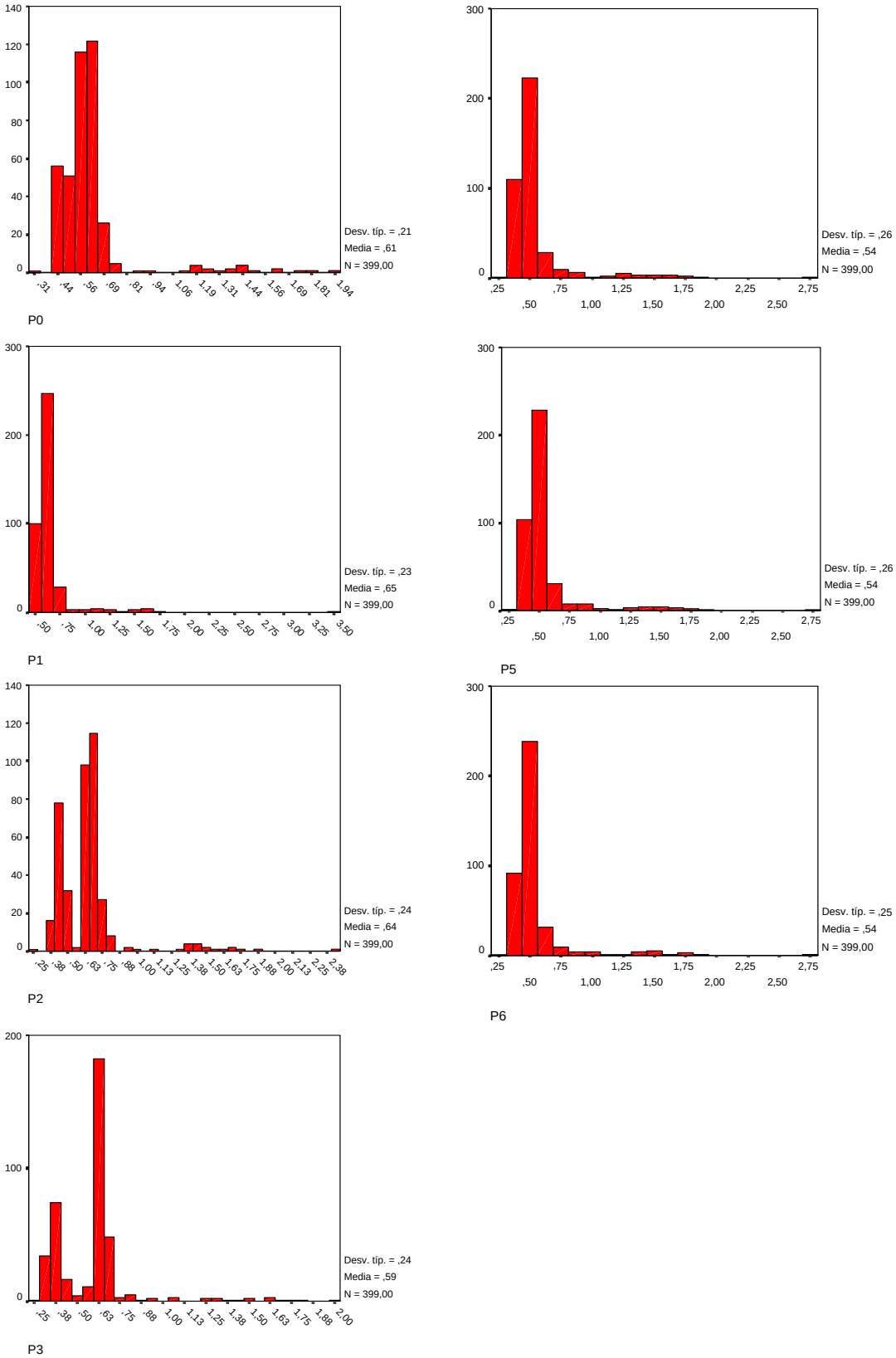


Figura 4.40. Histogramas representativos de los tiempos entre llamadas

Las diferencias obtenidas en estos tiempos de procesamiento (al igual que pasaba con el caso de cuatro procesadores) no son tan grandes como en el caso de dos procesadores. La dependencia del contenido de la imagen se ha ido suavizando porque se codifican trozos menores y más o menos tardan el mismo tiempo.

Una vez estudiados estos tres casos vamos a resumir lo ocurrido. Antes de nada, no podemos olvidar que existen otros patrones de carga también representativos del tráfico multimedia, como ya vimos, ¿Qué pasará en esos casos?. De la gráfica 4.16 (donde se representaban los distintos tiempos de lectura para distintos patrones de carga) podemos decir que existe una diferencia importante entre dos tipos de patrones:

a) Los que conllevan la lectura de segmentos de datos
b) Los que conllevan la lectura de la imagen completa

Los tiempos invertidos con estos patrones básicos aumentaban un poco más si además también se realizaban operaciones de escritura.

Todo el estudio sobre los efectos de la cache lo hemos realizado con el caso del patrón de carga de lectura solapada de segmentos de datos. Otra forma de leer era la lectura sin solapamiento. De la gráfica 4.16, podemos decir que existía muy poca diferencia en los resultados obtenidos al realizar lectura solapada de datos o lectura no solapada. Ahora podemos explicar esa similitud, que se debe principalmente a los efectos de la cache del servidor. Cuando un procesador pide abrir un fichero para leer, el servidor cargará en su cache el fichero completo, además de otra información útil (su localización y sus atributos). Cuando recibe peticiones del resto de procesadores ya no tiene que acceder a disco, sino que solo tendrá que tomar los datos de su cache. Gráficamente hemos demostrado que las diferencias en los tiempos obtenidos por los procesadores no se deben a la cantidad de datos que tienen que leer sino a otros fenómenos como los adelantamientos a la hora de pedir la información. Por lo tanto, si realizásemos un estudio con este otro patrón de carga los resultados serían muy similares

¿Qué va a ocurrir cuando todos leen los ficheros completos, es decir, exactamente la misma cantidad de información? Pues que va a haber muchas más peticiones al servidor que van a empeorar y aumentar los tiempos de lectura. Además va a haber muchas peticiones de la misma información, y el servidor, como es sin estado y no lleva ningún control, enviará las mismas respuestas repetidamente.

Prefetching de datos

En el servidor se realiza *prefetching* de datos, cuando un cliente realiza una petición de lectura de datos de un determinado fichero, y el servidor trae de disco esos datos a su cache, envía la respuesta al cliente, y, además, se trae también a la cache el resto de datos del fichero. De ahí la similitud de resultados entre los patrones 1 y 5, es decir entre la lectura con solapamiento o sin solapamiento. Si en la lectura sin solapamiento no existiera este *prefetching*, cada procesador que pide datos sufriría los accesos a disco, porque cada uno de ellos está pidiendo datos distintos, pero no es así, ocurre lo mismo que con el patrón 1, los procesadores más adelantados tardan más en leer los datos (debido a los fallos de cache y accesos a disco) y los más atrasados tardan menos (porque toman los datos directamente de la cache).

Por lo tanto si resumimos los efectos del *caching* y *prefetching* detectados podríamos decir:

1. *En el lado del servidor*: es de gran utilidad la cache de atributos y rutas de directorio ya que todos los ficheros están ubicados en el mismo directorio. También lo es la cache de datos, ya que todos los procesadores acceden a los mismos ficheros de datos. Aunque, en el caso de que existan adelantamientos por parte de algún procesador, va a ser éste el que sufra los accesos a disco, por ser el que primero realiza las peticiones sobre un fichero nuevo. Con la lectura de trozos de datos existen más diferencias entre los tiempos de lectura invertidos por los procesadores

2. *En el lado del cliente*: se está utilizando de forma muy útil la cache de atributos, por lo que se pueden hacer peticiones de lectura sucesivas sin pedir continuamente información acerca de los atributos. No se está utilizando la cache de datos, ya que solo se necesitan los datos leídos una vez y ya no se vuelven a leer más, por lo que este servicio no se está explotando. Se utiliza la operación *read-ahead*, que hace más efecto cuantos más datos se lean, es decir, hasta que no se han leído bastantes datos de forma secuencial no se detecta lo que está pasando, por lo que con la lectura de trozos de datos casi que no se pone en práctica (tan solo cuando participan 2 ó 3 procesadores). En cambio con la lectura de ficheros completos sí se pone en práctica.

Por lo tanto existen algunas causas del problema de escalabilidad existente en NFS:

- Los adelantamientos por parte de algunos procesadores, provocan aumentos en los tiempos de lectura debido a los fallos de cache y los accesos a disco
- La sobrecarga del servidor, al pedir todos los clientes las mismas grandes cantidades de información secuencial, va a implicar esperas cada vez mayores.

El *prefetching* del cliente ayuda bastante, sobre todo en la lectura de ficheros completos, y de hecho los resultados sin *prefetching* serían peores, como se mostrará en el capítulo de resultados finales. Pero no se explota demasiado. Con el *read-ahead* el cliente adelanta unos pocos datos a su cache antes de que se necesiten realmente por la aplicación. Pero va a haber casos en los que pasará inadvertida. Por tanto, vemos que es un gran problema el acceso inicial a los datos, es decir, acceder por primera vez a datos a los que el servidor aún no ha tenido que acceder ninguna vez, porque no hayan sido requeridos por ningún procesador, y, por tanto, el primero que los pida, sufrirá el fallo de cache del servidor y el acceso a disco.

En el siguiente capítulo, veremos cómo se les puede dar solución a este problema fundamental, planteando una técnica de *prefetching* a través de ficheros, en el cliente, que va a ser capaz de cargar en memoria el siguiente fichero que se va a solicitar, antes de que lo necesite la aplicación, con lo que se van a solucionar los problemas de primeros accesos.

CAPÍTULO 5

PLANTEAMIENTOS DE LAS MEJORAS A REALIZAR EN EL CLIENTE NFS

5.1. REFLEXIÓN INICIAL

En el capítulo 4 hemos realizado el análisis del comportamiento de NFS con aplicaciones multimedia paralelas sobre un cluster de alta velocidad.

En este análisis, partimos de una configuración inicial del cluster con 8 máquinas conectadas mediante dos conmutadores myrinet. Es una arquitectura cliente-servidor típica, donde una máquina es el servidor NFS y el resto los clientes. Como aplicación, utilizamos la codificación en paralelo de vídeo MPEG-2, para ello disponemos de una secuencia de entrada de 400 imágenes. Cada imagen consta de tres ficheros separados (uno de luminancia, y dos de crominancia). Cada uno de los ficheros se divide en franjas horizontales y se asigna a los distintos procesadores cliente, participantes en el proceso de codificación.

Cada uno de los procesadores, tras un proceso previo de sincronización y cálculo inicial de parámetros, ejecuta un bucle, de tamaño igual al número de imágenes a codificar, y realiza los siguientes pasos: lee trozo de imagen o imagen completa, codifica su trozo de datos y escribe los datos codificados en el servidor. Tras este bucle, presenta los tiempos de lectura y escritura invertidos para el total de la secuencia.

Realizamos distintas pruebas variando, primero, parámetros propios de NFS, después, representamos, los tiempos medios obtenidos, y se observan los primeros problemas:

- Importantes diferencias entre el caso secuencial (1 procesador) y los paralelos. En el caso secuencial los mejores tiempos se obtienen con rsize =

4096 bytes mientras que para el resto de casos esto no es así. Los valores obtenidos con 4096 bytes son los peores de todos.

- Para los casos paralelos, a partir de 5 procesadores, los tiempos de lectura no disminuyen con el aumento en el número de procesadores, sino todo lo contrario. Por lo tanto, aparecen problemas de escalabilidad. Además, esto ocurre para todos los valores de rsize, y de forma más pronunciada, para 4096 bytes.

Algo similar ocurre para los tiempos de escritura tomados para distintos valores de wsize y distinto número de clientes. Pero, en las gráficas, se puede ver claramente que se tarda menos en escribir que en leer. Al escribir, cada procesador lo hace en un fichero independiente, después el servidor se encargará de concatenarlos al final de todo el proceso. Mientras que en la lectura de datos, todos los procesadores están intentando acceder a los mismos ficheros y además, casi al mismo tiempo, por lo que este caso es mucho más serio.

Representamos, también, los tiempos obtenidos por procesador (ya que la media de todos puede ocultar lo que ocurre en realidad), centrándonos ya en el tiempo de lectura y en rsize=4096 bytes.

Si tomamos los tiempos de lectura obtenidos con tan solo 2 procesadores vemos que hay cosas interesantes. Con 2 procesadores ambos leen la misma cantidad de información y sin embargo se obtienen tiempos de lectura muy diferentes (25,14 segundos frente a 42,4 segundos). ¿A qué se deben esas diferencias? Para dar respuesta a esta pregunta hemos representado gráficamente los tiempos de lectura para cada una de las 400 imágenes, por procesador, los instantes de petición y los tiempos entre peticiones. Hemos visto que el procesador con tiempo de lectura mayor es el que va siempre por delante, es decir, el que pide primero los ficheros al servidor NFS, y por lo tanto el que va a sufrir los accesos a disco.

Resumidamente, el funcionamiento del servidor NFS es el siguiente: al recibir una petición de un fichero, comprueba si tiene los datos almacenados en su cache, si es así los envía inmediatamente y sino accede al disco, vuelca los datos en la cache y los envía. Por lo tanto, el procesador que pida datos al servidor, de un fichero ya pedido anteriormente, va a obtenerlos casi inmediatamente, porque el servidor tiene esos datos en la cache.

Además, hay que señalar que el cliente NFS hace uso del llamado *read-ahead*, una especie de *prefetching* secuencial en el mismo fichero, es decir, el cliente no sólo hace una petición del rsize bytes sino que, en el momento en el que el VFS detecta lectura secuencial del fichero, se activa el mecanismo para realizar de forma consecutiva varias peticiones de datos al servidor, así el cliente adelantará a su cache algunos bloques de datos, para evitar posteriormente las correspondientes peticiones al servidor.

Con los demás casos paralelos ocurre algo similar, existen diferencias muy importantes entre los tiempos de lectura obtenidos. La cantidad de bytes leídos no implica directamente el tiempo invertido en leer. Para el caso de 7 procesadores las diferencias son aún mayores. El tiempo de lectura menor son 15,65 segundos y el

mayor 48,3713 segundos. Curiosamente, los dos procesadores que ofrecen estos tiempos leen la misma cantidad de información, pero de nuevo, el que pide primero sale perjudicado.

También hemos realizado un estudio sobre los efectos del parámetro *timeo* (que mide el tiempo de espera de la respuesta del servidor antes de volver a mandar la misma petición). Este parámetro tiene un valor por defecto de 0,7 segundos (un tiempo demasiado grande para las actuales redes de alta velocidad). Hemos querido medir los tiempos en que se sirven las peticiones, es decir, cuanto tarda el servidor en responder una petición. Hemos visto que el 95% de las peticiones se atienden en menos de 0,001 segundos y que el 99% en menos de 0,01 segundos, por lo que este parámetro está bastante desfasado para las necesidades de comunicación más recientes. El caso de 7 procesadores es el que presenta mayor número de retransmisiones, por lo que eliminarlas implicaría también una reducción considerable de tiempo. Habrá que tenerlo en cuenta para las nuevas propuestas.

Hemos realizado también las mismas pruebas para distintas topologías de red, ya que queremos asegurarnos que los problemas detectados no están relacionados con la red utilizada. Pero el comportamiento es el mismo, por lo que la red no es la causa del problema.

Planteamos ahora medidas realizadas (para el patrón 1 de carga y la configuración A de la red) para conocer el ancho de banda utilizado por el servidor al enviar sus datos a los clientes. Lo hemos calculado para los casos de 1, 4 y 7 procesadores. Hemos tomado los tiempos invertidos en el proceso completo para la myrinet1. Si calculamos el número de llamadas de lectura respondidas, obtendremos la cantidad de datos que salen del servidor.

$$Caudal\ de\ salida = \frac{\text{bits transmitidos}}{\text{tiempo invertido}}$$

Con este valor calculamos el tanto por ciento del ancho de banda ocupado considerando 1,28 Gbps como el 100%. En la siguiente tabla mostramos los resultados.

	1P	**4P**	**7P**
Mbits transmitidos	1120	2339	2443
Tiempo invertido (s)	1385,1097	368,0931	236,04
Caudal de salida (Mbps)	0,809	6,3544	10,35
%Ancho de banda utilizado	0,0632	0,496	0,8085

Sorprendentemente, el ancho de banda utilizado es bajísimo, pero, vamos a plantear una medida más realista. Como sabemos que el tráfico NFS es a ráfagas, veamos lo que ocurre para la ejecución con 7 procesadores si no consideramos el proceso completo, sino solo un pequeño intervalo de tiempo (0,00833 s). Espacio comprendido entre dos peticiones lookup al servidor. En este intervalo sólo se hacen peticiones de lectura de datos, así que lo que salga de la myrinet1 serán datos dirigidos a responder las distintas peticiones. De la myrinet1 salen 25 mensajes de respuesta de 3720 bytes cada uno, lo que hacen un total de 93 000

bytes (744 000 bits). Si calculamos el caudal de salida ahora, obtendríamos 89,2943 Mbps y una utilización del ancho de banda de **6,97%.** Y esto es lo mejor que podemos esperar, ya que hemos tomado una situación de máximo envío de peticiones por parte del servidor.

Podemos decir, por tanto, que el canal de salida está infrautilizado. También comentar en este momento de arranque de las propuestas de mejora, que en múltiples trabajos realizados por la Universidad de Lyon, donde sólo se utiliza BIP y otros APIs ideales para Myrinet, sólo se ha llegado al 50% de la utilización del canal [94]. Con lo que no queremos plantear un ideal imposible de utilización máxima del canal, sino que vamos a plantear una propuesta real de mejora de la operación de lectura en NFS que llevará a una reducción total de tiempos en los clientes. El primer efecto de colapso del servidor se va a ver reflejado en el aumento en el tiempo de espera de los clientes, éstos van a necesitar leer cierta cantidad de datos antes de seguir con el procesamiento de la aplicación.

También probamos otros patrones de acceso a los datos compartidos, por ello definimos nuevas formas de lectura/escritura representativas de otras aplicaciones multimedia, simulamos así en nuestra plataforma y con nuestro codificador paralelo el comportamiento de E/S de otras aplicaciones.

Al representar los tiempos de lectura con estos patrones destacamos lo siguiente:

- Los peores resultados se obtienen con el patrón 2 (todos los procesadores leen las imágenes completas) donde la escalabilidad no existe y los resultados son imposibles.
- A continuación está el patrón 3 (igual que el anterior, pero sin operaciones de escritura) , un poco mejor que el 2, pero con la misma tendencia
- Con los patrones 1 y 4 (lectura de trozos solapados de datos, con y sin escritura) se obtienen tiempos muy similares, de ahí queda demostrado que las operaciones de escritura afectan levemente al proceso, aunque luego las tendremos en cuenta cuando hablemos del espacio utilizado de la cache.
- Si comparamos los patrones 1 y 5 (lectura de trozos sin solapamiento), vemos que existen pocas diferencias, por lo tanto, los problemas de escalabilidad no se van a deber a la lectura solapada. Lo mismo ocurre con los patrones 4 y 6.

Por lo tanto, existe un problema de escalabilidad claro en el servidor NFS, creemos que mejorable si se optimiza la operación de lectura (causa principal del problema). Hemos visto que la red no supone ningún problema, y que si probamos distintas formas de acceso a la información, el caso más problemático es aquel en el que todos los clientes quieren acceder a leer todos los ficheros completos de datos y escribir los resultados en el servidor.

También, quiero aclarar que las pruebas que se presentan en el capítulo 5 no han sido las primeras que se realizaron, ni las únicas. Ya hemos comentado que se han hecho otras evaluaciones para comparar otros elementos del sistema que no ponemos para no alargar en exceso el capítulo, pero sobre todo, para no perder el hilo conductor del mismo.

Antes de realizar el estudio con NFS versión 3, se realizó un exhaustivo análisis con NFS versión 2. En él ya aparecían los problemas presentados, pero tuvimos que desecharlos porque la versión 3 se presentaba como una mejora a la versión 2, y no sería lógico incluir nuevas propuestas sobre una versión que ya ha evolucionado a otra mejor. En las pruebas realizadas con la versión 2 utilizamos distintas secuencias de vídeo (*Flower&Garden*, *Hook* y *Composed*), pero decidimos utilizar sólo *Composed* para las pruebas siguientes porque era la única formada por trozos de distintas secuencias, y por tanto, era representativa de distinta clases de movimiento, con lo que era la candidata ideal a utilizar.

También nos planteamos trabajar sobre la versión 4, incluso pedimos el código fuente a la Universidad de Michigan, que tras rellenar unos cuantos formularios estuvieron de acuerdo en pasárnoslo. Pero, estaba *demasiado en desarrollo*, el fichero exports, que cambia totalmente con la versión anterior, no tenía aún un formato definitivo. Así que decidimos utilizar la versión 3. Además de todo lo que se ha dicho ya de ella, quedémonos con la idea de que devuelve los atributos de fichero en los resultados de todos los procedimientos para prevenir la necesidad de que el cliente haga un GETATTR (operación para pedir los atributos de un fichero) para actualizar los atributos cacheados.

Cuando cambiamos a la versión 3 de NFS también incorporamos el API gm a Myrinet para mejorar el rendimiento aún más, actualizamos la versión de Linux y cambiamos el kernel a la serie 2.4 que ya incorpora la versión de NFS V.3 como módulo totalmente estable.

Uno de los cambios más importantes en la versión 3 son las operaciones asíncronas y el almacenamiento estable. Las operaciones de modificación de datos deben ser síncronas, esto está muy relacionado con la naturaleza sin estado del servidor. Pero puede tener un importante efecto en el rendimiento, que puede mejorarse con operaciones asíncronas (aunque haya riesgo de pérdida de datos). El tema de las escrituras asíncronas se ha debatido durante muchos años por los ingenieros de NFS. Para escrituras asíncronas seguras hay que realizar peticiones WRITE seguidas de peticiones COMMIT.

Pero no sólo hemos hecho pruebas con NFS, también preparamos nuestro cluster como un entorno Samba, creando los *shares* correspondientes en el servidor para compartir la secuencia de vídeo y probando un montón de opciones del smb.conf, pero los tiempos de lectura eran mucho más grandes y abandonamos el intento. Por tanto, de Samba me quedan dos carpetas bien gordas de información y resultados, y la satisfacción de haber ayudado a todos los del departamento que querían configurar Samba. Y es que el objetivo de Samba es otro, es poder compartir todos los recursos de la red, ya sean hardware o software, y dentro de estos, compartir información en máquinas con distinto sistema operativo (linux y Windows, por ejemplo). Mientras que NFS se diseñó con la suposición de **baja latencia**. Hay, precisamente, algunas decisiones de diseño de NFS que limitan el conjunto de aplicaciones para las que NFS es apropiado [191]:

- El diseño concibe los clientes y servidores conectado en una red rápida. El protocolo no trabaja bien sobre enlaces lentos.

- El modelo de *caching* asume que **no** se van a compartir muchos ficheros. El rendimiento se ve afectado cuando los ficheros son compartidos duramente
- El protocolo sin estado requiere alguna pérdida de la semántica tradicional de UNIX

En [192] se puede ver una comparativa entre NFS, Samba y Coda con diferentes tests de lectura y escritura, y NFS es la solución más rápida. Añadir también, para tener una idea más cercana del entorno de trabajo en el que nos hemos movido, que el sistema de ficheros NFS es un ejemplo de éxito en la industria y es el estándar de facto a la hora de compartir particiones en UNIX, pero es de los pocos sistemas de archivos que **no** tienen fichero explicativo de implementación en /usr/src/linuxDocumentation. En el transcurso de la implementación he podido contar con el apoyo técnico de los expertos NFS de la lista nfs@lists.sourceforge.net que me han aclarado algunas dudas muy puntuales.

Vamos ahora a describir el comportamiento del cliente y del servidor NFS, y las operaciones básicas que vamos a modificar, para pasar a exponer la propuesta de mejora.

5.2. POSIBLES MEJORAS EN EL CLIENTE Y EN EL SERVIDOR NFS

A continuación vamos a describir el funcionamiento del servidor y el cliente en NFS. Ya que ambos fueron candidatos posibles de mejora. Describiremos ambos porque en un principio nos planteamos mejorar el servidor, modificando la forma en que se procesan las peticiones de lectura. Pero, un conocimiento más detallado del mismo, así como la lectura de algunos trabajos realizados en prefetching, nos llevaron a pensar que sería más adecuado realizar las modificaciones en el cliente, como después hemos podido comprobar. Partimos del patón de lectura 2, es decir, los clientes van a pedir leer ficheros completos de datos. Partimos de ésta por su simplicidad, luego veremos cómo se realizará también el prefetching para la lectura de trozos de datos, y cómo va a ser ésta una de las novedades principales de la técnica de prefetching implementada porque no se van a adelantar bloques de datos sino trozos de cualquier tamaño.

La idea inicial fue dotar de algún tipo de estado al servidor NFS, ya que esta idea ya se estaba gestando en la versión 4 pero sólo para ciertas operaciones. Nosotros nos planteamos en un primer momento anticipar datos a través de ficheros gracias a un modelo simple de prefetching basado en reglas, por lo tanto queríamos aportar algo de estado a la operación de lectura. El servidor podría crear unas tablas de conocimiento con información útil de cada fichero pedido (cliente y offset, por ejemplo) y aprender el comportamiento a seguir mediante sencillas comparaciones. Se aplicarían heurísticas como la siguiente: si un cliente empieza a leer un fichero en un punto determinado, entonces todos los ficheros los va a leer a partir de ese punto. Estas heurísticas se aplicarían en función de la información almacenada en las tablas

5.2.1. Descripción de las mejoras a realizar en el servidor

Lo primero que hay que hacer para conocer el funcionamiento del servidor es meterse de cabeza y sin pensarlo en /usr/src/linux/fs/nfsd, donde está todo el código fuente correspondiente al servidor y a los procedimientos definidos para él.

De forma general, la forma de trabajar del servidor se puede resumir en el siguiente diagrama de actividad (figura 5.1). En él vemos claramente la función del servidor, recibir y procesar peticiones. En el bloque Recibe siguiente petición, el servidor principalmente identifica el tipo de operación, prepara las estructuras, busca el socket a través del cual realizará esa operación e identifica al cliente.

En el bloque Procesa petición realizará las llamadas correspondientes a los procedimientos NFS necesarios. Dos de los procedimientos involucrados en la lectura son:

- *NFSPROC·_LOOKUP*: que busca el nombre del fichero, un directorio para un nombre específico y devuelve los atributos y el manejador de archivos para el correspondiente objeto del sistema de archivos. Como argumentos de entrada necesita el objeto a buscar y, si todo va bien, se obtiene el manejador de fichero para el objeto correspondiente, los atributos para el objeto correspondiente y los atributos para el directorio.

- *NFSPROC3_READ*: lee de un fichero. Los argumentos que se le pasan son el manejador del fichero del que se va a leer, el offset o posición del fichero en la que se va a empezar a leer, y la cantidad de bytes de datos que se quieren leer. Si todo va bien, se obtienen los atributos del fichero leído, el número de datos devueltos por la operación de lectura, una variable que indica si la lectura del fichero acabó con el final de fichero y los datos en sí leídos.

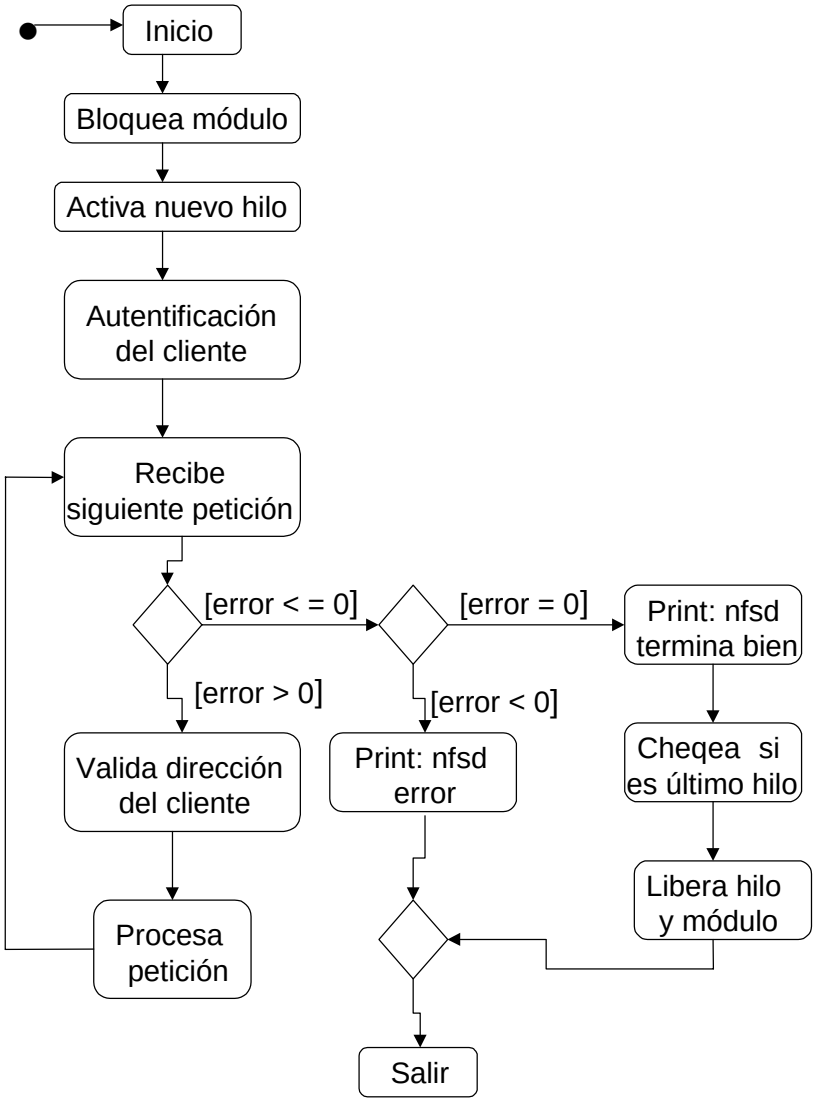


Figura 5.1. Diagrama de actividad del servidor NFS

Los procedimientos LOOKUP y READ eran los que nos planteábamos modificar. Veamos que se hace en cada uno de ellos (figura 5.2):

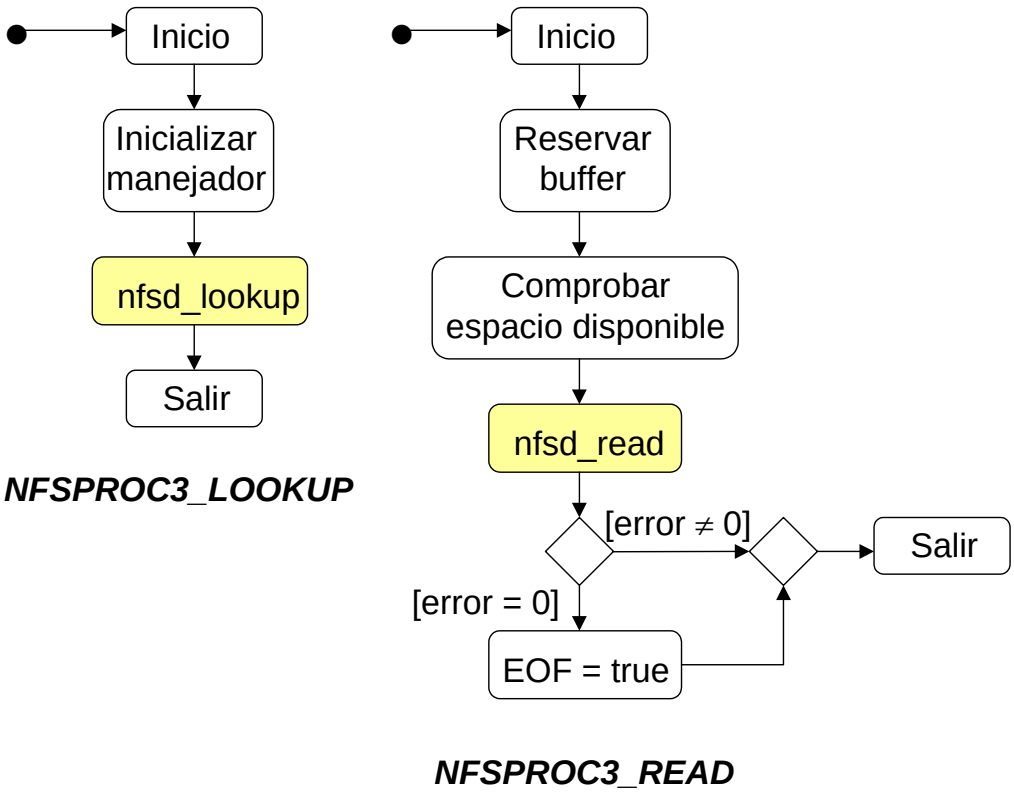


Figura 5.2. Diagrama de actividad de los procedimientos LOOKUP y READ

En NFSPROC3_LOOKUP tan solo se inicializan las estructuras donde se va a almacenar el manejador y se llama a la operación lookup propiamente dicha, donde se realizan detalladamente todos los pasos necesarios para conseguir el objetivo final.

En NFSPROC3_READ se reserva el buffer, se comprueba que hay espacio disponible, se llama a la operación read (nfsd_read) y después se pregunta si se ha leído el final de fichero. Esto se calcula simplemente comparando si el offset especificado más la cantidad de datos a leer igual al tamaño del fichero. Si es así, se pone una variable a TRUE y se acaba la operación, si no es así se sale igualmente.

Veamos ahora de forma resumida cómo se realiza la comunicación entre el cliente y el servidor cuando el cliente quiere leer tres ficheros cualesquiera f1, f2 y f3 (figura 5.3).

Para leer un fichero antes hay que crear un manejador para ese fichero, y después enviar los READ's necesarios hasta completar la cantidad de bytes que se quieran leer.

La novedad que nos planteamos incorporar consistía en que, después de la lectura de un fichero, si se ha llegado al final del mismo, empezar a leer el siguiente fichero sin que exista ninguna petición por parte del cliente. Nos planteamos deducir el fichero siguiente a partir de la información que se debe haber almacenado en el servidor. Esto tiene las siguientes implicaciones directas:

1. Conocer más a fondo cómo se realizan las operaciones de lectura y de lookup, pero modificando los procedimientos NFS afectados (NFSPROC3_#), y no las llamadas internas a las operaciones a más bajo nivel.

2. Especificar cómo se va a leer el nuevo fichero.

El punto 1 es de investigación pura y dura, pero para el punto 2 hay que decidir cómo se va a llevar a cabo la operación. La primera idea fue construir un *grafo de accesos* con los nodos pedidos por los clientes para leer, y utilizar ese grafo, una vez construido, para buscar el siguiente fichero a leer, localizando el nodo siguiente al actual. La idea parecía bastante simple en teoría, pero los problemas iban a aparecer en el momento de ponerse a especificar la operación.

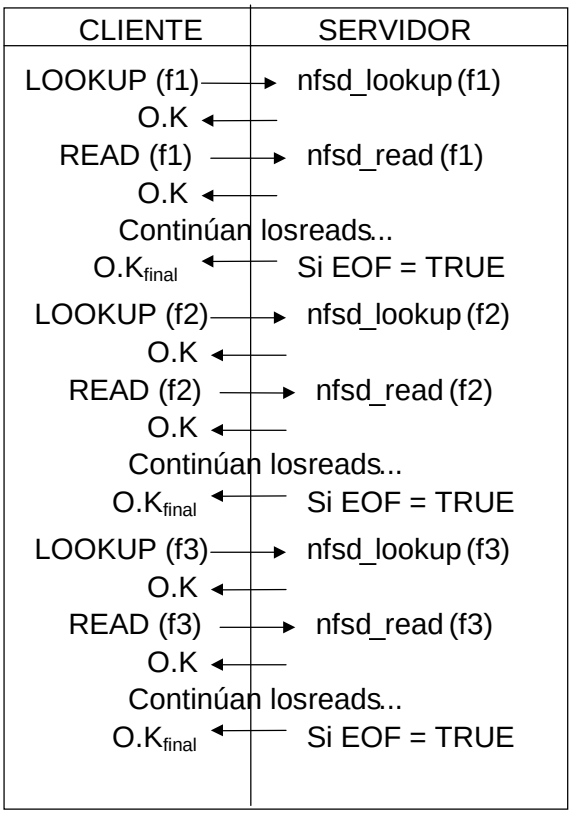


Figura 5.3. Comunicación cliente-servidor

Había distintas formas de plantear la operación:

Opción A

- Utilizar la misma petición del cliente para leer el siguiente fichero
- Insertar un nuevo nodo en el grafo cada vez que se realiza un LOOKUP de ese fichero
- Si al leer un fichero, se llega al final del mismo, entonces buscar en el grafo el siguiente fichero y llamar a las correspondiente operaciones de LOOKUP y READ.

Con esta posibilidad, si pensamos en la lectura que se realiza de las secuencias de vídeo, donde siempre se leen los ficheros de la misma forma consecutiva, el grafo se crearía sólo una vez y ya se utilizaría después siempre para consulta.

Otra posibilidad es crear un grafo siempre con cada ejecución de la aplicación

Opción B

- Utilizar la misma petición del cliente para leer el siguiente fichero
- En la operación LOOKUP de un fichero, realizar un listado de los ficheros presentes en el directorio y con ellos realizar un grafo, insertando los nodos de forma ordenada. El primer LOOKUP implicaría crear el grafo con todas las aristas a 0, después con los LOOKUPS sucesivos se irían incrementado los valores de las aristas correspondientes (si representamos en las aristas el número de acceso a un nodo destino desde un fichero en un nodo origen)
- Cuando se lee un fichero, el grafo ya está creado y puedo ver cual es el siguiente nodo en el grafo aunque nunca se haya accedido a él (el valor de la arista que une el actual con el siguiente está a 0).

Esta opción permite crear un grafo cada vez que se ejecute el programa y además se tiene el grafo completo desde el principio. El inconveniente principal es que es muy específico y muy orientado solo a este tipo de aplicación, ya que en una secuencia de vídeo lo normal es que se lean las imágenes que están en ficheros consecutivos (por orden alfabético). Por lo tanto, esto iría bien para la lectura de secuencias de vídeo, pero no para otro tipo de lecturas de ficheros arbitrarios, por lo tanto esto no lo queríamos plantear así. No pretendíamos hacer algo tan particular, sino, ¿para qué meterse en el núcleo del sistema operativo?. Lo mejor es proponer una solución general, o por lo menos, suficientemente general.

Por tanto la opción B se descartaría. Lo que nos planteamos entonces fue refinar la opción A. Lo ideal y lo normal es que en una primera ejecución de la aplicación se construya el grafo y en las siguientes se utilice ese grafo. Así se conseguiría hacer el *prefetching* a través de ficheros deseado.

Lo primero era plantear las modificaciones en las operaciones de LOOKUP y READ y definir cual sería ahora el desarrollo de las actividades por parte del servidor.

En el procedimiento LOOKUP lo que hay que incorporar es una nueva llamada a una función que inserte el fichero en un nodo del grafo, si no está. Esta inserción se hará solo en el caso de que se esté realizando un LOOKUP que proviene de una petición expresa del cliente.

En el procedimiento READ hay que activar el mecanismo de prefetching una vez que se ha llegado al final del fichero actual. Lo que implica buscar en el grafo construido previamente, el siguiente fichero, y si existe, pedir para él el manejador correspondiente (llamar al procedimiento LOOKUP) y realizar la petición de lectura (nfsd_read). Veamos gráficamente estos cambios (figura 5.4)

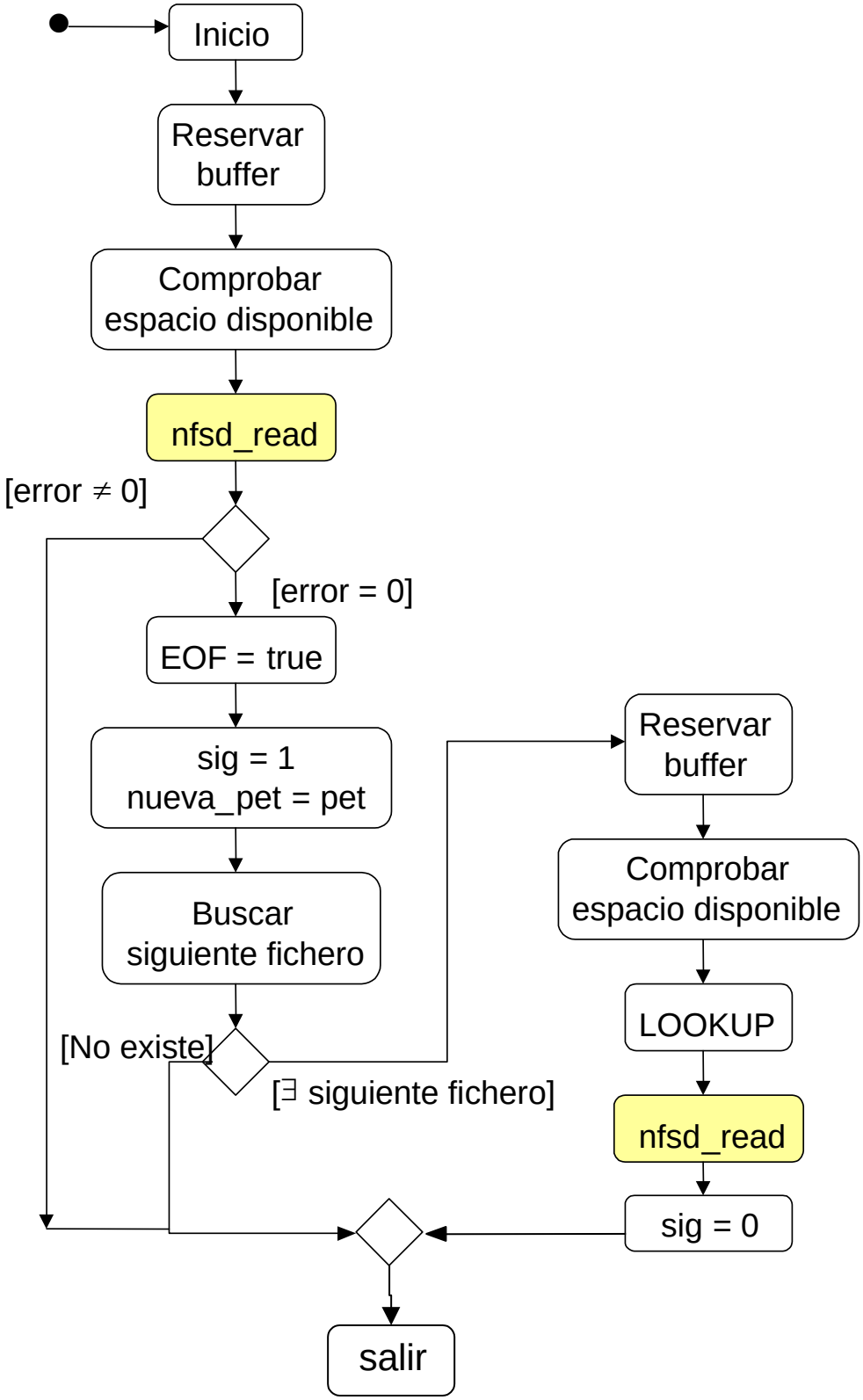


Figura 5.4. Modificaciones en el procedimiento NFS3PROC_READ

Con estos cambios, la forma de trabajar del servidor sería la siguiente:

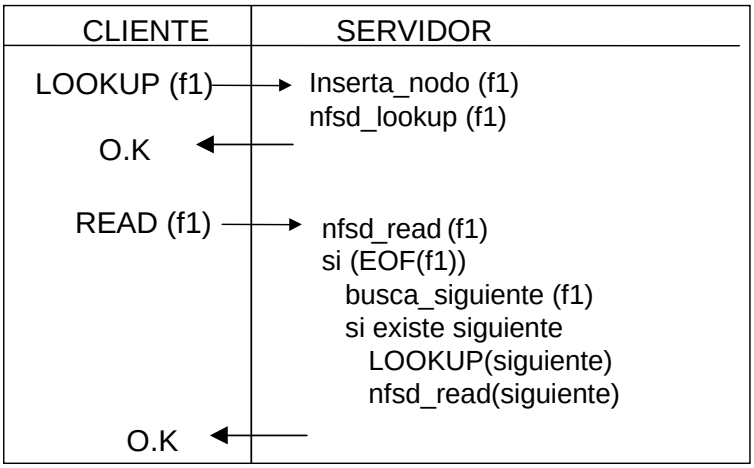


Figura 5.5. Funcionamiento del servidor con *prefetching* a través de ficheros

Con este planteamiento nos pusimos manos a la obra en el proceso de diseñar la operación de Inserta_nodo y Busca_siguiente.

La idea que se quería implementar era la siguiente:

- Tener un grafo por cada cliente
- Cada acceso a un fichero (lookup) inserta un nodo en el grafo (sino existe ya). Cada nodo del grafo representa un fichero
- Cada primer acceso a un fichero inserta una arista que va del último fichero accedido al nuevo nodo.

Ejemplo:

cliente 1 lee f1, cliente 2 lee f1, cliente 3 lee f2, cliente 1 lee f2, cliente 2 lee f4, cliente 3 lee f3, cliente 1 lee f3, cliente 3 lee f2, cliente 3 lee f3
Grafo generado:

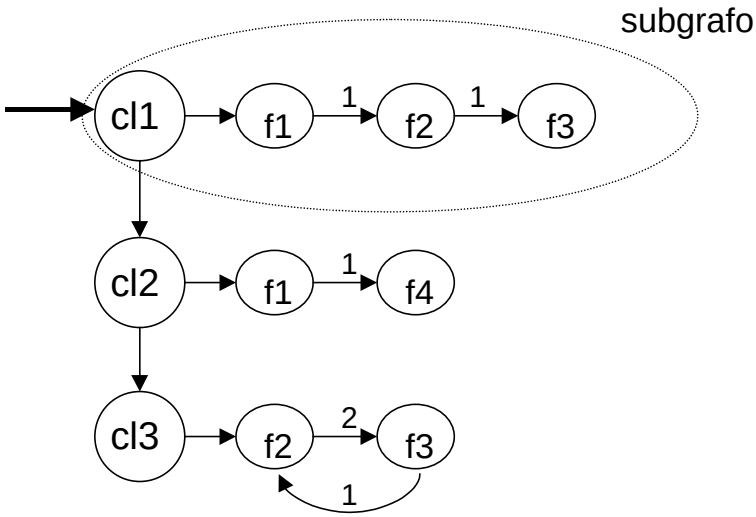


Figura 5.6. Ejemplo de generación del grafo de accesos

La idea parecía bastante clara, pero conforme más nos adentramos en el código del servidor más problemas encontrábamos. El inconveniente principal en el servidor era el siguiente:

Imposibilidad de generar una nueva petición a LOOKUP y READ por sí mismo. La petición que llega al servidor proveniente del cliente está gestionada por el RPC (Remote Procedure Call) y además codificada por el XDR (Extern Data Representation), por lo que nos era prácticamente imposible poder manejar, interpretar y generar esta información.

Por otro lado, el proceso no se podría estructurar como en la figura 5.5, ya que tal como se plantea ahí, el servidor va a mandar la confirmación de lectura (oK) después de realizar todo el proceso de inserción o búsqueda en el grafo. Esto no se pude plantear así porque lo estamos sobrecargando más. Hay que utilizar otros mecanismos para hacerlo, como por ejemplo, procesos hilo.

Además, en [192] aparece justificada la inclusión de técnicas de prefetching en el cliente, en arquitecturas cliente-servidor típicas. Así que, ante el desánimo producido por la cantidad de tiempo invertido en el servidor sin ningún resultado, decidimos irnos a conocer el funcionamiento del cliente, como única alternativa donde poder aplicar nuestra técnica. Precisamente será el trabajo presentado en [192] uno de los más tenidos en cuenta. Presenta el prefetching desde el lado del cliente justificado razonadamente (así se guardan los datos en la cache del cliente), y además plantea una técnica basada en la elaboración de un árbol de accesos, la primera vez que se accede a los ficheros, y después, las siguientes, se consultará ese árbol parea ver el patrón a seguir y pedir los datos al servidor. Este será también nuestro planteamiento, lo que ocurre es que en [192] no se llega a implementar en un sistema real y está pensado para ficheros UNIX en general.

Por último, algunos comentarios más acerca del servidor.

1. Al ser sin estado, el servidor no puede detectar que el fichero está abierto y debe hacer chequeo de permisos en cada llamada read y write.
2. Aunque es posible implementar un servidor de un único hilo, en la práctica, muchas implementaciones son altamente paralelas. Si un servidor está para proporcionar buen servicio a un gran número de clientes, debe ser capaz de servir peticiones NFS concurrentemente. El número de hilos se puede modificar en el kernel (aparece como un #define) y lo normal es tener 8 ó 16 en el servidor (nosotros hemos puesto 16).

5.2.2. Descripción de las mejoras en el cliente

El código NFS del cliente lo podemos encontrar en /usr/src/linux/fs/nfs, aquí lo que encontraremos es la descripción de los procedimientos, ya que el funcionamiento general no es un bucle continuo como el servidor, ahora es algo más abstracto,

interviene el VFS (Virtual File System). Sólo necesitamos conocer el código de los procedimientos que se llaman, sin meternos en cómo se harán las llamadas. Después de investigar en profundidad todo el código, nos decantamos por la modificación del fichero file.c, donde están las funciones de manejo de archivos regulares.

Pero, para hacer la exposición más gráfica e inteligible, volvamos a recordar una figura utilizada en la descripción de NFS (figura 5.7) en el capítulo 3. En ella podemos ver la comunicación NFS de forma general, sin profundizar demasiado. Desde la aplicación, cuando se necesite leer información de un archivo, se harán llamadas al sistema, que serán analizadas por el VFS y que es el que hace posible la convivencia de tantos sistemas de archivos en el kernel. Este VFS decide si la información está almacenada en un fichero local al cliente o, si por el contrario se encuentra almacenada en un lugar remoto. En este caso se producirá la correspondiente llamada al servidor NFS.

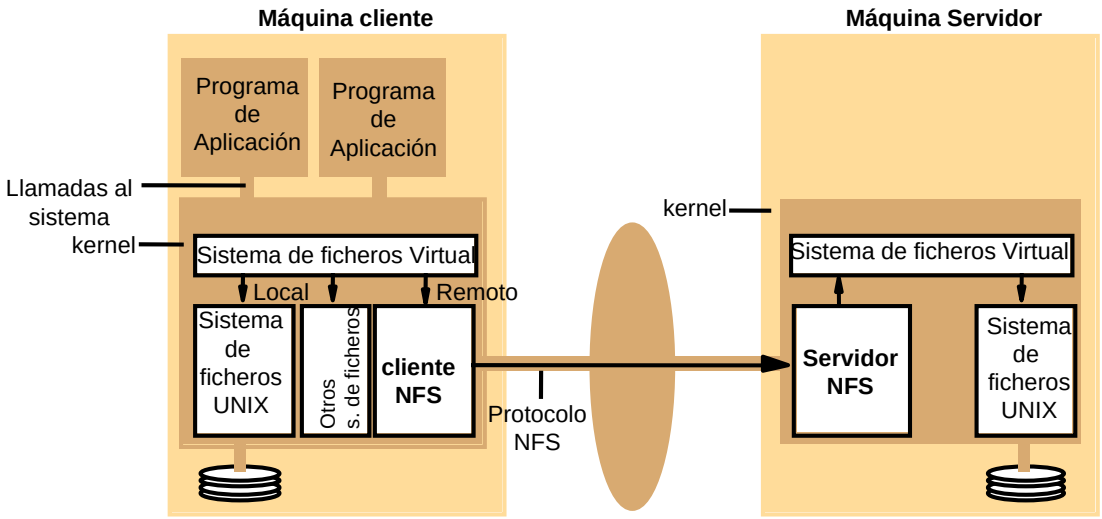


Figura 5.7. Arquitectura cliente-servidor NFS

A continuación veamos cuáles son los procedimientos que se desencadenan en el cliente ante una petición de lectura por parte de una aplicación (ver figura 5.8).

Ya hemos comentado que cuando el VFS recibe una petición de lectura de una aplicación, lo primero que decide es si se trata de una petición de un fichero situado en el disco local o si se trata de una petición de un fichero situado en una máquina remota. En el caso de NFS la información hay que ir a pedirla a una máquina remota y esto va a desencadenar las siguientes llamadas:

1. La llamada a la función de lectura NFS: nfs_file_read. El VFS llamará a esta función repetidas veces, hasta completar la cantidad de información que está solicitando la aplicación. En cada llamada se especifica el fichero, la

cantidad de datos a leer y la posición a partir de la cual se va a empezar a leer.

2. Dentro de esta función se realiza la llamada a la función genérica de lectura: do_generic_file_read, que va a ser la responsable de realizar todo el proceso de lectura. Esta función es la que evalúa la cache del cliente para evaluar si los datos pedidos ya están allí y evitar una petición al servidor. Si no encuentra los datos, llamará a la función que se encarga de pedir los datos al servidor.
3. Lo último que se hace en nfs_file_read es comprobar si se ha llegado a fin de fichero. Nosotros utilizaremos esta marca para invocar nuestra función de prefetching

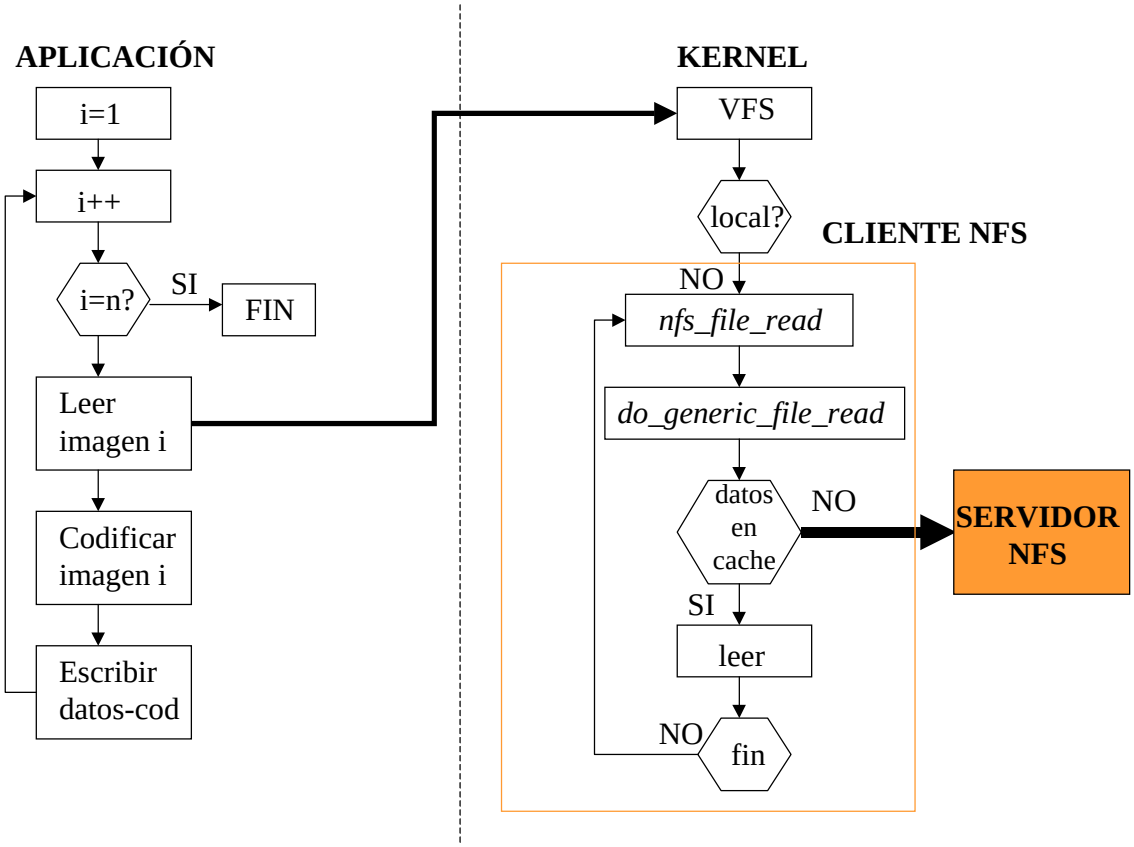


Figura 5.8. Funcionamiento normal del kernel

Pues bien, con este planteamiento, nuestra idea es crear un proceso hilo, justo después de detectar que se ha llegado a fin de fichero. Este proceso hilo es el que va a crear el grafo en los primeros accesos a los ficheros y el que, después, buscará en el mismo, el fichero siguiente al que se ha leído actualmente. Este fichero siguiente se leerá como de la misma forma que otro fichero convencional, por lo tanto, se llamará a la función genérica de lectura para completar el proceso.

La idea del prefetching a través de ficheros en el kernel, la podemos ver en la figura 5.9, en la que ya se supone que está el grafo creado.

De esta figura nos queda detallar la toma de decisión sobre la existencia de los datos en la cache. Ya que será un proceso muy importante, una vez que esté el prefetching en marcha, y a la hora de leer el fichero actual. Antes de leer cualquier fichero Fi, se supone que si todo ha ido bien, el hilo ya se lo debe haber traído a la cache del cliente, pues cuando se terminó de leer el fichero F_{i-1} se invocó al proceso hilo para leer el fichero Fi. Por lo tanto vamos a detallar también qué es exactamente lo que se produce en esa toma de decisión.

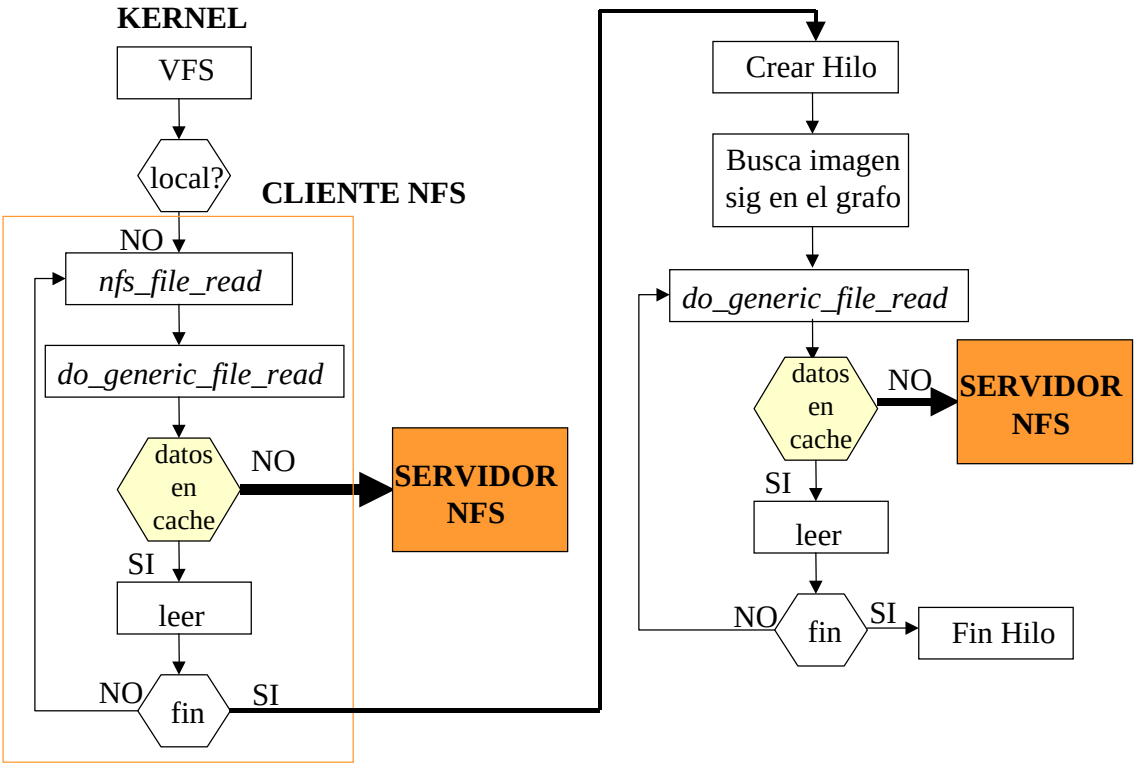


Figura 5.9. Prefetching en el kernel

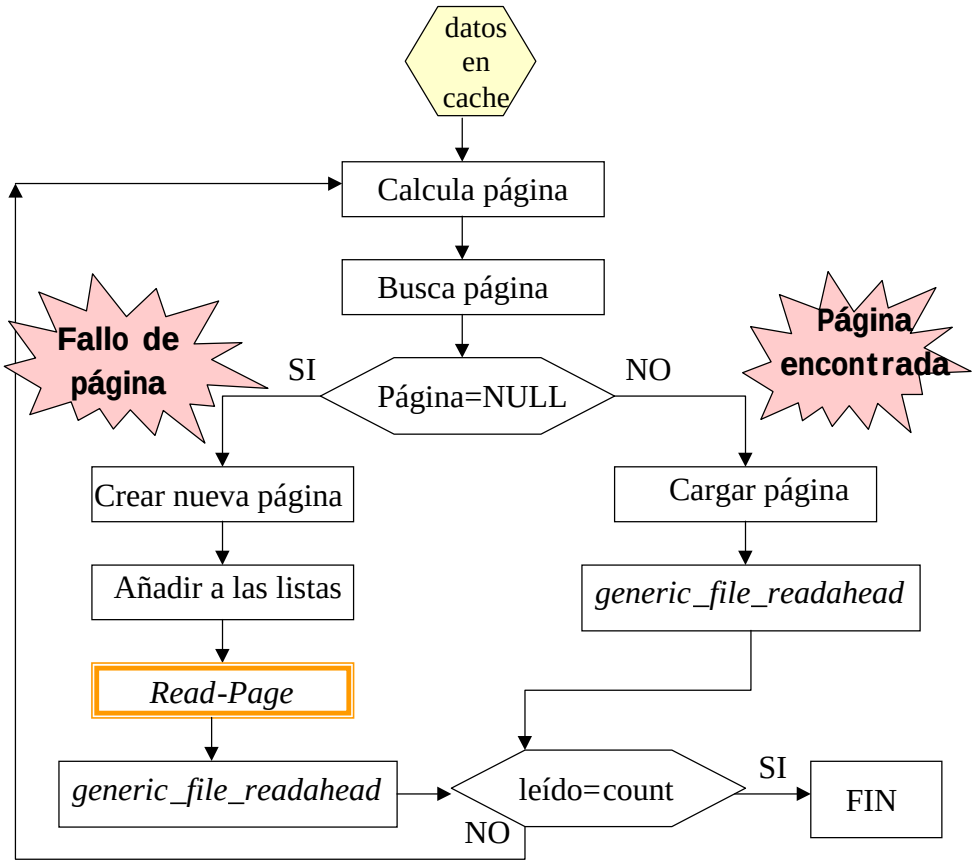


Figura 5.10. Decisión sobre la existencia de datos en la cache

Cuando se pide leer una cierta cantidad de información, esto se traduce en leer un número determinado de páginas de memoria. Lo primero que se hace es calcular la página correspondiente y buscar esa página. Si se encuentra (la página no es nula), entonces se carga y se llama a la función generic_file_readahead, que leerá las páginas siguientes. Pero, si no se encuentra, se debe crear un nueva página, y se llama a la función read_page, que es la que realizará las peticiones al servidor. También, después de esta petición se intenta leer la página siguiente llamando a la función generic_file_readahead.

Cuando las páginas leídas coincidan con la cantidad que se especifica, se termina la función do_generic_file_read, que es desde donde se realiza esta toma de decisión.

Por último, presentamos un poco más en detalle el funcionamiento de esta función de lectura adelantada en la figura 5.11.

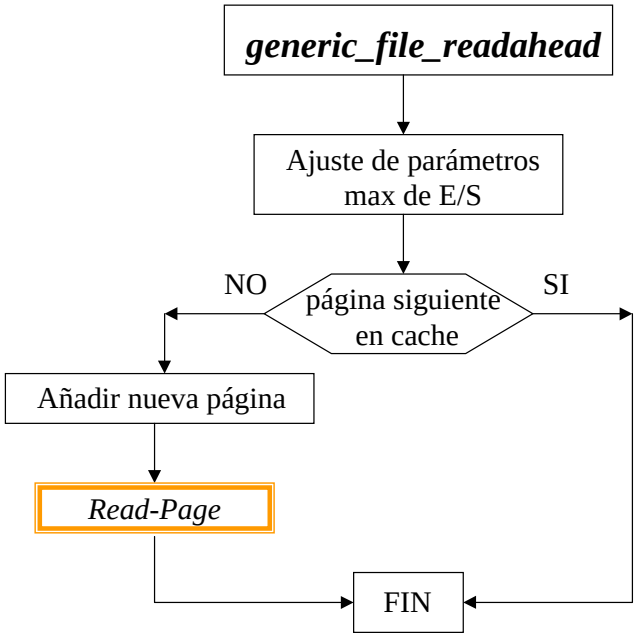


Figura 5.11. Funcionamiento de la función generic_file_readahead

Hasta el momento tan sólo hemos presentado en qué podría consistir la mejora en el cliente (figura 5.9). Este sería la nueva forma de operar el núcleo. Pero, ahora falta describir verdaderamente cómo se llega a este funcionamiento. Como se crea el grafo de accesos y cómo se utiliza el mismo para buscar el siguiente fichero. Por tanto vamos a describir el algoritmo de prefetching en el que nos basamos y como lo implementamos. Antes, creemos oportuno exponer algunos trabajos realizado en prefetching y que hemos tenido en cuenta para realizar nuestro trabajo.

5.3. TRABAJOS DESTACADOS EN PREFETCHING

Se han realizado variados e interesantes trabajos sobre *prefetching* en los últimos 10 años. Desde la publicación en 1991 del trabajo *"Optimal Prefetching via Data Compression"* de Vitter y Krishman, tantas veces referenciado en trabajos posteriores, hasta las más actuales publicaciones orientas a multimedia, existen toda una serie de trabajos en los que se diseñan, implementan, simulan y testean distintos algoritmos de *prefetching* útiles en distintas situaciones y con distinto propósito, ya sea con sistemas uni o multiprocesador u orientado a clusters, para acceso secuencial o paralelo, para predicción de bloque de datos o de ficheros completos, pero eso sí, todos con el mismo propósito: mejorar el rendimiento del sistema anticipándose a las peticiones de los usuarios. El *prefetching* se podría definir como una técnica general que mejora el rendimiento de E/S leyendo datos y volcándolos a la cache antes de que se pidan, solapando E/S con computación.

Voy a citar a continuación los trabajos más interesantes y las conclusiones sacadas de estas aportaciones, para después entender con claridad la importancia de nuestra aportación en este tema.

He querido destacar cinco grupos de investigación principales en este tema y algunos otros trabajos realizados en distintas universidades. Los cinco grupos de investigación son:

- El grupo de Jeff Vitter de la Universidad de Duke, http://www.cs.duke.edu/~jsv/Papers/catalog/node71.html, al que se le atribuye uno de los primeros trabajos importantes en *caching* y *prefetching*, y además más referenciado posteriormente [154] y otros posteriores más especializados [155]. De este grupo también es importante destacar el proyecto TPIE (*Transparent Parallel I/O Environment*) diseñado para servir de puente entre la teoría y la práctica de los sistemas de E/S paralelos e implementado como un conjunto de clases y funciones C++ [156]. También destacar el grupo de trabajo dedicado a la E/S en computación a gran escala, al desarrollo de algoritmos y entornos eficientes de E/S [157].

- Los trabajos del profesor G. A. Gibson, http://www.cs.cmu.edu/People/garth, sobre el famoso *Informed Prefetching*, método de optimización de los patrones de acceso colectivo capaz de explotar cualquier cantidad de memoria disponible y solapar E/S con computación. Surgido de la realidad de que los sistemas de ficheros paralelos no han dado la respuesta esperada y con el deseo de que la propia aplicación proporcione información acerca de los patrones de acceso a los ficheros [158, 159]. De este grupo también son muy destacables los trabajos realizados en la definición de una nueva arquitectura de almacenamiento (*NASD, Network-Attached Secure Disk)* [160, 161] y en la contínua investigación sobre el almacenamiento compartido [162] y la ejecución especulativa [163].

- Los trabajos del grupo del profesor Darrell D. Long, http://www.cse.ucsc.edu/~darrell, orientados a mejorar los servicios de los protocolos de transmisión de vídeo [164], sistemas de ficheros escalables [165] y mejora de rendimiento a través de la cache predictiva [166, 167]. Ha colaborado también en interesantes trabajos de *prefetching* como [190] donde ya se plantean un diseño de modificación del VFS, pero en el que se *simula* el comportamiento de sistemas de computadoras típicas, llegando a la conclusión de que el *prefetching* predictivo tiene un enorme potencial para reducir las latencias de E/S.

- El *Multimedia Networking Group* con el profesor K. W. Ross del departamento de Multimedia Communications del Instituto Eurécom de Francia, http://www.eurecom.fr/~ross/MMNetLab.htm, donde se han publicado interesantes trabajos sobre web *caching* y protocolos de *prefetching* para vídeo [168, 169, 170, 171].

- Los trabajos realizados por el grupo de Toni Cortés en la Universidad Politécnica de Cataluña, http://www.ac.upc.es/homes/toni, sobre *prefetching* y caches corporativas [172, 173]

Además de todos estos trabajos fundamentales para el conocimiento y puesta al día de las principales técnicas y algoritmos de *prefetching*, existen otros trabajos muy interesantes:

- Incorporación de técnicas de *prefetching* para demostrar la viabilidad y mejora de determinadas arquitecturas [174, 175, 176]
- Entornos de simulación para evaluación del *prefetching* [177]
- Definición de nuevos sistemas de gestión de almacenamiento de datos [178]
- Determinadas técnicas de acceso a la cache [179, 180]
- Desarrollo e implementación de librerías que acercan el *prefetching* a todos los sistemas [181]
- Estudio del contenido semántico de los documentos como base para predecir futuros accesos [182]
- Reducción de latencias en sistemas web [183, 184, 185, 186]
- Sistemas orientados a multimedia completamente [187]
- Protocolos de *Prefetching* para entornos inalámbricos [188]
- Utilización de la información de la red y de la aplicación, para *prefetching* y *caching* de los datos necesarios, en paralelo con la ejecución de la aplicación, para reducir el impacto de la red. Este trabajo se plantea sobre redes de área amplia y aplicaciones multimedia y metacomputación [189]

Después del *caching*, el *prefetching* es la siguiente etapa lógica para incrementar el rendimiento de los sistemas de E/S. Las líneas de trabajo en *prefetching* han ido por las siguientes vías:

- *Prefetching* orientado a incrementar el rendimiento de los sistemas de ficheros mediante:
  - Predicción de bloques de datos
  - Predicción del siguiente fichero (de bloques del siguiente fichero)
- *Prefetching* orientado a mejorar los sistemas web

Entre los trabajos de *prefetching* orientados a incrementar el rendimiento de los sistemas de ficheros mediante la predicción del siguiente bloque del fichero del que se están leyendo datos, tenemos como destacados los trabajos de [154] donde se proponen nuevos algoritmos para *prefetching* basado en técnicas de compresión de datos que son óptimas en teoría y funcionan bastante bien en la práctica, y con el propósito de reducir las latencias en sistemas de almacenamiento secundario. Están basado en modelos de Markov. Estos algoritmos han sido el punto de partida de otros algoritmos de *prefetching* más recientes como los presentados en [172] donde se proponen algoritmos de *prefetching* para incrementar el rendimiento de un sistema donde existen muchas aplicaciones corriendo concurrentemente. Se presenta una propuesta de *prefetching* agresivo que no solo se basa en la predicción de bloques necesarios para una aplicación sino en que, si los bloques son correctamente predichos, el sistema observará una mejora considerable de rendimiento. Además se muestran interesantes resultados sobre dos sistemas de ficheros paralelos/distribuidos: PAFS y xFS.

En [157] se hace una clasificación de las aproximaciones básicas que soporta el desarrollo de código eficiente de E/S:

- Sistemas orrientados a arrays de discos (PASSION y ViC)
- Sistemas orientados a accesos (sistemas de ficheros UNIX y proyecto Panda)
- Sistemas orientados a marcos de trabajo (TPIE)

Y además, se discute el potencial y las ventajas de TPIE para permitir computación de E/S eficiente.

Existen otros protocolos de *prefetching* para vídeo VBR [171] que se basan en la observación de períodos de tiempo en los que el ancho de banda de la red está infrautilizado, y durante esos períodos el servidor puede anticipar frames a los clientes. Se han realizado *simulaciones* basadas en MPEG codificado y se comparan los protocolos de *prefetching* (el centralizado JSQ, *Join-the-shortest-Queue*, y otros descentralizados) con otros protocolos de transporte.

En [158] se presenta el *Informed Prefetching*. El problema aparece claramente: los cuellos de botella en la E/S. La motivación principal es que los sistemas de ficheros distribuidos, las redes de área amplia y los servidores web han llevado a los usuarios muy lejos de sus datos y, por tanto, han añadido una latencia de acceso a esos datos. Entonces, ¿Cómo ayudar a las aplicaciones a sacar ventaja del ancho de banda disponible para minimizar la latencia?. Su propuesta es que las propias aplicaciones construyan “pistas” que describan accesos a los datos futuros, siempre con el objetivo de reducir la latencia de lectura.

En [159] se utiliza el *Informed Prefetching* en E/S colectiva, es decir, en un patrón de acceso global donde los procesadores acceden simultáneamente a porciones de un fichero paralelo. Se han revisado las distintas técnicas existentes para proporcionar información sobre el patrón de acceso a los sistemas de ficheros:

a) A través de un API que permita a los discos reordenar las peticiones
b) La E/S en dos fases: definición de librerías a nivel del usuario para soportar la E/S colectiva
c) E/S dirigida a disco: utilizar procesadores de E/S que ordenen las peticiones de bloques físicos y transfieran los bloques a los procesadores que los pidan

Y se propone una técnica más general en la que la aplicación construye pistas para descubrir accesos futuros y que un predictor usará para tratar las peticiones antes del flujo de demandas de los usuarios. Con información especificada por la aplicación y soporte a nivel del sistema se conseguirá una planificación óptima de peticiones de E/S colectiva. Se muestran interesantes resultados, obtenidos *por simulación*, con distintos patrones de acceso a los bloques físicos de datos (secuencial entrelazado, bloque 1-D, bloque 3-D)

Otros trabajos más recientes como [187] proponen mecanismos automáticos de *prefetching* específicos para una aplicación, orientados principalmente a mejorar el rendimiento de programas multimedia con patrones de acceso complicados (no secuenciales). Se basan en la idea de transformar el *programa de aplicación* en uno nuevo con dos hilos: el hilo de computación (donde está el programa original que contiene los cálculos y la E/S en disco) y el hilo de *prefetching* (que contiene todas las instrucciones del programa original que están relacionadas con los accesos a disco). En tiempo de ejecución se planifica el hilo de *prefetching* para que vaya por delante del hilo de computación y así los bloques de disco se anticipan y se ponen en la cache del

sistema de ficheros antes de que se necesiten. A esta técnica se le llama AASFP (*Automatic Applications Specific File Prefetching*).

Con respecto a la predicción del siguiente fichero, existen dos trabajos muy interesantes y que además serán tenidos en cuenta a la hora de realizar las modificaciones en NFS. Uno de ellos es [181] orientado a aplicaciones multimedia. Plantea el uso de técnicas de *prefetching* adaptativo que reducen la latencia y el jitter. Las aplicaciones deben proporcionar un plano del acceso a ficheros al *prefetcher*, éste generará las peticiones de E/S al sistema operativo y gestionará la cache para aislar la aplicación de las variaciones en el rendimiento de los dispositivos. Por lo tanto, se describe una nueva arquitectura para *prefetching* dirigido a la aplicación que proporciona E/S independiente del dispositivo para aplicaciones multimedia. Se basan en los siguientes principios:

- La E/S es el cuello de botella para aplicaciones multimedia
- No sólo es necesario conocer los datos que se van a necesitar sino cúando se van a utilizar
- Las variaciones en la latencia reducen la calidad de las presentaciones multimedia
- Las técnicas de *prefetching* tradicionales no van bien para aplicaciones multimedia
- Se presenta una nueva propuesta y se discuten las mejoras sobre las técnicas actuales

Lo que se hace realmente es definir una librería software que:

- Mapea la información proporcionada por la aplicación en tiempo real al buffer de la cache, gestionando operaciones necesarias para ocultar latencias de E/S y jitter
- Simplifica la tarea de programar aplicaciones multimedia adaptativas
- Permite, simplemente, declarar sus requisitos y su comportamiento de forma anticipada.

El segundo trabajo que vamos a tener en cuenta es [166]. Los accesos a ficheros están fuertemente correlacionados con los accesos precedentes. Los sistemas de ficheros (y en particular NFS) hacen *prefetching* secuencial (*read-ahead*) del siguiente bloque de disco. Esto se hace porque hay un concepto del siguiente elemento de datos secuencial intrínseco en la abstracción del sistema de ficheros. Sin embargo, no hay un concepto intrínseco del siguiente fichero en la secuencia. Esto limita la habilidad del sistema de ficheros para anticipar datos a través de ficheros. En este trabajo se exponen tres modelos para predecir los accesos a ficheros (*last-succesor*, basado en grafos y FMOC *Finite Multiorder Context Modeling*) y se realizan distintos tests sobre ellos y se presenta un modelo del FMOC mejorado llamado PCM (*Partitioned Context Model*) presentándose también los resultados de sus predicciones. Con este nuevo modelo se obtiene una exactitud mejorada debido a su habilidad para adaptarse más rápidamente a los cambios en los patrones de acceso.

Otra de las líneas de investigación interesantes la ocupan las técnicas de *prefetching* orientadas a la reducción de latencias en sistemas web, como podemos ver en [167, 182, 183, 184, 185, 186]. En [167] se clasifican los métodos de *caching* y *prefetching* y se especifican sus limitaciones. Se dividen las latencias web en dos

componentes principales: la latencia externa (causada por la red y el servidor web externos a la organización) y la latencia interna (red y computadores de la organización). En este trabajo se muestran el *caching* y el *prefetching* como técnicas útiles, pero con limitaciones en su habilidad para reducir latencias. El objetivo será determinar el límite superior de efectividad de estas latencias. Interesantes son también las diferencias entre la cache de los sistemas de ficheros y los sistemas web mostradas en [186] que explican por qué los métodos para *caching* en sistemas de ficheros no funcionan bien para tráfico web. En [182] se distingue entre *caching* individual y colectivo y además se propone el uso del contenido semántico de los documentos como la base para predecir los accesos a los ficheros. Estas semánticas se definen con respecto a un grupo de trabajo (un grupo de trabajo es un grupo de usuarios trabajando en la misma tarea o en tareas similares). Además se muestran ejemplos, como las tiendas en Internet, donde se está haciendo uso de algo similar para sugerir compras futuras a los clientes en función de un historial de sus compras previas

5.4 DISEÑO DEL CLIENTE NFS PREDICTIVO

5.4.1. Introducción

De toda la bibliografía expuesta se puede concluir que existe una gran cantidad de investigadores centrados en técnicas de* prefetching *con distintos propósitos. Hay dos situaciones principales, los inmersos en el mundo de los sistemas operativos o los inmersos en las aplicaciones, y sus interfaces con el sistema operativo. Los que están orientados a dotar al propio sistema operativo de técnicas de adelantamientos de datos, para mejorar las prestaciones, utilizan distintas técnicas de adelantamiento de bloques de un mismo fichero, de reparto de información en disco eficiente para luego adelantar más fácilmente, o de propuestas de adelantamiento de bloques de los ficheros siguientes al actual. Estos últimos trabajos son los que más nos interesan a nosotros por la similitud de la idea, aunque nuestra propuesta va a ser mucho más agresiva y realista.

Tomamos de [190] la idea del algoritmo* Last Succesor*, que se basa en lo siguiente:* si se ha leído el fichero f2 después de f1, entonces siempre va a ser así. *Sobre esta idea básica se desarrollan otras posibilidades que incorporan probabilidades, número de accesos, etc. Pero la idea fundamental es la repetitividad en los accesos, es decir, como el núcleo no es capaz de adivinar, lo que sí va a poder hacer es, en función de lo que ha pasado antes, predecir qué es lo que va a pasar a continuación. Y esto es lo que nosotros vamos a incorporar en NFS, la capacidad de predecir el siguiente fichero a leer, en función de un historial almacenado de lo ya ocurrido antes, en accesos previos.

5.4.2. Contrucción y utilización del grafo de accesos

Como ya hemos comentado, en todos nuestros experimentos hemos utilizado el codificador de vídeo MPEG-2, como generador de tráfico multimedia y demandante de servicios.

Al partir de distintos patrones de carga, nos hemos podido hacer una idea de la cantidad de situaciones que se podrían plantear, así que teníamos que tener muy claro que no nos debíamos viciar con esta aplicación sino que siempre debíamos tener en mente la idea de generalidad.

Había varias preguntas fundamentales que responder, como:

¿Cuándo hacer prefetching? ¿De cuánta información? ¿Cómo hacerlo?

Para responder a ellas había que estudiar distintas situaciones usuales con aplicaciones multimedia.

Para responder a la primera, suponemos que en todas ellas va a haber una parte de lectura de datos y una parte de procesamiento de esos datos. Queremos lanzar un proceso que adelante los datos siguientes en paralelo con la etapa de procesamiento para aprovechar las etapas más ociosas del servidor. Por lo tanto, concluida la etapa de lectura de datos habrá que lanzar el proceso de prefetching.

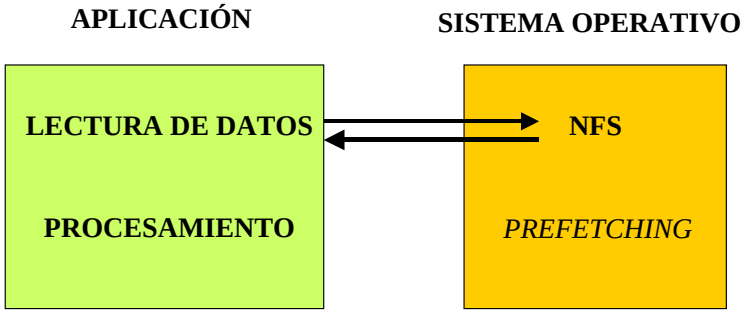


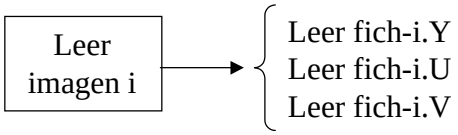
Figura 5.12. Realización del prefetching

Ahora hay que analizar en qué va a consistir la parte de lectura de datos, ya que habrá casos más simples en los que se corresponda con la lectura de un fichero individual y otros más complejos, que implique la lectura de distintos ficheros. Lo que sí tenemos claro es que hasta que no termine la etapa de lectura de datos no va a ser conveniente lanzar un proceso de adelantamiento de datos porque realmente no va a dar

tiempo, no se va aprovechar, y lo podemos decir con experiencia demostrada. El adelantamiento de ficheros se debe hacer durante una etapa de procesamiento en la que estemos seguros que va a dar tiempo a traer los datos a la cache del cliente, ya que sino esta técnica no reflejaría toda su potencia.

En el caso de la codificación MPEG-2, la lectura de una imagen implica leer consecutivamente tres ficheros (uno de luminancia y dos de crominancia). Una vez terminada la lectura de esos tres ficheros, se realizaría el prefetching de los tres siguientes, que en esta aplicación son los que se van a pedir en la siguiente etapa de lectura. Por tanto hay que plantear este proceso de forma general. Así que, podemos concluir, en primer lugar, que el prefetching se realizará cuando haya tiempo.

En la figura 5.8 se presenta el diagrama de actividad del codificador MPEG-2 de forma general. Vemos que hay un bloque donde se realiza la lectura de la imagen i, esto implica lo siguiente:



Para responder a la segunda pregunta y saber exactamente cuánta información adelantar, tampoco nos podemos dejar llevar por una aplicación en concreto, así que plantearemos lo siguiente: se adelantará información mientras se tenga tiempo.

Ahora vamos realmente a hacer posibles todas estas ideas y responder a la tercera pregunta de cómo hacer el prefetching. Ya hemos comentado que vamos utilizar un grafo de accesos, ya que es lo que se ha utilizado en la bibliografía de forma exitosa, y nosotros queremos aplicarlo realmente y optimizarlo con nuevas ideas.

Intentaremos no profundizar demasiado en detalles de implementación, pero alguno habrá que dar para que quede lo suficientemente claro. Empezamos describiendo su función, la información que en él se almacena, los momentos de utilización, y después, veremos un ejemplo.

Función: el grafo de accesos va a realizar dos funciones principales. En primer lugar utilizaremos el grafo de acceso como dispositivo para almacenar, en cada nodo, los ficheros que se han ido pidiendo al servidor para leer, así como otra información útil que más adelante comentaremos. En segundo lugar, utilizaremos el grafo como estructura de conocimiento histórico, en donde se va buscar el fichero (o el nodo) que se va a necesitar a continuación, en función de un orden de acceso a ficheros anterior.

Información almacenada: Para explotar toda la funcionalidad anterior se debe tener almacenado en cada nodo alguna información adicional, además del nombre del fichero en cuestión. La estructura de un nodo será algo como esto:

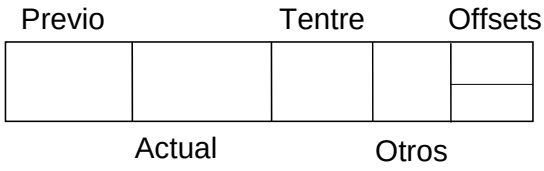


Figura 5.13. Estructura de un nodo del grafo

Previo: contiene el nombre del fichero accedido con anterioridad. Será el previo al actual

Actual: contiene el nombre del fichero que se acaba de ser accedido

Tentre: es el tiempo transcurrido entre la lectura del fichero previo y el actual. Si este tiempo no es superior a un determinado umbral, no se adelantará el fichero (ya que no va a dar tiempo). Sin embargo, si este tiempo es superior a ese umbral sí se adelantará.

Otros: también se debe guardar otra información fundamental para la localización de los ficheros, como son los inodos correspondientes y cierta información del fichero almacenada en la estructura struct file que almacena la información fundamental del archivo y su localización

Offsets: guardará los offsets de inicio y fin para el fichero actual. Así se especificará el trozo de datos leídos en el primer acceso.

Veamos un ejemplo de peticiones típicas al servidor en el cliente NFS tradicional:

```
VFS: nfs: read(composed/comp0.Y, 4096@0)
VFS: nfs: read(composed/comp0.Y, 4096@4096)
VFS: nfs: read(composed/comp0.Y, 4096@8192)                    ①
...
VFS: nfs: read(composed/comp0.Y, 4096@410624)
VFS:nfs: flush(8/2051)
VFS: nfs: read(composed/comp0.U, 4096@0)
VFS: nfs: read(composed/comp0.U, 4096@4096)
VFS: nfs: read(composed/comp0.U, 4096@8192)                    ②
...
VFS: nfs: read(composed/comp0.U, 4096@100352)
VFS:nfs: flush(8/2053)
VFS: nfs: read(composed/comp0.V, 4096@0)
VFS: nfs: read(composed/comp0.V, 4096@4096)
VFS: nfs: read(composed/comp0.V, 4096@8192)                    ③
...
VFS: nfs: read(composed/comp0.V, 4096@100352)
VFS:nfs: flush(8/2055)
VFS:nfs: write(encoded/comp0.Y.0(55319), 4096@0)
...
VFS:nfs: flush(8/55319)
VFS:nfs: write(encoded/comp0.U.0(55341), 4096@0)               ④
...
VFS:nfs: flush(8/55341)
VFS:nfs: write(encoded/comp0.V.0(55360), 4096@0)
...
VFS:nfs: flush(8/55360)
VFS: nfs: read(composed/comp3.Y, 4096@0)
VFS: nfs: read(composed/comp3.Y, 4096@4096)
VFS: nfs: read(composed/comp3.Y, 4096@8192)                    ⑤
...
VFS: nfs: read(composed/comp3.Y, 4096@410624)
VFS:nfs: flush(8/61234)
VFS: nfs: read(composed/comp3.U, 4096@0)
VFS: nfs: read(composed/comp3.U, 4096@4096)
VFS: nfs: read(composed/comp3.U, 4096@8192)                    ⑥
...
VFS: nfs: read(composed/comp3.U, 4096@100352)
VFS:nfs: flush(8/61245)
....
```

Esta es una traza típica del tráfico NFS que genera MPEG-2. En 1, 2 y 3 aparece la lectura consecutiva de los ficheros correspondientes a la primera imagen (comp0.Y, comp0.U y comp0.V), después se realizará la codificación local en cada máquina, y después, como vemos en 4, la escritura de los ficheros codificados en el servidor. Tras todo esto, y una vez realizada la codificación de la primera imagen, se lee la segunda. En 5 aparece la lectura del fichero de luminancia y del primer fichero de crominancia, correspondientes a la segunda imagen, y así se sigue hasta completar el proceso.

En este caso convendría hacer prefetching mientras se está codificando, por tanto, en el espacio de tiempo que hay entre 1, 2, 3 y 4.

No va a convenir adelantar un fichero tras leer comp0.Y, por ejemplo, porque no va a dar tiempo (esto está comprobado y lo expondremos en el capítulo siguiente de resultados). Como se ve en la traza anterior, después de realizar la última petición de lectura para comp0.Y, y tras una llamada a flush (limpia las páginas sucias y chequea posibles errores de escritura), se pide casi inmediatamente empezar a leer el siguiente fichero, comp0.U. Mientras que tras la lectura de comp0.V, sí va a transcurrir un espacio de tiempo ideal para pedir los siguientes ficheros. Así, mientras se está codificando se piden al servidor los ficheros comp3.Y, comp3.U, y comp3.V (que son los que corresponden por el orden de codificación de MPEG-2) y después, cuando se vayan a leer estos ficheros por la aplicación, no se producirá petición NFS porque esos datos ya van a estar en la cache, y por tanto no se tendrían que realizar 5 y 6.

El grafo que se crearía con estos primeros accesos sería el siguiente:

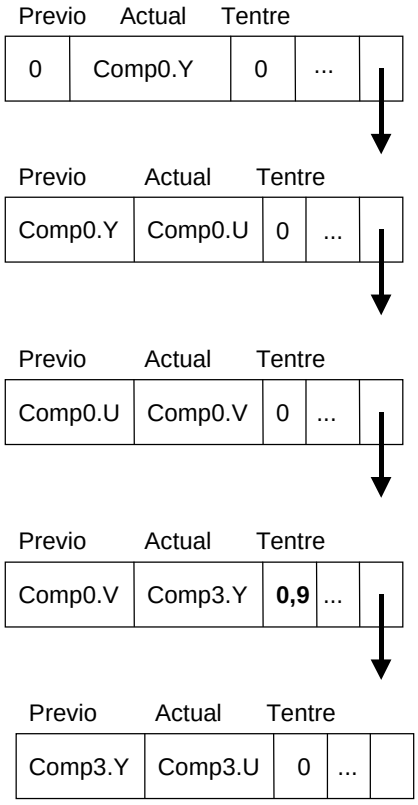


Figura5.14. Ejemplo de grafo de accesos

Este grafo se va generando la primera vez que se accede a los ficheros. Después, cuando se vuelve a acceder a ellos lo utilizaremos para determinar el siguiente fichero: La toma de decisiones en ese segundo acceso es como sigue:

Se lee comp0.Y, como no está en la cache, se envían las peticiones correspondientes al servidor. Al llegar al final de ese fichero, se llama a la función de prefetching, que va a buscar en el grafo si comp0.Y aparece en el campo previo de algún nodo. Sí encuentra un nodo, el segundo nodo de la figura anterior, por lo tanto comp0.U va a ser el fichero que se va a leer después. Esto quiere decir que hay que leer el fichero que aparece en el campo Actual porque es el que se va a solicitar a continuación, además toda la información, almacenada en los campos que hemos etiquetado como otros se refiere a ese fichero actual. Así que la función de prefetching devolvería este nodo como resultado de la búsqueda. Pero todavía, hace falta una última e importante comprobación. Se va a examinar el tiempo ente lecturas almacenado en el campo Tentre. En este caso el campo está a 0. Para esta aplicación se podría especificar un umbral de, por ejemplo, 0,5 segundos. Así, el valor almacenado en este indica que es un tiempo demasiado pequeño como para hacer prefetching. Por lo que, en este caso, la función de prefetching no va a realizar ninguna lectura al servidor.

Después se lee comp0.U. Tampoco está en la cache, por lo que se realizan las correspondientes lecturas al servidor. Tras la última petición se llama a la función de prefetching. Ésta busca en el grafo si comp0.U está en el campo previo de algún nodo. Efectivamente lo encuentra y concluye que el fichero siguiente va a ser comp0.V. Pero examina el campo Tentre y comprueba que el valor es 0, menor que el umbral, luego no devuelve nada.

Después se lee comp0.V. Se realizan las correspondientes peticiones de NFS y tras la última, se llama a la función de prefetching. Ésta busca si comp0.V está en el campo previo de algún nodo. Lo encuentra y resulta que el fichero que está en el campo actual es comp3.Y, que es el siguiente en la secuencia. Falta hacer la última comprobación acerca del tiempo entre peticiones de lectura. Ahora el valor almacenado sí es mayor que un umbral fijado para esta aplicación (pero que también se podría generalizar con el estudio correspondiente). Así que la función de prefetching va a realizar la correspondiente lectura del fichero almacenado en el campo actual, pero no sólo eso. Se ha llevado un contador de ficheros consecutivos leídos y está a tres (comp0.Y, comp0.U, comp0.V) con lo que se va a hacer la lectura de tres ficheros consecutivos. El especificado en el campo actual del nodo encontrado, y los localizados en los campos actual de los dos nodos siguientes. Por lo tanto se van a leer tres ficheros de forma consecutiva, con lo que se repite con el prefetching, el patrón seguido en el desarrollo actual de la aplicación

Ahora se lee comp3.Y, y la situación es nueva, los datos ya se han traído a la cache por lo que no va a haber peticiones a través de la red al servidor sino que los datos se van a localizar en la cache del cliente. Pero el proceso continúa, y al terminar la lectura del comp3.Y, se busca si se encuentra en el campo previo de algún nodo. Se encuentra pero el valor de Tentre es igual a cero, por lo que no se hace nada más. Y así sucesivamente se realizarían las sucesivas llamadas a la función de prefetching para que adelante en el momento adecuado los ficheros.

5.4.3. Cuestiones de implementación

Hay algunos detalles fundamentales que queremos mencionar en esta sección, son los siguientes:

1. Hemos modificado el fichero /usr/src/linux/fs/nfs/file.c, desde donde se llama a la función de prefetching. Esta función junto con otras que realizan distintas tareas (crear grafo, insertar nodo, encontrar nodo, etc.) así como las correspondientes estructuras de datos se han implementado en un fichero graph.h, que se incluye en file.c y que se une al código del cliente NFS recompilando el kernel.

2. También se ha incluido en el kernel un fichero cabecera, utime.h, con distintas funciones para tomar tiempos en el kernel (tarea mucho más compleja de lo que puede parecer) implementada por Juan Piernas de la Universidad de Murcia

3. Se ha modificado el fichero /usr/src/linux/mm/filemap.c. Este fichero es el que contiene la función genérica de lectura comentada anteriormente. En ella se ha incluido una parte especial del prefetching para leer de una, todo el trozo que se quiere adelantar. La forma normal de leer un fichero, cuando existe petición expresa de la aplicación, es mediante sucesivas llamadas a la función nfs_file_read, como se puede ver en la figura 5.8. Cuando se activa el prefetching no hay petición de la aplicación, no va a haber distintas llamadas a nfs_file_read, sino que es desde dentro de ésta, la última vez que se llama para un fichero determinado, cuando se arranca el prefetching. Así se tiene que implementar en una función de más bajo nivel (como es generic_file_read) el bucle que realice la lectura del trozo completo de datos.

4. También se ha tenido que añadir una función para leer los datos del fichero adelantado en el espacio de memoria del kernel, y no en el espacio de memoria del usuario, que es como normalmente se hace.

5. Los campos offset de inicio y offset final almacenados en cada nodo van a indicar el trozo de datos que se ha leido de un fichero. Este trozo será el mismo que se adelante con el prefetching. Si el patrón de carga es el 1, entonces esos contadores indicarán el trozo leído, que puede ser distinto para los distintos clientes en función del número de ellos que participe en la codificación. Si el patrón es el 2 (todos leen todo), entonces el inicio y el final se corresponderán con el fichero completo a leer. Esto es bastante novedoso porque hasta el momento sólo se adelantaban bloques de datos, pero con esta técnica de prefetching se puede adelantar cualquier trozo, sea del tamaño que sea.

6. Se ha utilizado un contador de ficheros consecutivos, que determinará el número de ficheros que se adelantan. Así, se sigue el patrón de lectura que marque la aplicación en concreto que se está tratando. Para la codificación en paralelo, se leen tres ficheros consecutivos, luego, lo más lógico es que, también, se adelanten con el prefething tres ficheros consecutivos.
7. Vamos a interpretar de forma general el valor del campo tentre. Este valor se va a comparar con un umbral de prefetching fácilmente medible. En nuestro entorno de cluster dedicado, sólo hemos utilizado la codificación de vídeo en sus distintas versiones y patrones. Aún con esta diversidad ha sido muy fácil estimar el tiempo que conlleva el procesamiento de las imágenes, debido a los cientos de experimentos previos que teníamos. Con lo que el cálculo del umbral de prefetching ha sido muy simple. Este umbral se puede poner como un parámetro del kernel y explicitarlo en algún fichero del /proc, de donde el kernel toma ciertos valores de entrada que necesita. El tema está en cómo determinar este umbral en un cluster mixto, donde se puede ejecutar distinto tipo de aplicaciones multimedia. Habría que hacer un estudio previo del tiempo de procesamiento de los clientes NFS involucrados y cambiar este parámetro. Como usualmente lo que tendremos es o un cluster dedicado de ASP's, o de servidores web, o de servidores de base de datos, o de vídeo bajo demanda, sería fácilmente estimable este umbral. Pero debe quedar claro, que este parámetro no debe ser un inconveniente para esta técnica, por supuesto, se puede dejar este valor a 0, y se haría prefetching de todos los ficheros a los que se accede, que puede ser el caso más general, y que no va a depender de ningún tipo de aplicación particular, unas le sacarán más partido y otras menos.

8. Habría muchos detalles propios del kernel que ahora aislados pueden resultar demasiado aburridos, así que los que no hayamos comentado, aparecerán en el capítulo siguiente de resultados, cuando tengamos que explicar algunas de las situaciones conflictivas con las que nos hemos tenido que enfrentar. Lo que sí quiero reflejar es la enorme complejidad del entorno de trabajo en el que me he desarrollado la nueva técnica de prefetching. La depuración del código no existía y toda necesaria comprobación y prueba ha sido a base de recompilaciones sucesivas e impresión de mensajes del kernel. Ha sido un trabajo largo y constante, pero en el capítulo siguiente se verá la recompensa a todas estos problemas

CAPÍTULO 6

NUEVAS PRUEBAS CON EL CLIENTE NFS PREDICTIVO MULTIHILO

6.1. INTRODUCCIÓN

Una vez implementada la técnica de prefetching explicada en el capítulo anterior y tras sucesivos refinamientos, se ha llegado a una versión definitiva de la que se ha realizado todo un completo banco de pruebas, cuyos resultados se expondrán en este capítulo.

Expondremos los resultados obtenidos para los tiempos de lectura con los patrones de carga uno y dos, ya que son los más representativos de entre todos los clasificados.

La incorporación de esta nueva técnica de prefetching en el cliente NFS va a permitir la lectura anticipada de datos, y por tanto intenta eliminar los fallos de cache en el cliente.

En cada cliente NFS coexistirán dos procesos:

- El proceso secuencial normal que ejecuta los procedimientos oportunos para realizar todas las peticiones al cliente
- El proceso hilo que se genera cada vez que el proceso secuencial acaba la lectura de datos, y que en los primeros accesos a ficheros, construye el grafo de accesos, y que, en los siguientes, busca, en ese grafo, la información necesaria para pedir al servidor, el fichero, que a continuación va a necesitar el proceso secuencial

Tan sólo queremos hacer hincapié en la forma o en el momento en que se crea ese proceso hilo que adelantará los datos a la cache del cliente. Ya comentamos en el

capítulo cinco, los trabajos relacionados con el prefetching, y vimos, como, la forma tradicional de hacer prefetching está totalmente basada en bloques de disco. Existen distintas técnicas para adelantar bloques de un mismo fichero, o bloques del fichero que viene a continuación. Nosotros en nuestro trabajo queremos tratar el tema de las aplicaciones multimedia en general, para ello hemos implementado distintos patrones de carga en el codificador de vídeo, poniendo de manifiesto que distintas aplicaciones tendrán distintas necesidades de lectura de datos. Aunque en las aplicaciones en paralelo el reparto de datos se lleve a cabo para la reducción de los tiempos de ejecución total, puede que ese reparto de datos afecte al procesamiento que con ellos se hace, pero no a la lectura de los mismos. Es decir, puede que distintas aplicaciones paralelas en las que interviene un determinado número de clientes, realicen un reparto de los datos a procesar por cada una de ellos, pero quizás los clientes necesiten todos los datos de entrada por algún motivo, aunque solo vayan a procesar determinado trozos de datos. Por eso, nos planteamos que debíamos reflejar estos casos.

La técnica de prefetching está pensada para adelantar el trozo de datos que necesite cada uno de los clientes que colaboran en el procesamiento paralelo de una determinada aplicación multimedia. El concepto de adelantar bloques se ha desechado totalmente y ahora lo que se va a adelantar son trozos de datos. Además, el sistema tiene que aprender cuál es el patrón de lectura de una determinada aplicación, es decir, si se lee fichero a fichero, o si se lee un número consecutivo de éstos antes de que comience el procesamiento de los datos que le corresponden.

La restricción más fuerte la plantea el patrón 1 de carga, donde cada cliente lee sólo una franja de la imagen. En estos casos puede que ciertos clientes no lleguen al fin de fichero. Así, que esta marca sólo se podría utilizar para el patrón de carga 2 donde sí se leen los ficheros completos de toda la secuencia. Esto se resuelve rápidamente sabiendo que inmediatamente después de leer los datos, el cliente hace una llamada a la función nfs_file_flush, por lo tanto, será en esta función en donde se cree el proceso hilo, que en el primer acceso a un fichero, creará un nodo del grafo para almacenar toda la información correspondiente (previo, actual, tentre, offsets, otros), y que, en los sucesivos accesos, buscará el nodo que se debe leer a continuación, si es que hay tiempo para ello.

La ejecución de esta técnica no va a suponer sobrecarga ninguna para el cliente, ya que será otro proceso diferente el que se ocupe de la creación y búsqueda en el grafo. Lo único que tendremos son más procesos en el cluster, más procesos enviando peticiones al servidor, pero que no van a coincidir nunca en el tiempo, con los procesos secuenciales, ya que la ejecución de los procesos hilos está planificada para que se realice justo mientras los procesos secuenciales están procesando su información de forma local, y por lo tanto, se quiere aprovechar los intervalos ociosos del servidor.

Quizás sea este el momento de hablar, también, del coste que conlleva este grafo de accesos. Como se ha expuesto en el capítulo 3, donde se presentan las características de las máquinas que forman el cluster, actualmente los clientes son pentiums II con 728 megabytes de memoria. Pero esto no ha sido siempre así. Inicialmente las máquinas tenían 128 megas de memoria. Con esta capacidad se

realizaron todas las pruebas experimentales del capítulo 4. Pero, antes de incorporar las mejoras en el kernel, decidimos aumentar la capacidad de las máquinas, porque continuamente hay que ir actualizando el cluster para mantenerlo fresco y joven. Este aumento de memoria no es problemático ya que hemos vuelto a tomar los tiempos de lectura para el kernel tradicional sin prefetching para poder comparar con la nueva técnica. Es una cuestión bastante simple. Los problemas que aparecen en el capítulo 4, sobre todo la figura 4.16 en la que aparecen todos los patrones de datos, con 128 megabytes de memoria, volverán a aparecer de igual forma ahora con 728 megabytes pero para un número mayor de imágenes, ya que todos los resultados presentados en esta tesis son para 400 imágenes. Por lo que decidimos no repetir todo el estudio que ya teníamos, sino repetir los experimentos que nos van a servir para comparar directamente con los resultados obtenidos con el kernel predictivo.

Bueno, todo esto lo hemos contado para hablar del coste que puede conllevar la creación y mantenimiento del grafo. Cada uno de los clientes va a generar un grafo de accesos que después va a utilizar. En principio, hemos tenido más que suficiente con la RAM actual, y de hecho nunca se han llegado a ocupar más de 400 megabytes de memoria. Claro está que el grafo más completo que han mantenido los clientes en su memoria ha estado formado por 1200 nodos. ¿Qué va a pasar cuando se construyan grafos más grandes? Pues absolutamente nada, porque la memoria ya no es ningún problema en los PCs actuales, y sobre todo, porque nos vamos a plantear futuros algoritmos de mantenimiento de ese grafo para que no esté siempre completamente en memoria, por lo que la forma presentada en este trabajo no va a ser la definitiva pensada para nuestra técnica, pero sí nos va a servir para presentarlo, antes de seguir trabajando en sus mejoras.

6.2. ORGANIZACIÓN DE LAS PRUEBAS

Con la técnica de prefetching se va a conseguir adelantar trozos de los siguientes ficheros antes de que se necesiten. El caso más general supone que ese trozo sea exactamente el fichero completo. Así que empezaremos la exposición mostrando los resultados obtenidos para el patrón de carga 2, en el que todos los clientes van a leer los ficheros completos aunque después sólo necesiten un trozo para codificar su parte correspondiente.

A continuación, veremos el caso particular en el que los clientes leen trozos del fichero. Esto se corresponde con el patrón 1 de carga, en el que la imagen se divide en franjas y cada cliente lee y codifica una de esas franjas de un tamaño determinado.

Esta será la clasificación global de las pruebas. Plantearemos resultados obtenidos con el patrón 2 o prefetching completo de datos, y con el patrón 1 o prefetching parcial de datos. Para cada uno de estos estudios se han realizado distintos experimentos que se exponen en las dos secciones siguientes.

6.3. RESULTADOS OBTENIDOS CON PREFETCHING COMPLETO

La técnica de prefetching definitiva no ha surgido de casualidad ni al primer intento, ha seguido un proceso muy elaborado y continuos refinamientos y mejoras, aunque el entorno para ello haya sido el menos adecuado posible. Por lo tanto, aunque no vamos a presentar el diario de los experimentos, sí vamos a exponer, de forma general, como se llegó a la versión definitiva y, sobre todo, las principales etapas seguidas, de ahí que hayamos dividido este apartado en algunos subapartados, para facilitar la exposición de los datos de mayor interés.

6.3.1. Primeras pruebas

La idea inicial no estaba tan elaborada como la planteada en el capítulo anterior. Inicialmente en cada nodo del grafo sólo se guardaba la información de cada fichero (nombre e inodo). No se almacenaba ni el tiempo entre peticiones de lectura y tampoco se tenía en cuenta el patrón seguido por la aplicación, es decir, el número de ficheros leídos antes de hacer prefetching. Estos dos últimos parámetros están totalmente relacionados y se incorporaron juntos. Antes de incluirlos se hacía prefetching tras la lectura de cada fichero. Siempre se hacía la predicción fichero a fichero. Estos son las primeros resultados que vamos a exponer.

En la comparación de resultados vamos a utilizar dos kernels distintos, uno sin prefetching (el kernel tal cual viene con la distribución de Linux) y nuestro kernel con prefetching (con el cliente NFS modificado).

De cada uno de los experimentos se han lanzado 10 ejecuciones para asegurarnos la fiabilidad de los datos. Entre cada una de las ejecuciones tenemos que limpiar la cache del cliente y la cache del servidor, ya que si quedaran restos en alguna de estas caches los resultados no serían independientes y no se podría ver con claridad el efecto de esta técnica. Por lo tanto es muy importante asegurarnos que entre ejecuciones las caches están totalmente limpias.

En el kernel sin prefetching, como ahora veremos, en las 10 ejecuciones deben obtenerse resultados similares. Así quedará demostrado que las caches han quedado limpias y que cada nueva ejecución implica una réplica exacta de todo el proceso. El cliente no va a tener nada en su cache y va a hacer todas las peticiones, de todas las imágenes que forman la secuencia, al servidor. Por otra parte, el servidor, no tendrá tampoco nada en su cache y, ante la primera petición de un fichero, deberá acceder a disco para volcarlo a la cache y de ahí reenviarlo a los clientes que lo soliciten.

Con el kernel con prefetching la situación va a ser distinta. La primera ejecución supondrá crear el grafo de accesos (ya que son los primeros accesos a los ficheros). En las siguientes (cuando se vuelve a acceder a los ficheros), como el grafo ya está

construido, se va a utilizar, se va a buscar en él el fichero siguiente, y esto implica que el prefetching va a demostrar su utilidad. Por lo tanto, en la primera ejecución con este kernel, los resultados deben ser similares a los obtenidos con el kernel sin prefetching, y con las ejecuciones siguientes ya se debe apreciar una reducción en los tiempos. El proceso de limpieza de las caches se sigue manteniendo, ya que no deben quedar restos de la ejecución anterior. Será en una misma ejecución cuando se consulta el grafo, se busca el fichero siguiente y se pide al servidor la lectura del mismo. Todo esto lo realiza el proceso hilo de forma solapada con el proceso secuencial que estará procesando sus datos.

En la tabla siguiente aparecen los tiempos medios de lectura para 400 imágenes y 7 procesadores. Empezamos exponiendo estos resultados para comprobar la efectividad del método y no nublarnos con el cambio de otros parámetros. Vamos a enfrentar el caso más problemático en ambos clientes (el no predictivo y el predictivo), y después, ya expondremos todos los demás.

Tabla 6.1. ***Tiempos de lectura con 7 procesadores***

SIN PREFETCHING	1ª EJEC	SIG EJEC
T. LECTURA	48,53967	48,4587
CON PREFETCHING	1ª EJEC	SIG EJEC
T. LECTURA	48,5366	36,1123

Como podemos ver hay una reducción de tiempos en las siguientes ejecuciones si utilizamos el kernel con prefetching. Esto es una gran noticia ya que es la primera señal de que el prefetching logra reducir tiempos.

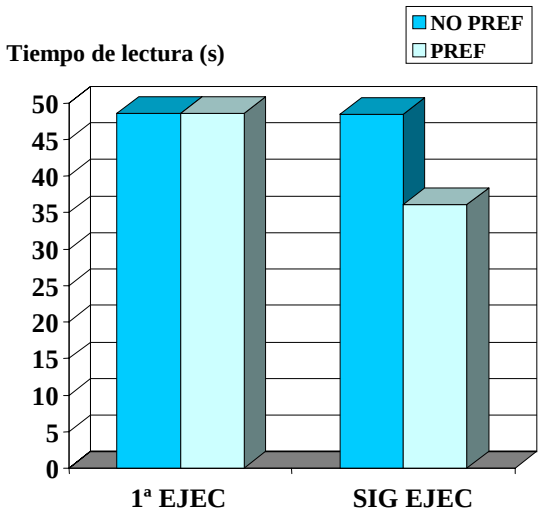


Figura 6.1. Tiempos de lectura para 7 procesadores y 400 imágenes

Hemos conseguido reducir el tiempo de lectura en un 26%, lo que es un indicativo excelente de que la técnica funciona y mejora el funcionamiento del kernel tradicional.

Pero, cuando empezamos, tampoco sabíamos exactamente hasta cuanto íbamos a poder mejorar, no sabíamos hasta donde podíamos llegar, ni cuáles eran los tiempos que queríamos conseguir. Para ello, intentamos buscar algún umbral comparativo que nos sirviese de objetivo final a perseguir. Y decidimos lo siguiente: si en el kernel sin prefetching no limpiamos la cache del cliente entre ejecuciones, se mantendrá información en la misma, y una vez acabada la primera ejecución parte de la secuencia (o casi toda) se mantendrá en la cache, con lo que al realizar la segunda ejecución, el cliente no tendrá que pedir los datos al servidor. Así que nos montamos este otro entorno como objetivo a cumplir. Los tiempos obtenidos con la cache sucia en el kernel sin prefetching, en la segunda ejecución, podría ser un ejemplo de lo que podríamos conseguir si se utiliza el prefetching en el kernel. Los datos obtenidos fueron los siguientes:

Tabla 6.2. Tiempos de lectura en el kernel sin prefetching con cache sucia

SIN PREFETCHING Y CACHE SUCIA	1ª EJEC	2ª EJEC
T. LECTURA	**48,53967**	3,7714

Los resultados son impresionantes, la reducción es de 93%. Por tanto, eso nos llevo a una nueva revisión de todo el proceso, a tomar tiempos de manera exhaustiva en el kernel, en cada función que se invocaba y a plantearnos nuevas modificaciones. En este momento examinamos los requisitos de nuestra aplicación, y comprobamos que parte del prefetching que se hacía se estaba desperdiciando.

En el kernel estábamos haciendo lo siguiente:

leer comp0.Y

hilo: leer comp0.U

leer comp0.U

hilo: comp0.V

leer comp0.V

hilo: comp3.Y

Mientras que en la aplicación se hace:

leer comp0.Y

leer comp0.U

leer comp0.V

codificar comp0

Esto implicaba que no se estaba aprovechando realmente el prefetching, porque aunque el proceso hilo pida los ficheros comp0.U y comp0.V, estas lecturas se van a solapar con las lecturas del proceso secuencial, lo que significa que además de no dar tiempo a adelantar los datos, se está cargando el servidor con más procesos, ya que no se le está pidiendo en los intervalos más ociosos sino en los de más trabajo. Sólo se adelantaría con utilidad el fichero comp3.Y porque una vez leídos los ficheros comp0.Y, .U y .V, el proceso secuencial ya se pone a codificar. Por tanto esto nos llevó a la conclusión de que se pueden plantear distintos patrones de lectura de la carga y que, por lo tanto, hay que tenerlos en cuenta.

6.3.2. Planteamiento definitivo

Veamos ahora los resultados obtenidos con los nuevos requisitos implementados en el kernel. Se va a tener en cuenta el tiempo transcurrido entre lecturas de ficheros distintos, para ver si conviene lanzar el prefethingo no, y se va a establecer un contador para determinar el número de ficheros leídos de forma consecutiva antes de lanzar el prefetching.

Así, para nuestra aplicación, se va a hacer prefetching cada tres ficheros, y los resultados obtenidos con estas modificaciones son los siguientes:

Tabla 6.3. Tiempos de lectura haciendo prefetching 3 a 3 para 7 procesadores

CON PREFETCHING 3 A 3	1ª EJEC	2ª EJEC
T. LECTURA	49,93	30,16587

La reducción de tiempos en la segunda ejecución es de un 40%. Se han rebajado los tiempos de la segunda ejecución, pero no tanto como esperábamos. Bien en verdad que haciendo prefetching fichero a fichero, lográbamos adelantar sólo el fichero de luminancia (.Y), pero es que este fichero, es justamente el que mayor información contiene. Su tamaño es de unos 400 kbytes, frente a los 100 kbytes que tienen cada uno de los ficheros de crominancia. Adelantábamos ya el 66% de los datos, de ahí que la reducción no sea mucho mayor.

Como el tiempo de lectura de la segunda ejecución nos parecía todavía muy lejano al tiempo que se obtiene sin prefetching y sin limpiar la cache, pensamos en plantearnos otro objetivo a cumplir, ya que la diferencia de tiempos en muy grande y la situación es que, realmente, no es la misma. Por tanto, decidimos implementar esta misma técnica de prefetching pero en la propia aplicación, para ver así, qué tiempos se obtienen, y tener otro punto de referencia de lo que quisiéramos obtener.

La forma de incluir prefetching en la aplicación se puede observar en la siguiente figura.

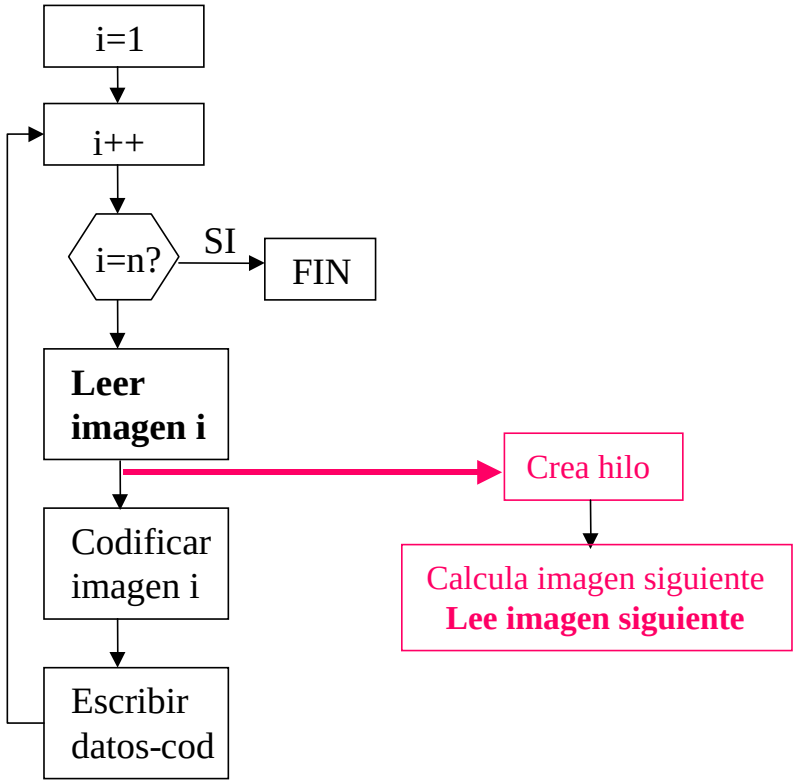


Figura 6.2. Prefetching en la aplicación

Como se observa en la figura, justo después de leer la imagen que va a ser procesada, se crea un hilo para leer la siguiente. Para el cálculo del siguiente no se ha utilizado ningua estructura de datos particular. Hemos implementado un mecanismo de cálculo del siguiente fichero en función del nombre del mismo y del orden de codificación de MPEG, que, como ya sabemos, es distinto del orden de display.

Esta implementación se ejecuta, claro está, con el kernel sin prefetching, lo que provoca el adelantamiento de datos a la cache del cliente NFS por parte expresa de la aplicación. Esta va a ser una medida mucho más cercana a nuestro caso de prefething, ya que implica que el kernel traiga los datos que la aplicación solicita.

Vamos a exponer los resultados obtenidos para esta técnica de prefetching en la aplicación:

Tabla 6.4. Tiempos de lectura con prefetching en la aplicación

CON PREFETCHING EN LA APLICACIÓN	1ª EJEC
T. LECTURA	25,5374

El tiempo obtenido está también bastante lejos del que se obtiene sin prefetching y con la cache sucia, por lo tanto, está claro que la situación es completamente distinta y que, por tanto, la comparación anterior no era realista, sino que la implementación del prefetching en la aplicación va a estar mucho más próxima a la implementación del prefetching en el kernel, sólo que al hacerlo en el kernel nos va a servir para todo tipo de aplicaciones.

Para demostrar que no se solapan el proceso hilo con el proceso secuencial, pusimos marcas en el kernel justo, antes de empezar a hacer prefetching de ficheros, al finalizar, y cuando el proceso secuencial va a leer los datos que el proceso hilo se ha traído a la cache. En la siguiente figura representamos estos tiempos para las 24 primeras imágenes (72 ficheros). En el eje vertical se representa el instante de tiempo en el que se realiza la petición. El significado de la leyenda es el siguiente: Antes, indica el instante de tiempo justo antes de hacer prefetching, es decir cuando se ha visto que sí va a dar tiempo a adelantar el fichero siguiente; Después, indica el instante de tiempo después de que el proceso hilo haya leído tres ficheros (el patrón que se sigue); Read, indica el instante en el que el proceso secuencial hace la primera petición de lectura sobre los ficheros que se ha traído a la cache el proceso hilo.

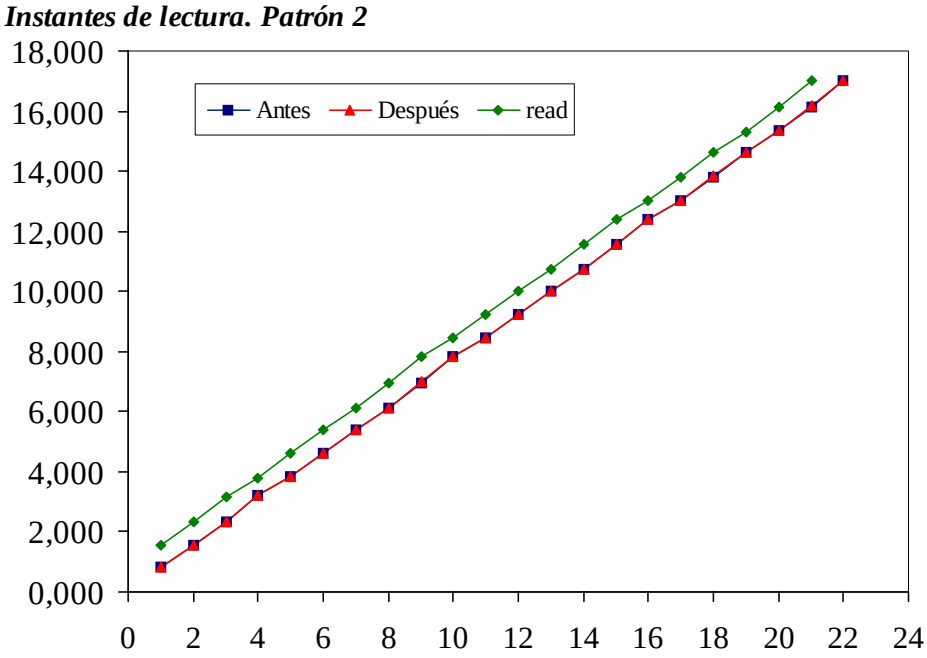


Figura 6.3. Instantes de tiempo antes y después del prefetching que realiza el proceso hilo, y antes de la primera petición de lectura del proceso secuencial

Antes de pasar a exponer y representar gráficamente los resultados con distinto número de procesadores, vamos a plantear otros experimentos realizados para ver que ocurre en otras situaciones.

6.3.3. Pruebas adicionales

Se realizaron las siguientes pruebas para buscar otros casos comparables con el que se plantea con el prefetching en el kernel. Veremos primera las características de cada prueba y, después, expondremos la tabla con los tiempos obtenidos en cada una de ellas.

Lectura local de datos

En esta prueba se utilizará el kernel sin prefetching. Cada cliente va a leer la secuencia de su propio disco duro local. No va a haber peticiones NFS al servidor a través de la red. Esta prueba no es nada realista, ya que las secuencias de vídeo originales correspondientes a una película completa ocuparían varios gigabytes por lo que no será viable tenerlas en el disco local, de ahí el uso de protocolos como NFS.

Copia de datos desde el servidor

En esta prueba donde se utiliza también el kernel sin prefetching, el servidor va a copiar la secuencia completa desde su disco duro al disco duro de los clientes. Después se limpia la cache del servidor, y se ejecuta el codificador, pero, especificando en los clientes que la secuencia la lean del servidor NFS.

Copia de datos por el cliente

Es similar a la anterior, pero ahora los clientes copian la secuencia desde el directorio compartido del servidor NFS a sus discos locales. Tras esta copia se ejecuta el codificador en paralelo especificando que la lectura de datos se haga desde el servidor. Esta va a ser otra forma de ensuciar la cache de los clientes.

Sincronización via Fast-Ethernet

Aparte de modificar las formas de leer la secuencia, vamos a probar también a cambiar la forma en la que se sincronizan los procesadores para realizar la codificación en paralelo, para ello hemos probado a realizar la comunicación MPI a través de Fast-Ethernet, en lugar de hacer toda la comunicación (de datos y de control) por Myrinet.

Tabla 6.5. Tiempos de lectura obtenidos en distintos casos (7 procesadores, 400 imágenes)

Sin prefetching cache limpia	48,53967
Sin prefetching cache sucia	3,7714

Con prefetching en el kernel	30,16587
Lectura local de datos	30,269
Copia de datos desde el servidor	20,947
Copia de datos por el cliente	4,0153
Sincronización Fast-Ethernet	58,7728

Como se preveía el caso de copia de datos por el cliente es bastante similar al caso de cache sucia en el kernel sin prefetching, ya que la secuencia está en la cache antes de ejecutar la aplicación, por lo que los tiempos se ven enormemente reducidos.

En el caso de copia de datos desde el servidor no va a haber tráfico NFS, ya que es el propio servidor el que copia los datos en los discos duros de los clientes. Cuando son los clientes los que copian los datos sí que hay peticiones NFS de los clientes a los servidores para poder leer esos datos y copiarlos a sus discos duros correspondientes, con lo que las caches NFS están repletas de información.

Para el caso de lectura local de datos, podemos ver que los tiempos son menores que los obtenidos de forma tradicional compartiendo datos con NFS (sin prefetching), lo que indica que el caudal de datos del disco va a ser mayor que el da la red, pero este es un caso que no se puede plantear ya que no se podría dar con aplicaciones reales.

Por último, vemos que los resultados son mucho peores si se realiza la sincronización entre procesadores a través de la Fast-Ethernet, por lo que seguiremos haciéndolo como hasta ahora, es decir, todo el tráfico a través de Myrinet.

6.3.4. Resultados con distinto número de procesadores

Llega el momento de ver que pasa en otros casos, con distinto número de procesadores y ver así la tendencia de reducción de tiempos que obtenemos con el aumento en el número de procesadores.

A continuación exponemos una tabla con los tiempos medios de lectura para distinto número de procesadores. En principio vamos a comparar los valores de la segunda ejecución, ya que es a partir de esa ejecución cuando se aprecian los efectos del prefetching. Recordamos que estos tiempos medios se han obtenido de la media aritmética de 10 ejecuciones.

Tabla 6.6. Tiempos de lectura con distinto número de procesadores (patrón 2)

# PROC	**SIN *PREFETCHING***	**CON *PREFETCHING***
1	38,107	30,7039
3	35,39665	22,2877

5	37,4023	28,0266
7	46,6787	30,16587

La reducción es considerable en todos los casos, tan solo para 1 procesador parece que la reducción es menor. Vamos a representar gráficamente estos resultados

Hemos querido representar en esta gráfica tres tiempos distintos. Primero, los obtenidos sin prefetching, que ponen de manifiesto el problema que se quiere resolver, el problema de aumentos de tiempos con el aumento en el número de procesadores. Estos tiempos se obtienen de la media de 10 ejecuciones, entre las cuales se limpia la cache del servidor y de los clientes. Después, los tiempos obtenidos con prefetching. Distinguimos la primera ejecución del resto. Lo hacemos así para ver que durante esta ejecución, que es cuando se crea el grafo en el kernel, no va a haber problemas de sobrecarga en el cliente y no va a haber ningún efecto negativo en su funcionamiento normal, de ahí que los tiempos obtenidos sean casi idénticos a los obtenidos sin prefethingo. Por último, representamos los tiempos obtenidos utilizando la técnica de prefetching, donde se aprecia una importante reducción en los tiempos de lectura. Por supuesto, también en estas ejecuciones limpiamos la cache de los clientes y del servidor.

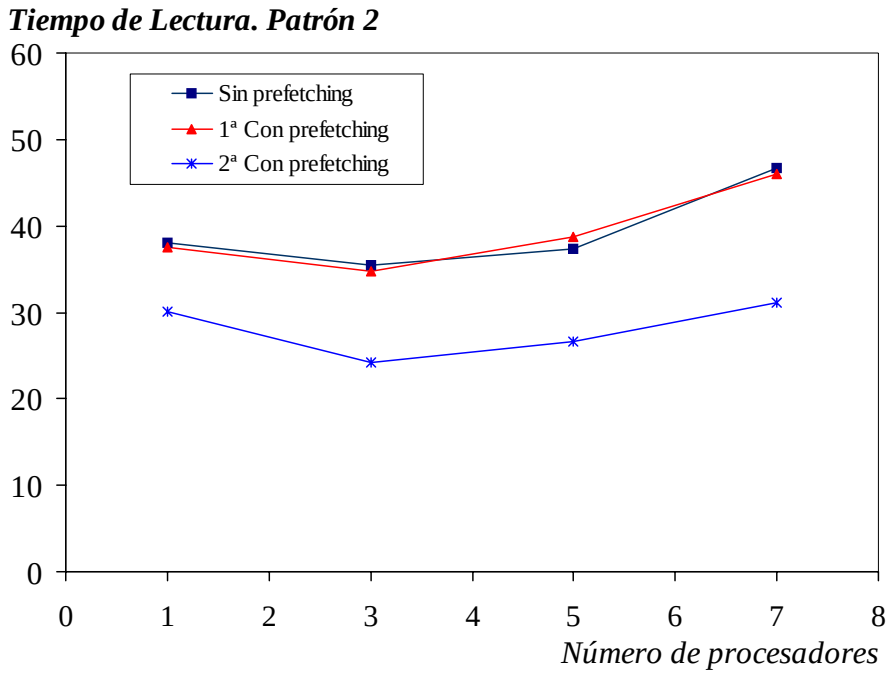


Figura 6.4. Tiempos de lectura con distinto número de procesadores

Como lo que representamos es la media de las diez ejecuciones, vamos a señalar los intervalos de confianza, donde aparecerán también los tiempos mayores y menores y no sólo el caso medio como ocurre en la gráfica anterior.

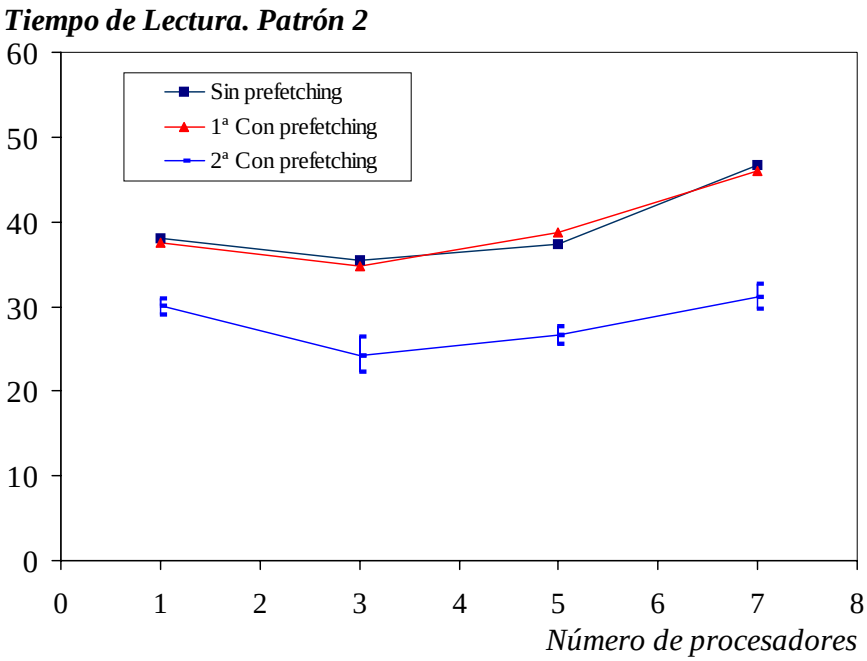


Figura 6.5. Intervalos de confianza

Podemos observar una variación mayor para el caso de 3 procesadores. Recordemos que todos leen la misma cantidad de información, pero ahora como no van a empezar los tres procesadores a leer exactamente en el mismo instante de tiempo, se van a producir adelantamientos, esto va a provocar que algunos procesadores saquen ventaja y pidan al servidor los datos que tiene ya en su cache, por eso se obtiene una reducción de tiempos importante al pasar de 1 procesador a 3, ya que en ese caso tampoco está el servidor demasiado cargado. Los problemas aparecerán a partir de 5 procesadores. El aumento de los mismos va a conllevar un aumento en los tiempos de lectura. Lo mismo ocurre con el aumento de 5 a 7 procesadores, aunque ya se puede observar como ese aumento no es tan pronunciado como en el caso sin prefetching.

6.3.5. Tendencias de interés

Como en nuestro cluster solo disponemos de 8 máquinas, no podemos ver que ocurriría si seguimos aumentando el número de procesadores. En la gráfica anterior se puede ver cómo el aumento de los tiempos con el número de procesadores no es demasiado pronunciado para los resultados con *prefetching*, pero, queremos saber que ocurriría si utilizásemos más procesadores. Para ello vamos a utilizar líneas de tendencia en gráficos. Estas líneas se usan para analizar los problemas de predicción. Este análisis también se denomina *análisis de regresión*. Con este análisis puede ampliarse una línea de tendencia en un gráfico hacia adelante o hacia atrás de los datos actuales para que muestre la tendencia.

El análisis de regresión es una forma de análisis estadístico utilizado para hacer previsiones. Estima la relación entre variables de modo que se puede predecir una variable a partir de otra u otras variables.

Para el cálculo de líneas de tendencia se pueden utilizar varias ecuaciones (lineal, logarítimica, exponincial, potencial, etc.), representaremos algunas para comparar. En las siguientes figuras comparamos la lineal, la exponencial, la potencial y la logarítmica.

Las ecuaciones para el cálculo de líneas de tendencia son:

Lineal: calcula el número mínimo de cuadrados en una línea que representa la siguiente ecuación: $y = mx + b$, donde m es la pendiente y b es la intersección.

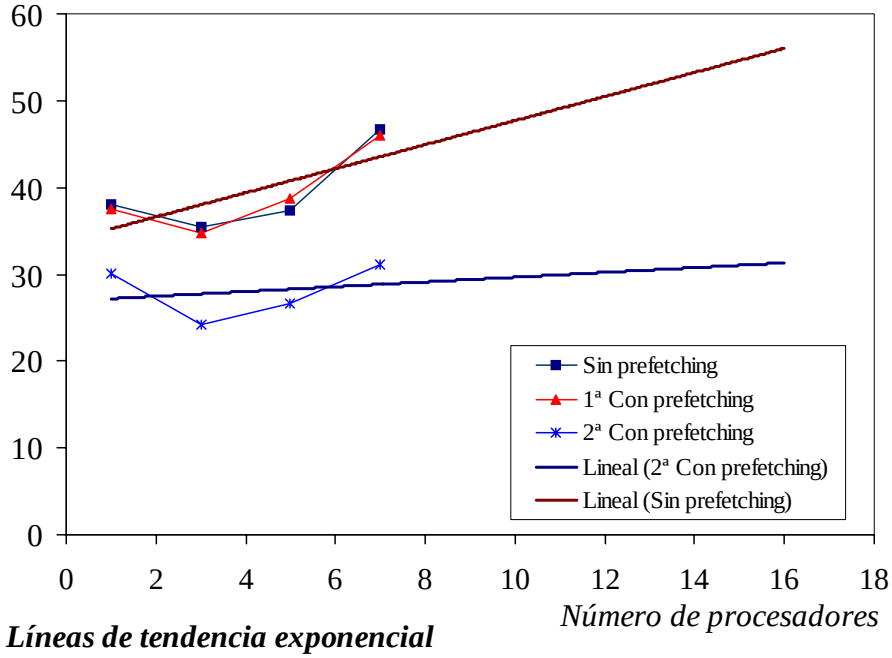
Logarítmica: calcula el número mínimo de cuadrados mediante puntos utilizando la siguiente ecuación: $y = c \ln x + b$, donde c y b son constantes y ln es el logaritmo natural (neperiano).

Exponencial: calcula el número mínimo de cuadrados mediante puntos utilizando la siguiente ecuación: $y = ce^{bx}$, donde c y b son constantes.

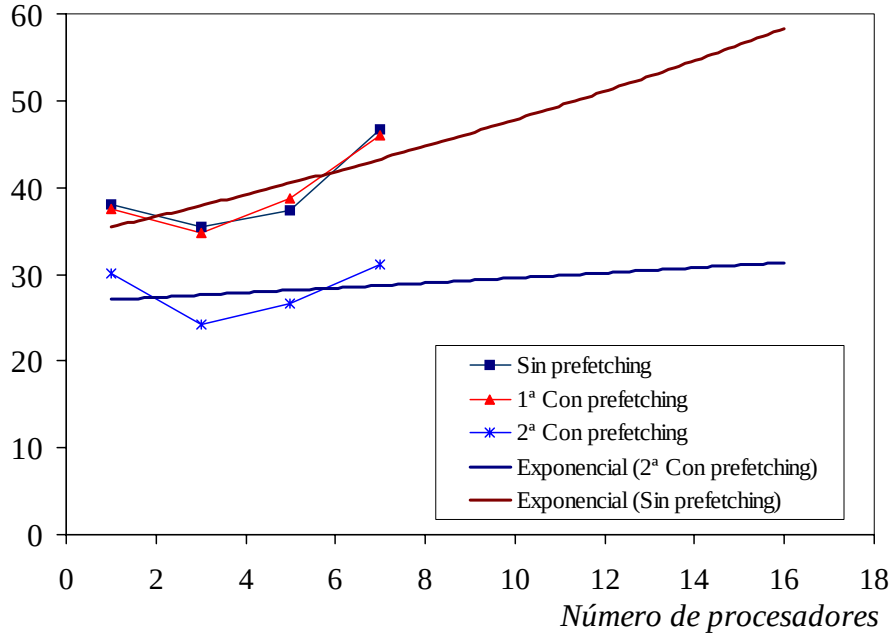
Potencial: calcula el número mínimo de cuadrados mediante puntos utilizando la siguiente ecuación: $y = c\, x^{b}$, donde c y b son constantes

En las dos figuras siguientes hemos querido representar las líneas de tendencia para un cluster de hasta 16 procesadores. Con estas líneas veremos la dirección que siguen los tiempos de lectura, es decir, hacia donde se mueven. Se ha seguido una tendencia alcista (como se diría en bolsa), ya que la representación de las líneas se basa en la unión de sucesivos mínimos. En todas ellas, la línea de tendencia, para los resultados sin prefetching, ponen de manifiesto el crecimiento de los tiempos con el número de procesadores, de forma acusada. En la gráfica de la línea de tendencia exponencial este crecimiento es bastante importante. Mientras que para los valores con prefetching estas

Líneas de tendencia lineal



Líneas de tendencia exponencial



líneas se mantienen, no hay un crecimiento claro con el aumento en el número de procesadores, como ocurre con los resultados sin prefetching, por tanto, es un dato muy interesante a tener en cuenta.

Figura 6.6. Líneas de tendencia lineal y exponencial

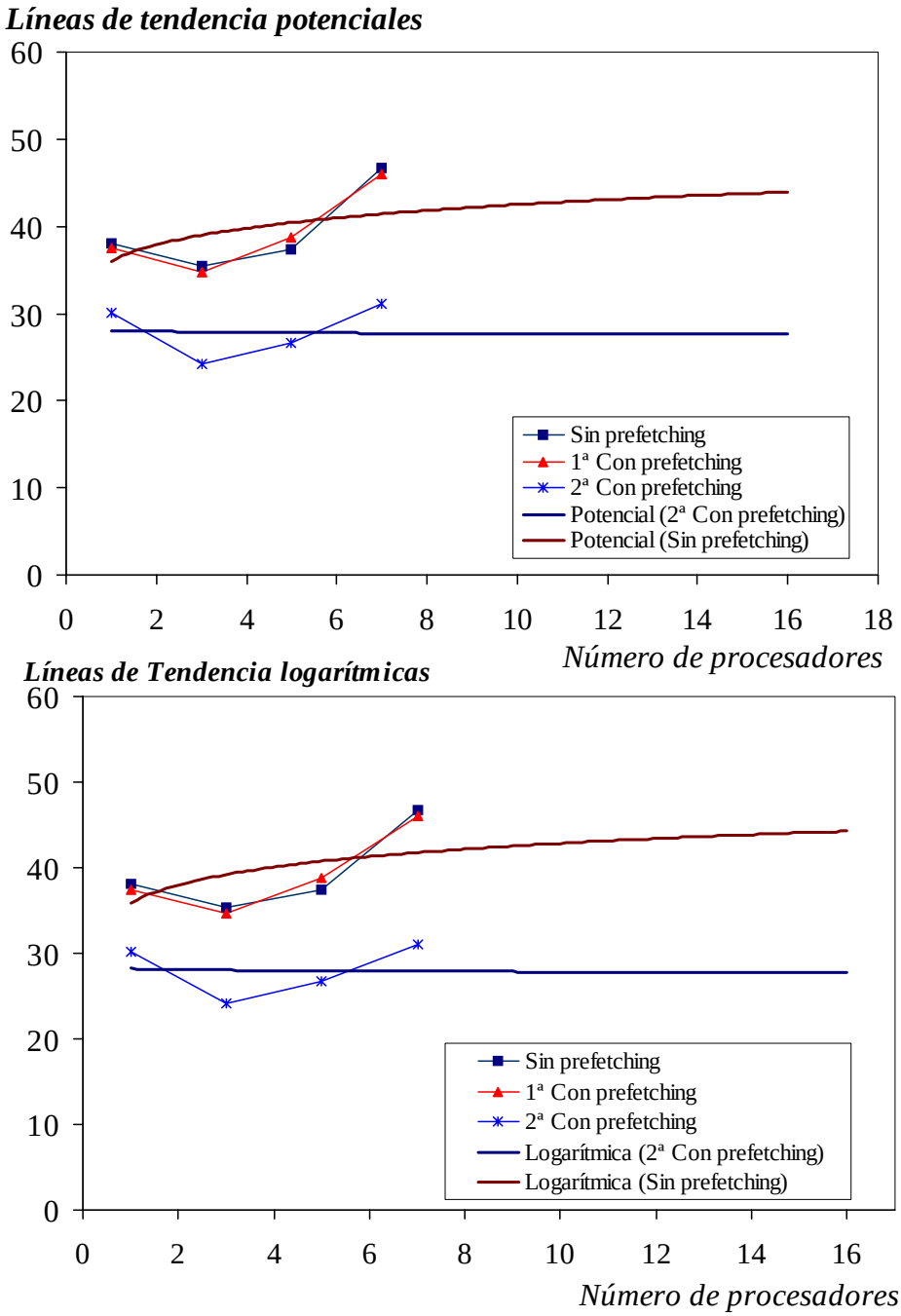


Figura 6.7. Líneas de tendencia potencial y logarítmica

6.3.6. Ganancia obtenida

Vamos a representar también la ganancia conseguida con la utilización de la técnica de prefetching. Para ver cómo afecta la mejora en el tiempo de lectura con el prefetching implementado, voy a utilizar la Ley de Amdahl [194], que me va a dar la ganancia que se puede obtener con la mejora de una porción de la ejecución.

Representaremos la ganancia conseguida al utilizar prefetching para el tiempo de lectura y para el tiempo de ejecución total de la aplicación. Los datos aparecen en la siguiente tabla

Tabla 6.7. Ganancia obtenida (patrón 2)

#PROC	*TIEMPO TOTAL*	*TIEMPO DE LECTURA*
1	0,40%	24,00%
3	2,52%	34,25%
5	5,31%	33,45%
7	6,66%	54,74%

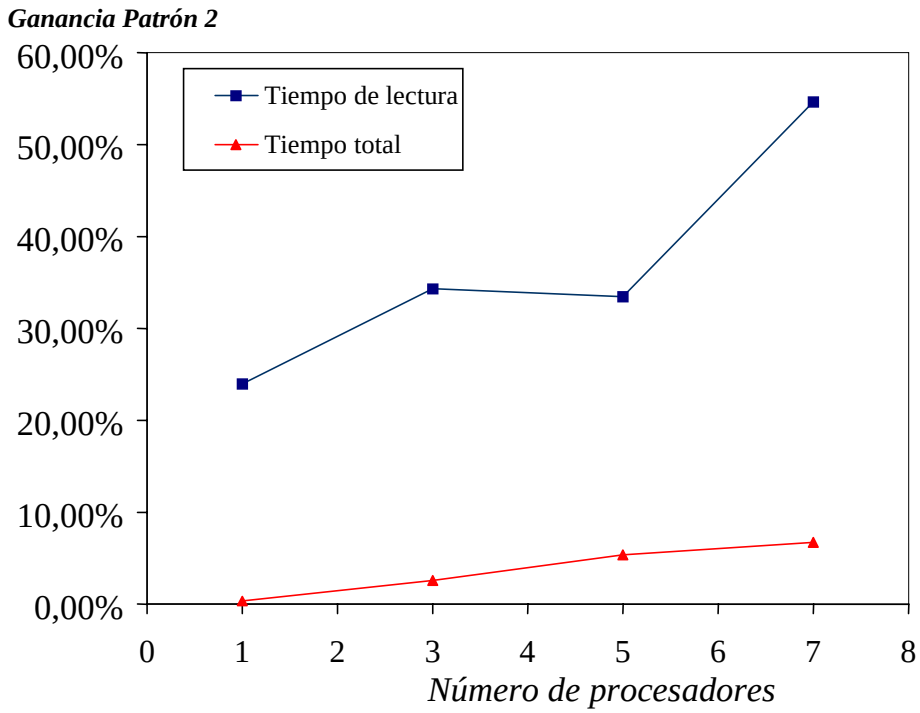


Figura 6.8. Ganancia obtenida para el tiempo total y para el tiempo de lectura

La ganancia para el tiempo de lectura crece bastante con el aumento en el número de procesadores. La ganancia para el tiempo total crece también, pero de forma más suave.

Examinemos a fondo el caso de 7 procesadores. Vemos que, para este caso, la ganancia es de un 54.74% para el tiempo de lectura y de un 6.66%para el tiempo total. Puede que nos parezca pequeña para el tiempo total, pero es porque la fracción que mejoramos (el tiempo de lectura) es sólo un 12% del total de ejecución, para esta aplicación.

Si el porcentaje del tiempo de lectura sobre el tiempo total fuera mayor, la ganancia para el tiempo total también sería mucho mayor. Hemos mejorado una fracción del tiempo total que es el tiempo de lectura, y en función de lo grande que sea esa fracción, la ganancia será más o menos significativa.

Por este motivo, hemos querido calcular, cuál sería la ganancia en aquellas aplicaciones en las que la fracción de tiempo dedicada a la lectura (fracción mejorada) sea mayor. Lo mostramos a continuación:

Tabla 6.8. Ganancias para el tiempo total en función de la fracción del tiempo de lectura, para 7 procesadores

% Tiempo de lectura	*%Ganancia*
0	0,00%
5,00%	2,75%
10,00%	5,51%
12,09% MPEG-2	**6,66%**
15,00%	8,26%
20,00%	11,02%
25,00%	13,77%
30,00%	16,53%
35,00%	19,28%
40,00%	22,03%
45,00%	24,79%
50,00%	27,54%

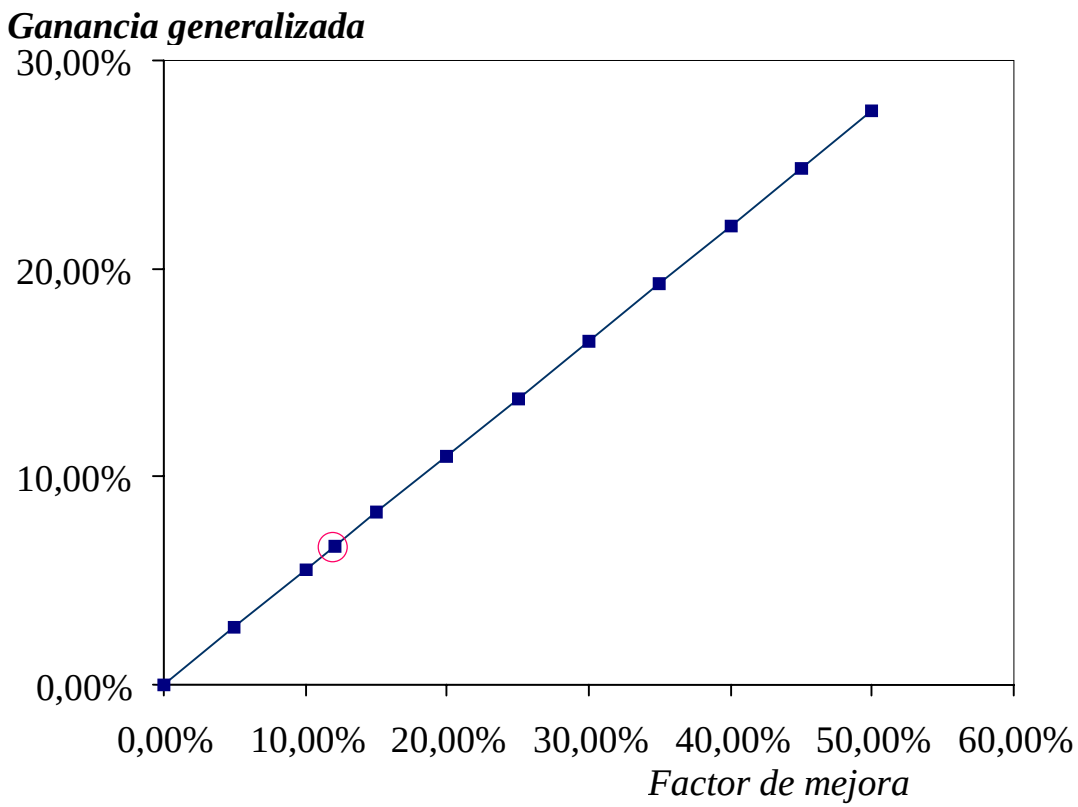


Figura 6.9. Ganancias generalizadas para otras aplicaciones (patrón2)

Vemos como, para otras aplicaciones, en las que el porcentaje del tiempo de lectura sea mayor, también será mayor la ganancia conseguida.

Representamos sólo hasta un factor de mejora del 50%, ya que para casos en los que sea superior, nuestra técnica de prefetching no será de utilidad, ya que su fundamento es adelantar los datos mientras se está procesando la información, por lo tanto si el tiempo de lectura superase al de procesamiento ya no sería ventajoso, pues no daría tiempo a adelantar dicha información.

Con estos resultados terminaríamos la exposición de resultados para este patrón de carga, donde todos los procesadores leen cada una de las imágenes de la secuencia de forma completa. Vamos a ver los resultados con otro patrón de carga significativo, el patrón 1, en el que cada cliente lee un trozo de datos de cada una de las imágenes. Después de completar este estudio realizaremos un análisis conjunto y enumeraremos las conclusiones oportunas sobre los resultados obtenidos.

6.4. RESULTADOS OBTENIDOS CON PREFETCHING PARCIAL

Para este patrón de carga, en el que los clientes sólo leen un trozo de cada una de las imágenes, el tiempo dedicado a la lectura de datos será menos que para el caso anterior, excepto para el caso de 1 procesador.

Vamos a plantear los resultados con el prefetching definitivo, la misma implemetación comentada anteriormente. En el kernel no tenemos que cambiar nada, lo que ocurre ahora es que los valores de los offsets almacenados en cada uno de los nodos del grafo no contendrán los valores de inicio y fin de fichero, sino dos valores cualquiera de inicio y fin de lectura en el fichero, pero todo sigue exactamente igual.

Utilizamos el patrón 2 en todo la experimentación inicial porque era el que peor resultados ofrecía con el cliente NFS tradicional. Pero la implementación definitiva ya quería perseguir la generalidad y la implementamos con la idea central de poder adelantar cualquier trozo de datos.

6.4.1. Comparación de resultados

Vamos a exponer los resultados obtenidos en el kernel sin prefetching, en el kernel con prefetching, y en la implementación del prefetching en la propia aplicación. Para el prefetching en la aplicación realizamos dos versiones, una que solo adelanta el trozo correspondiente del fichero siguiente, y una segunda, que adelanta el fichero completo. Como el tiempo de procesamiento iba a ser mayor que el de lectura, queríamos comprobar si iban a existir diferencias por adelantar un trozo o por adelantarlo todo.

Como en el análisis anterior, primero vamos a comparar los tiempos obtenidos con 7 procesadores y después, representaremos todos los tiempos variando el número de clientes NFS.

La mecánica de las pruebas ha sido la misma que con el patrón anterior. Entre ejecuciones se limpia la cache tanto de los clientes como de los servidores. Se realizan 10 ejecuciones y se presenta la media aritmética de todas ellas. En las 10 ejecuciones realizadas en el kernel sin prefetching los tiempos obtenidos van a ser muy similares, pero, en los resultados con el kernel con prefetching se debe diferenciar claramente la primera ejecución del resto. En la primera ejecución se creará el grafo de accesos, y en las siguientes se utilizará para descubrir el fichero siguiente que debe adelantar el proceso hilo. Vamos a ver cuáles han sido los tiempos obtenidos.

Tabla 6.9. Tiempos de lectura con y sin prefetching en el kernel

SIN PREFETCHING	*1ª EJEC*	*2ª EJEC*
T. LECTURA	***22,8961***	***22,4581***
CON PREFETCHING	*1ª EJEC*	*2ª EJEC*
T. LECTURA	***22,6173***	*12,1935*

Hay una importante reducción, como se puede ver en la tabla anterior. La reducción obtenida con el prefetching en el kernel es de un 47%, lo que pone de manifiesto la importancia de esta técnica y la necesidad de mejoras en el cliente NFS.

Si lo representamos gráficamente, podemos apreciar esta reducción de forma más clara.

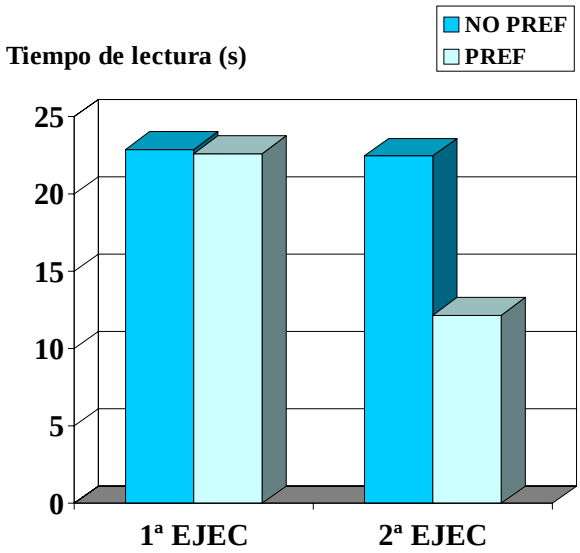


Figura 6.10. Tiempos de lectura con y sin prefetching en el kernel

Veamos también los resultados obtenidos con la implementación del prefetching en la aplicación.

Tabla 6.10. Tiempos de lectura para el patrón 1 con 7 procesadores y 400 imágenes

SIN PREFETCHING	22,8961
PREFETCHING EN EL KERNEL	12,1935
PREFETCHING COMPLETO EN LA APLICACIÓN	12,6804
PREFETCHING PARCIAL EN LA APLICACIÓN	12,6791

Con las lecturas adelantadas es posible reducir, casi a la mitad, el tiempo de lectura. También es importante resaltar la igualdad de resultados obtenidos con el prefetching en el kernel y con el prefetching en la aplicación, y de esta última técnica, la igualdad de resultados adelantando todo el fichero completo (prefetching completo) o sólo la parte que se necesita (prefetching parcial).

6.4.2. Resultados con distinto número de clientes NFS

Vamos a ver qué ocurre para el resto de casos paralelos, cuando interviene distinto número de procesadores. En primer lugar mostramos la tabla con los tiempos obtenidos y a continuación representamos gráficamente estos resultados.

Tabla 6.11. Tiempos de lectura con distinto número de procesadores (patrón 1)

# PROC	SIN *PREFETCHING*	CON *PREFETCHING*
1	38,107	30,7039
3	20,5467	14,6167
5	17,12	11,956
7	22,8961	12,1935

La reducción es bastante considerable, sobre todo en los casos paralelos. Además hay que resaltar que el aumento en el número de procesadores no conlleva el incremento en los tiempos de lectura que se puede ver en el caso sin prefetching. Al aumentar de 5 a 7 procesadores empiezan los problemas de aumento en los tiempos de lectura, cuando en este patrón la cantidad de datos a leer es menor con el aumento en el número de procesadores. Para el caso sin prefethcing se aprecia un mínimo aumento que habrá que estudiar más en detalle con el análisis de regresión correspondiente.

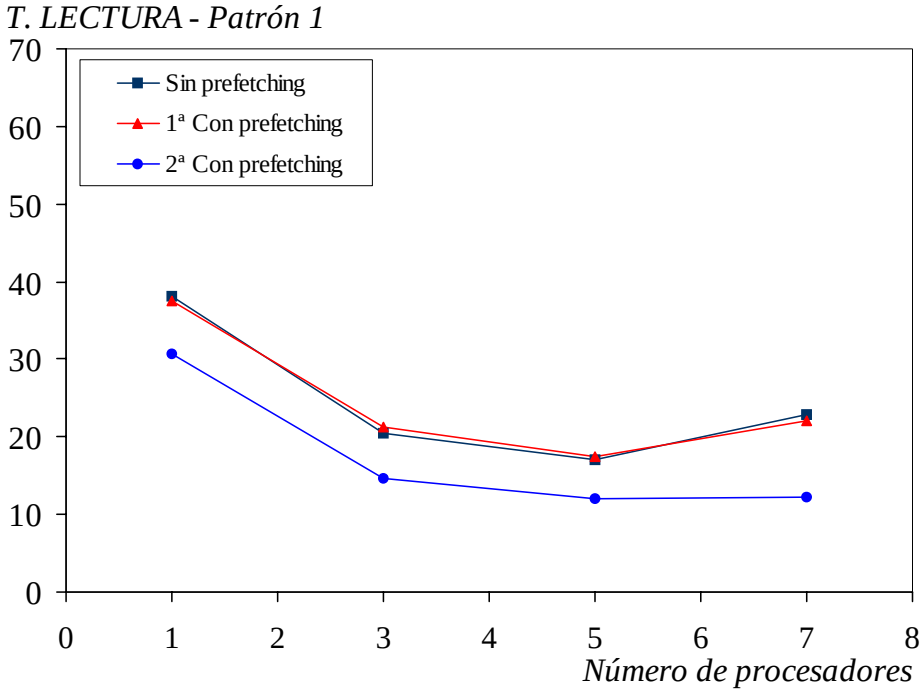


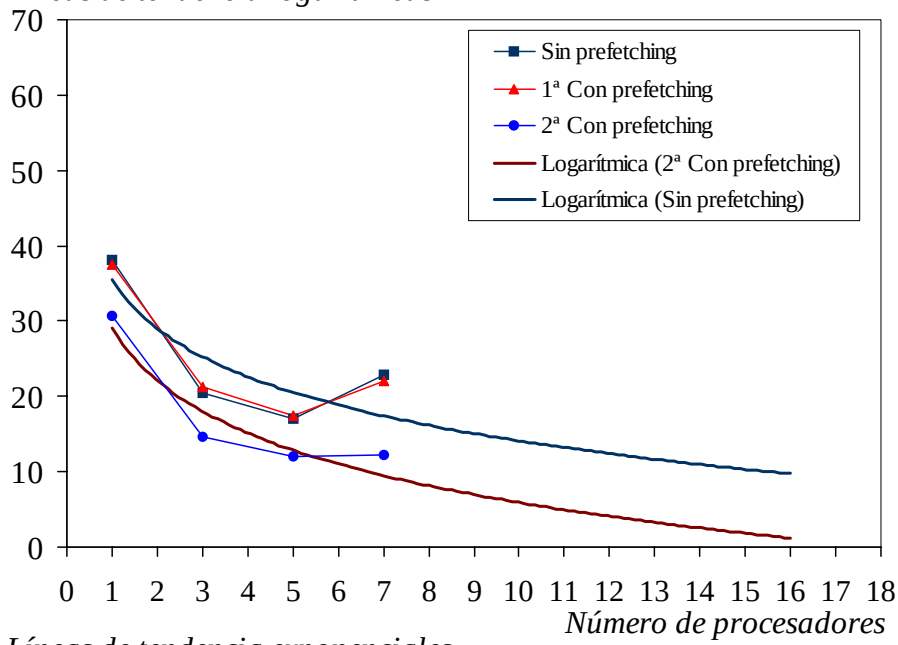
Figura 6.11. Tiempos de lectura para distinto número de procesadores

Hemos representado los tiempos que se obtienen sin prefetching, los tiempos obtenidos con el kernel con prefetching, en la primera ejecución, para comprobar que no hay efectos negativos por la ejecución del hilo, y en la segunda ejecución, donde ya se saca ventaja de la técnica implementada.

6.4.3. Cálculo de tendencias

Vamos a representar también lo que ocurriría si tuviésemos un cluster de 16 máquinas. Vamos a mostrar las líneas de tendencia hacia delante, para analizar los tiempos que obtendríamos si tuviésemos un mayor número de clientes en nuestro cluster. Representaremos las líneas de tendencia logarítmica y exponencial.

Líneas de tendencia logarítmicas



Líneas de tendencia exponenciales

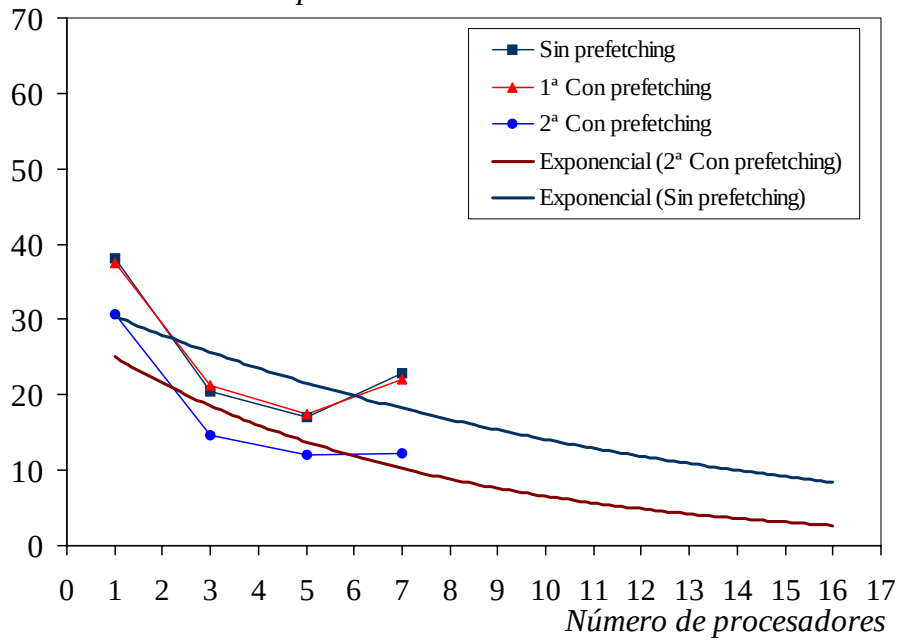


Figura 6.12. Líneas de tendencia logarítmica y exponencial para el patrón 1

Para este patrón se observa que, para ambos casos (sin prefetching y con prefetching) la línea de tendencia decrece, con el aumento en el número de procesadores. Para ver claramente mayores diferencias entre ambas, vamos ver que ocurriría en un cluster con 32 nodos.

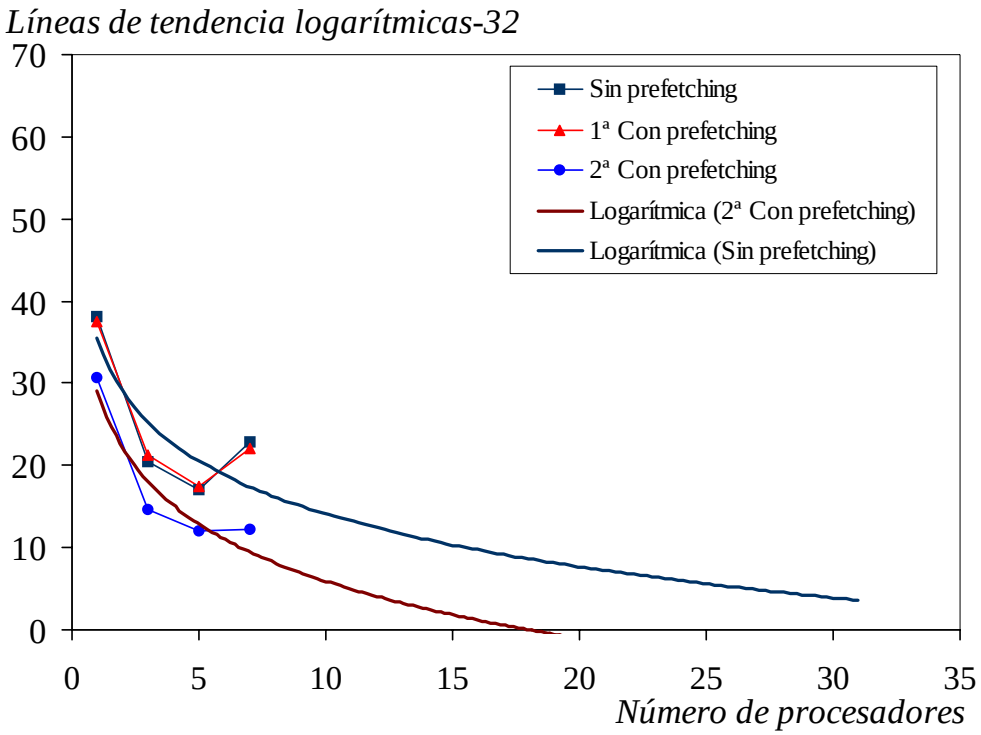


Figura 6.13. Líneas de tendencia logarítmicas para un cluster de 32 nodos

Como se puede observar, la previsión de tiempos para los clientes NFS predictivos es mucho menor que la previsión de tiempos que se obtendrían sin aplicar técnicas de prefetching. Por lo que esta técnica va a resultar muy ventajosa incluso si no se adelantan ficheros completos. Veamos ahora la ganancia conseguida con este patrón de lectura.

6.4.4. Ganancia conseguida

Representemos también los porcentajes de ganancia conseguida, tanto para los tiempos de lectura como para el tiempo total, con distinto número de procesadores.

Tabla 6.12. Ganancias conseguidas (patrón 1)

#PROC	*TIEMPO TOTAL*	*TIEMPO DE LECTURA*
1	0,40%	24,11%
3	1,03%	40,57%
5	1,40%	43,19%
7	6,58%	87,77%

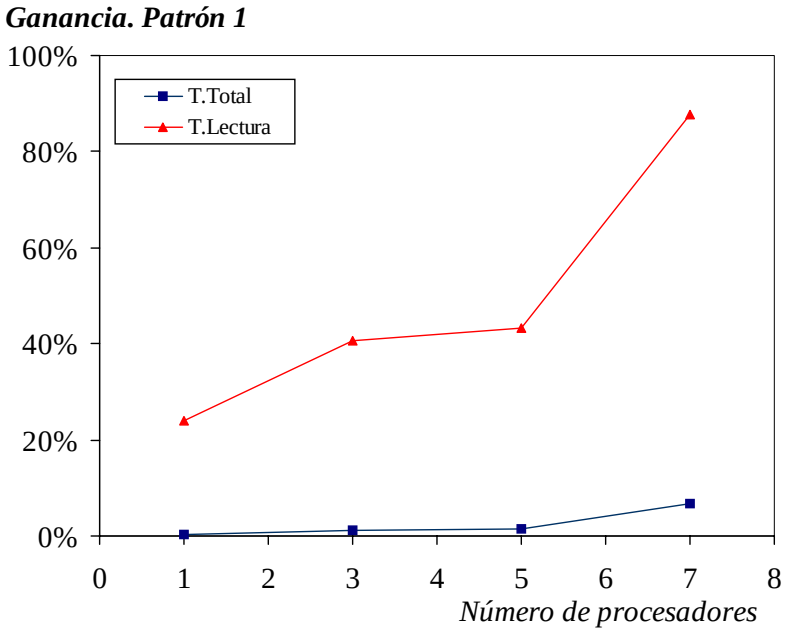


Figura 6.14. Ganancia para el tiempo total y el tiempo de lectura (patrón 1)

Hay una sorprendente ganancia en los tiempos de lectura para 7 procesadores, como ya comentamos al principio de este estudio la reducción era casi del 50% y esto se ve reflejado en esta gráfica. El motivo del porcentaje tan reducido para la ganancia del tiempo total ya lo hemos comentado. Ahora con este patrón, la fracción de tiempo que se mejora con el prefetching es pequeña en relación con el tiempo de procesamiento y de ahí que no se obtenga una ganancia más considerable para el tiempo total.

Podemos hacer un estudio más detallado para 7 procesadores y calcular la ganancia que se podría conseguir para otros porcentajes del factor de mejora, de la fracción del tiempo de lectura en relación al tiempo total. Se tiene una ganancia de un 6,58% para el tiempo total, y la fracción que se mejora es de un 8%. Por lo tanto podemos elaborar una tabla similar a la realizada para el patrón anterior.

Tabla 6.13. Ganancias para otros porcentajes de fracción a mejorar

% Tiempo de lectura	*%Ganancia*
0	0,00%
5,00%	4,11%
8,00% MPEG	**6,58%**
10,00%	8,23%
15,00%	12,34%
20,00%	16,45%
25,00%	20,56%
30,00%	24,68%
35,00%	28,79%
40,00%	32,90%
45,00%	37,01%
50,00%	41,13%

Con este patrón se podía llegar hasta una ganancia del 41% para el tiempo total de ejecución, en aquellas aplicaciones cuya fracción dedicada a la lectura de datos sea del 50%. Existe una clara diferencia con el patrón anterior, donde veíamos que la ganancia correspondiente para el caso en el que la fracción de mejora fuera del 50%, era de un 27,54%.

Gráficamente tendríamos lo siguiente:

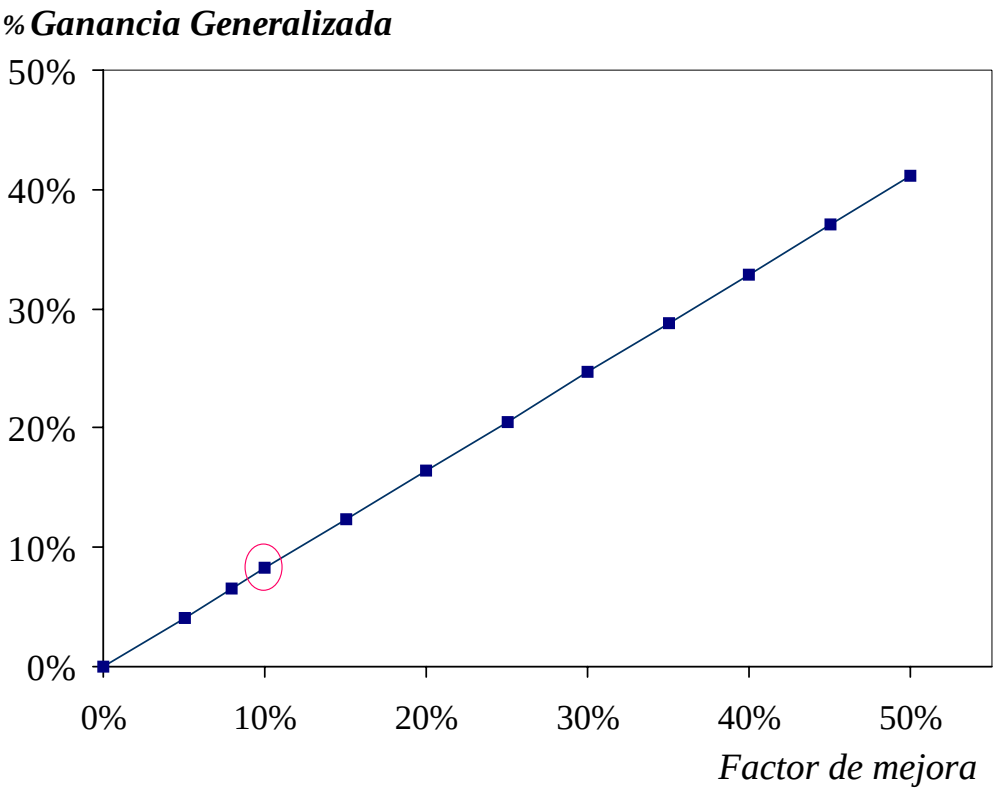


Figura 6.15. Ganancias generalizadas para otras aplicaciones (patrón 1)

6.5. CONCLUSIONES DEL ESTUDIO

La implementación de una técnica de prefetching a través de ficheros supone una importante mejora del cliente NFS. Esta es la mejor forma de empezar un apartado de conclusiones sobre los experimentos finales realizados. El código NFS estaba esperando este tipo de novedades. Las aplicaciones evolucionan, las tecnologías de interconexión también, y por tanto, el software y el middleware entre ambos mundos, también. Debe realizarse un gran esfuerzo de implementación para actualizar o para acercar un estándar universal como es NFS, a las últimas novedades en el campo de la computación paralela. La versión 4 de NFS va a ser una revolución con respecto a las dos versiones anteriores, pero no cubre todos los aspectos posibles de mejora que se podrían plantear. De ahí que nos planteásemos aportar la idea del prefetching a través de ficheros a este sistema de ficheros.

Los resultados saltan a la vista. Con una versión muy trabajada, pero todavía mejorable, se han conseguido importantes reducciones en los tiempos de lectura. Y esto es algo que recompensa toda una etapa intensiva de trabajo con el código de NFS.

Pasamos bastante tiempo verdaderamente obsesionados por conseguir mayores reducciones en los tiempos de lectura, sin pararnos ni siquiera a pensar en lo que ya habíamos conseguido. Y es que, en realidad, se puede seguir consiguiendo mejorar ciertos aspectos del funcionamiento del cliente NFS.

Estamos trabajando en un módulo inmerso en el núcleo del sistema operativo, y por tanto, éste marcará el funcionamiento de todo el sistema. Y decimos esto, porque hemos trabajado incorporando mejoras en el código de NFS, pensando en que se verían reflejadas inmediatamente como si de algo único e independiente se tratara, y eso no es así. El sistema operativo sigue gestionándolo todo, y en particular, la memoria, las páginas de memoria. Además, no es nuestro objetivo cambiar esto. Tal como hemos explicado la técnica de prefetching, el funcionamiento esperado sería el siguiente:

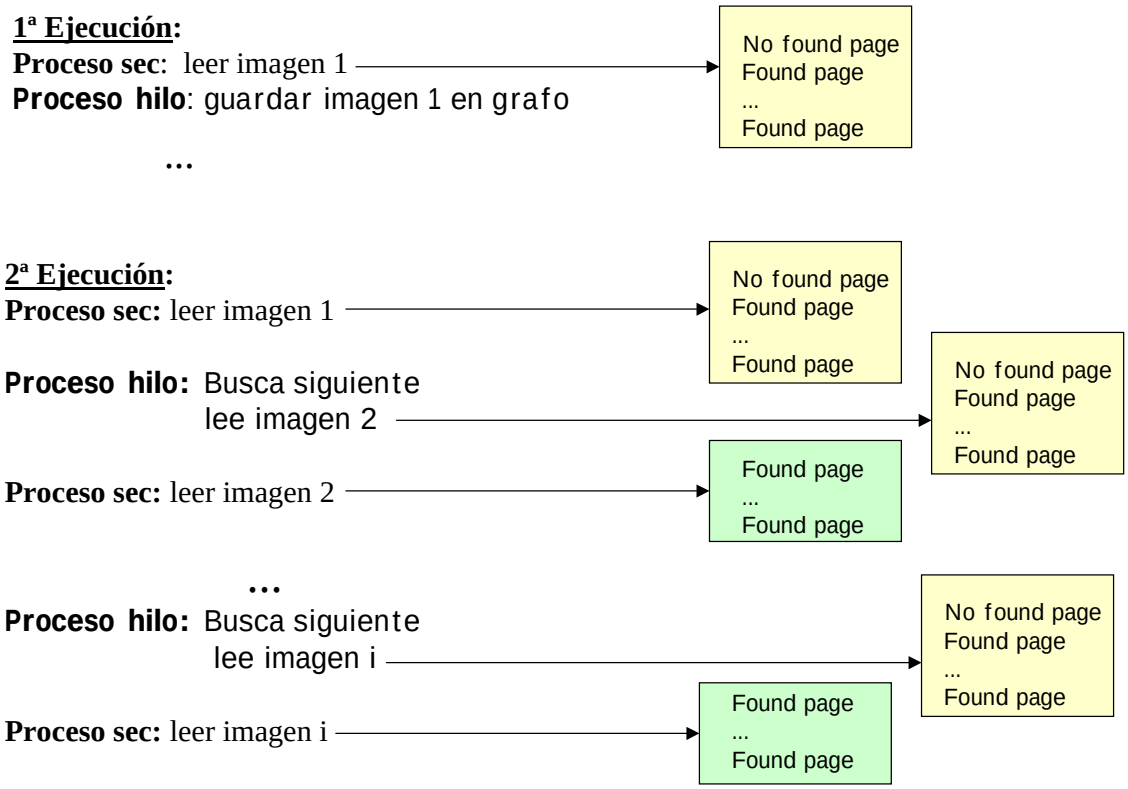


Figura 6.16. Funcionamiento esperado en el cliente NFS

Los bloques amarillos reflejan la situación general si examinamos en detalle los fallos de página que se producen. Es la forma general en la que el cliente lee las páginas de memoria con los datos que necesita. Solo hay un fallo de páginas al principio, porque después la función de readahead las habrá pedido al servidor, para adelantarlas a la cache para que el cliente las lea cuando quiera.

Esto es lo que le ocurre también al proceso hilo, que traerá todas las páginas a la cache, así cuando un proceso secuencial quiera leer el mismo fichero que el proceso hilo ha leído antes, se debería encontrar todos los datos en la cache y no se produciría ninguna petición al servidor NFS, que es lo que se muestra en los cuadros verdes.

Pero, realmente lo que ocurre realmente no es esto, sino lo que aparece en la siguiente figura*:*

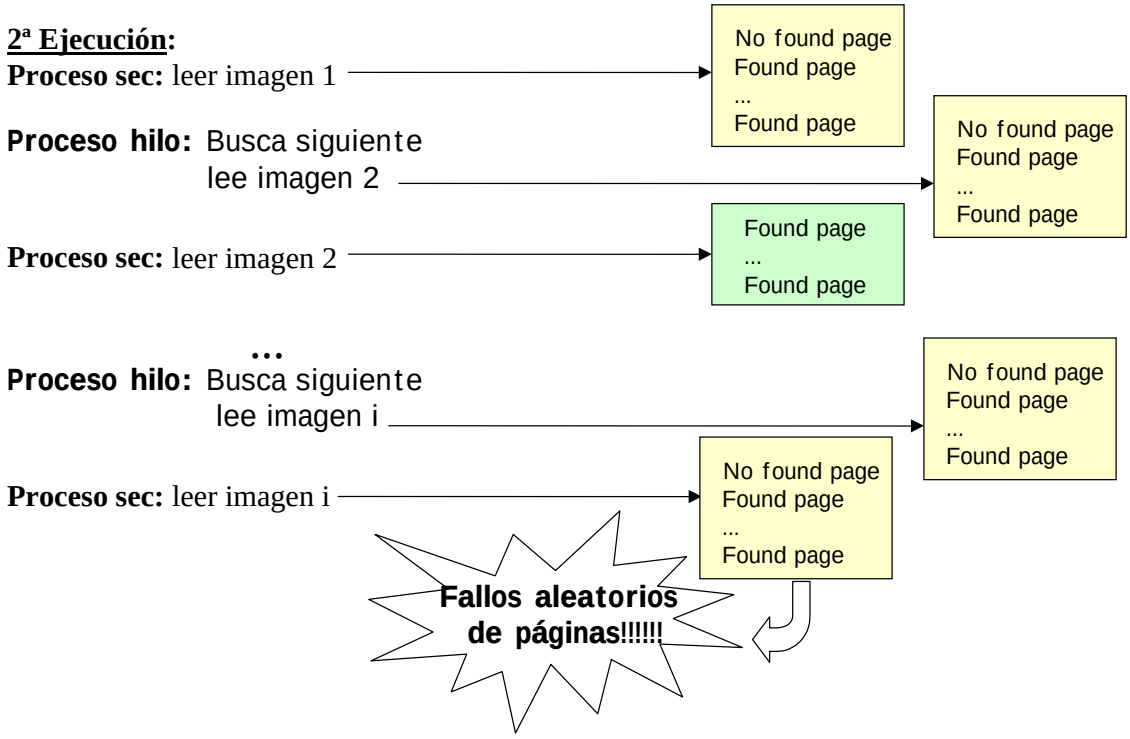


Figura 6.17. Funcionamiento real en el cliente NFS

Como se puede ver en la figura anterior, se van a producir fallos aleatorios de páginas que van a provocar que el proceso secuencial pida de nuevo los datos al servidor. Este ha sido un tema estudiado en detalle, ya que si hubiésemos observado un patrón similar de fallos de páginas, quizás nos habríamos planteado la forma de solucionar estas faltas, pero es algo que ocurre de forma aleatoria y en los dos patrones de carga estudiados. El que el proceso hilo lea un fichero y traiga los datos a la cache del cliente no va a ser garantía de que esos datos vayan a estar ahí cuando los pida el proceso secuencial. El kernel tiene su propia política de gestión de memoria y reemplazará las páginas que crea oportuno. No hemos considerado conveniente exponer en detalle las políticas de reemplazo de páginas del kernel, ni las trazas detalladas observadas, porque habría muchos casos que comentar, y alargaríamos el estudio innecesariamente. Esta es una de las tantas partes invisibles de la tesis, es decir, uno de tanto estudios que hemos tenido que realizar para entender el funcionamiento del kernel, siempre tan complejo y profundo para programadores de no tan bajo nivel. Así que damos por hecho que esta es la causa de no haber obtenido, tal y como está planteada ahora mismo la técnica de prefetching, más reducciones en los tiempos de lectura.

Por último, volver a insistir en la forma en la que se han planteado los resultados en este capítulo. Nuestro objetivo era plantearlos de la forma más clara posible, sin enmarañar demasiado las gráficas y las tablas. Esperamos haberlo conseguido. Para todos las pruebas se han realizado 10 ejecuciones, entre las cuales se ha limpiado las caches de los clientes y del servidor, salvo en algunos casos comentados donde no se

han limpiado. Estas 10 ejecuciones quieren representar diez accesos consecutivos, pero independientes, a los ficheros. Para el cliente NFS tradicional, sin prefetching, se van a obtener más o menos los mismos resultados en las diez ejecuciones, por lo que se presenta la media de las diez. En el cliente con prefetching va a existir una diferencia muy importante entre los resultados de la primera ejecución y las siguientes. Los resultados de la primera ejecución serán similares a los obtenidos sin prefetching, ya que la forma de trabajar del proceso secuencial va a ser la misma, no se va a encontrar ninguna imagen adelantada en su cache y las va a tener que pedir todas al servidor. El proceso hilo estará construyendo el grafo de accesos en esta primera ejecución. En las siguientes ejecuciones los resultados van a ser también muy similares entre ellos. Para ver este cambio significativo hemos enfrentado, en la mayoría de las ocasiones, los resultados obtenidos en la primera ejecución y en la segunda, que es donde se aprecian los efectos de la técnica de prefetching. Por supuesto, después de repetir varias veces cada situación de primera y segunda ejecución. También se muestran los intervalos de confianza que reflejan los valores máximos y mínimos obtenidos en estas 9 ejecuciones siguientes, las líneas de tendencia para ver que ocurriría si tuviésemos más procesadores y la ganancia obtenida con esta técnica de prefetching.

En las propuestas de trabajos futuros presentadas en el capítulo siguiente volveremos a este tema y a otros muy interesantes relacionados con las mejoras de nuestro trabajo en el cliente NFS.

CAPÍTULO 7

CONCLUSIONES Y TRABAJOS FUTUROS

Tras exponer los capítulos centrales de la memoria, es el momento de presentar en este, las conclusiones principales a las que llegamos después de toda una etapa de trabajo. Asimismo, presentaremos las distintas contribuciones a congresos a las que ha dado lugar este trabajo, y terminaremos enunciando posibles trabajos futuros que podríamos realizar para seguir en esta misma línea de investigación iniciada.

7.1. CONCLUSIONES Y APORTACIONES

En el capítulo uno de introducción planteamos los objetivos principales de este trabajo: por un lado analizar el comportamiento de los clusters de computadoras soportando procesamiento paralelo de aplicaciones multimedia, y, por otro, tras descubrir algunos importantes problemas en la entrada/salida, mejorar el cliente NFS, para acelerar el acceso a la información multimedia almacenada físicamente en el servidor NFS. El planteamiento de estos objetivos nos ha llevado a la elaboración de los capítulos siguientes, en los que abordamos los problemas principales encontrados y planteamos una solución real y concreta.

El capítulo dos ha sido un repaso por el mundo de los clusters de computadoras, y, más detalladamente, por ciertos aspectos que despiertan en nosotros un mayor interés, como es el de las aplicaciones. Aquí se presenta la aplicación que posteriormente utilizaremos como generadora de tráfico, el codificador de vídeo MPEG-2.

El capítulo tres nos ha servido para explicar los aspectos fundamentales del cluster de computadoras con el que hemos trabajado. Hemos expuesto los principales aspectos software y hardware. Entre ellos, el codificador de vídeo concreto que hemos utilizado, el codificador paralelo MPEG-2 que nosotros mismos hemos implementado

en la Universidad de Ciencia y Tecnología de Hong-Kong. Esta herramienta, totalmente software, nos ha permitido realizar una gran cantidad de experimentos, y nos ha permitido utilizar múltiples versiones distintas del mismo, explotando paralelismo temporal y espacial. Todos los resultados de esta memoria utilizan el paralelismo espacial, ya que es mucho más realista, no supone tener la secuencia completa para empezar a repartir los datos, sino que se divide cada una de las imágenes entre los procesadores que vayan a participar en la codificación. En esta versión espacial, también se han implementado distintos patrones de carga, distintas formas de leer, con el objetivo de acercarnos a las distintas formas de leer que pueden plantearse en las distintas aplicaciones multimedia. Incluso, al final, también hemos implementado una versión de codificador MPEG-2 predictivo, para probar los resultados de realizar prefetching en la propia aplicación. Esta es una aplicación real que va a ejecutarse sobre una plataforma real de trabajo, lo que, ha tenido sus ventajas e inconvenientes. Todo son ventajas desde el punto de vista investigador, pero en más de una ocasión nos hemos visto totalmente abatidos ante problemas que nos parecían no tener solución. Por lo tanto, en esta primera parte del trabajo aportamos una aplicación con múltiples implementaciones que han dado pie a construir un intenso banco de pruebas. También nos han ayudado a sacar conclusiones acerca de la comunicación en los clusters. Nos sirvió para descartar BIP y LAM, y elegir MPICH como librería de paso de mensajes ideal. Nos sirvió para comprobar que hay que evitar al máximo la comunicación de datos entre procesadores, ya que esto iba a retrasar el proceso, por lo que el reparto de datos es totalmente independiente para cada procesador. Cada cliente leerá todos los datos que necesita del servidor NFS y no tendrá que enviar porciones de los mismos a clientes vecinos del cluster.

En este capítulo, también, exponemos los fundamentos de NFS. Lo primero decir, que tenemos mucha más información y que hemos hecho un gran esfuerzo por resumir estos apartados, pero de todas formas hemos creído necesario exponer con claridad y, hasta un nivel de detalle determinado, ciertos aspectos importantes a los que se vuelve más adelante cuando hablamos de la nueva implementación propuesta. Exponemos otros sistemas de ficheros que se podrían haber utilizado, pero ninguno tiene la madurez y la solidez de NFS.

En el capítulo cuatro, presentamos un resumen de los primeros trabajos de evaluación realizados, orientados a detectar los problemas existentes en la entrada/salida.

- Realizamos un primer ajuste de los parámetros NFS como rsize, wsize y timeo. De este primer ajuste podemos concluir que, en sistemas con un solo cliente NFS pidiendo datos al servidor, será conveniente utilizar valores de rsize y wsize grandes. Pero cuando existe concurrencia en el acceso a la información, es decir cuando varios clientes están accediendo a la misma información, el peor de los casos se presenta, justamente, utilizando tamaños de grandes de rsize y wsize. Con respecto al parámetro timeo hemos podido comprobar que el valor por defecto que se le asigna queda totalmente obsoleto para las nuevas redes de alta velocidad donde 0,7 segundos es muchísimo tiempo. Se podrían realizar interesantes modificaciones con este valor como plantearemos más adelante. De todo este estudio, elegimos como

valores de rsize y wsize para posteriores experimentos, precisamente los que peores resultados ofrecían. Así fijamos rsize y wsize a cuatro kilobytes.

- Nos planteamos, dentro de nuestras posibilidades, distintos cambios en la topología de la red, para comprobar la existencia de problemas en los propios conmutadores utilizados. De este estudio sacamos la conclusión de que la red no era la causa del problema detectado, que implicaba un aumento importante en los tiempos de lectura con el aumento en el número de procesadores.

- También probamos distintos patrones de carga de la información multimedia. En todos los casos el algoritmo MPEG-2 era el mismo: leer datos, codificarlos, y escribir los datos codificados en el servidor. El reparto de la imagen era siempre igual. Si participan varios procesadores en la codificación, a cada uno le corresponde una franja. Lo que se variaba era el trozo de datos que tiene que leer cada cliente: justo el trozo que va a codificar (lectura no solapada de datos), el trozo que va a codificar más un trozo necesario para codificar los píxeles de los bordes (lectura solapada de datos), o todo el fichero completo. De esta comparación de casos observamos que los peores resultados aparecen cuando los clientes tienen que leer completamente cada uno de los ficheros de la secuencia.

- Una vez realizadas estas pruebas, en la que se representaban los valores de tiempos medios de todos los procesadores, pasamos a realizar un estudio detallado de los valores de tiempos ofrecido por cada procesador, con la intención de descubrir a qué se debía el incremento de tiempos. Aquí descubrimos los efectos de la cache del servidor que provocaba diferencias importantes en los tiempos individuales de los procesadores, y descubrimos también el read-ahead en el cliente NFS, que es una técnica de gran utilidad ya que adelanta datos dentro del mismo fichero, pero que se detecta tarde, y a veces no se llega a utilizar si lo que se quiere es leer un trozo aleatorio de datos dentro de un fichero. Pudimos comprobar la potencialidad de las técnicas de caching y prefetching, pero también, nos empezamos a plantear cómo sacarles más partido y utilizarlas de forma más eficiente.

La conclusión más importante de este capítulo tres es la necesidad de mejorar el código NFS para optimizar las lecturas, ya que existen muchos trabajos y discusiones acerca de cómo realizar la escritura de datos, y de hecho ha sido uno de los aspectos más interesantes en el cambio de la versión 2 a la 3, pero no se ha hecho nada para mejorar la operación de lectura de datos. Por lo tanto, aquí empezamos a planteárnoslo como un objetivo posible a conseguir.

En el capítulo cinco exponemos los planteamientos de las mejoras que vamos a realizar. Las primeras ideas, su evolución y el planteamiento final. Empezamos con el objetivo de adelantar datos a través de ficheros, ya que veíamos que la cache del cliente no se utilizaba para nada, al limpiarla entre ejecuciones. Pasamos por un copioso proceso de recopilación de artículos y trabajos relacionados con técnicas de prefetching, y nos encontramos con tal cantidad de información que quedamos sorprendidos por la

cantidad de investigadores inmersos en este interesante tema. La idea del prefetching, como la de los clusters, no es de ahora ni mucho menos, existen trabajos de prefetching desde hace bastantes años, lo que ocurre es que han ido evolucionando ciertas características y principios, aunque los fundamentos de la técnica son los mismos. De todos estos trabajos seleccionamos los más cercanos a nosotros, y la lectura y entendimiento de todos ellos nos llevaron a elaborar una posible técnica de prefetching a través de ficheros para NFS.

Inicialmente decidimos examinar el diseño del servidor. Dedicamos bastante tiempo a esta tarea, pero al final, creímos más oportuno revisar el diseño del cliente NFS.

Decidimos modificar, entonces, el diseño del cliente, tras encontrar, por esos mismos días, algunas publicaciones que aconsejaban esta posibilidad y la planteaban como la más natural. Que sea el propio cliente el que se adelanta sus propios datos. Y aquí es donde vivimos los momentos más intensos, ya que aquí se pudo hacer realidad la idea de un proceso hilo que almacena los primeros accesos a los ficheros en un grafo, que después, en los siguientes, utiliza dicho grafo para saber qué fichero debe leer y cargarlo en la cache del cliente antes de que el proceso secuencial normal los tenga que pedir al servidor. Por lo tanto se presenta el cliente predictivo multihilo, con el que se pretenden mejorar los tiempos de lectura hasta entonces obtenidos. Es muy importante la idea del proceso hilo, ya que sino fuera así sobrecargaríamos en demasía al cliente, y retrasaríamos, aún más, su procesamiento. En todo momento se pretende separar el diseño de la técnica, de la implementación final, pero suponemos que un breve acercamiento al código fuente de NFS puede dar una idea de la complejidad de implementar algo nuevo en el mismo núcleo del sistema operativo.

En el capítulo cinco se exponen los resultados obtenidos con este nuevo cliente NFS, comparándolo siempre con los resultados obtenidos con un cliente tradicional (sin prefetching). Empezamos exponiendo las pruebas realizadas para el patrón de carga en el que todos los clientes leen los ficheros completos y presentamos una importante reducción en los tiempos de lectura para este cliente predictivo. Además, mostramos otras pruebas realizadas para completar aún más el estudio comparativo, prefetching en la aplicación, pruebas con cache sucia, etc. Pero, destacar la importancia de la gráfica 6.3 donde se puede apreciar la reducción obtenida. También realizamos análisis de líneas de tendencias para saber qué ocurriría si tuviésemos más procesadores, y las perspectivas son bastante alentadoras para el cliente predictivo, ya que para el cliente tradicional los resultados se disparan con el aumento en el número de procesadores. Finalizamos el estudio para este patrón con el calculo de la ganancia conseguida con la incorporación de esta técnica en el cliente, y llegamos a valores del 54% para el tiempo de lectura.

Realizamos un estudio similar también para otro patrón de carga (el propio de MPEG-2) donde cada cliente lee solo la porción de datos que necesita dentro de cada fichero. Para este patrón también se observan importantes reducciones en los tiempos de lectura obtenidos, visible en la figura 6.10, y, también, se hacen los correspondientes estudios de tendencias y ganancias obtenidas. Para las líneas de tendencia, se puede

observar que para los resultados con prefetching la reducción va a ser más significativa que en el otro caso. La ganancia también es muy importante en este caso, obteniéndose valores del 87% para el tiempo de lectura. Completándose también el estudio con la realización de la estimación de ganancias para aquellas aplicaciones en las que la fracción del tiempo dedicado a la lectura de datos fuera mayor.

Por lo tanto, estamos muy satisfechos de los resultados obtenidos y dispuestos a mejorar esta técnica de prefetching en el cliente para conseguir mejorar aún más los tiempos finales.

Queremos volver a destacar que para que estos experimentos se hayan podido exponer de forma clara y "simple", hemos tenido que hace muchas otras pruebas en la oscuridad para resolver ciertas lagunas o dudas acerca del propio funcionamiento del sistema con el que estábamos tratando. Realizamos pruebas, por ejemplo, anulando el read-ahead en el cliente, para ver el efecto, y por supuesto fue, un aumento significativo en los tiempos, a parte de otras muchas relacionadas con tomas de tiempos en partes determinadas del kernel.

Quiero volver a resaltar lo ya comentado en las conclusiones del capítulo seis, donde hemos mencionado la realidad de estar inmersos en el núcleo del sistema operativo, y por tanto, de dejarnos guiar por él. Por muchas páginas que adelante el proceso hilo, si después el sistema decide reemplazarlas, el cliente, de nuevo, tendrá que pedirlas al servidor. Como consecuencia del propio sistema operativo, y en lo que se refiere a cómo afecta a la aplicación, se puede decir, que los efectos se pueden entender como aleatorios, y, por lo tanto, no hemos querido ni siquiera intentar modificarlo. Este es el funcionamiento del kernel y estos los resultados con nuestro cliente predictivo. Todo lo demás, lo plantearemos en la siguiente sección de trabajos futuros. Así que en ella expondremos las ideas que rondan nuestras mentes acerca de las posibles modificaciones, mejoras y novedades sobre el trabajo ya realizado.

7.2. PUBLICACIONES RELACIONADAS CON LA TESIS

Durante el desarrollo de este Tesis se han publicado distintas partes del trabajo realizado, en diversos congresos nacionales e internacionales. Además, falta por presentar la última parte de resultados con el cliente NFS predictivo, que se va a empezar a publicar justo cuando terminemos esta memoria. En orden cronológico, las publicaciones presentadas son:

Publicaciones Internacionales:

- T. Olivares, P. Cuenca, F. Quiles, A. Garrido, J. Sanchez and J.Duato, *Interconnection network behavior on a multicomputer in the parallelization of the MPEG coding algorithm. Worm-hole vs Packet-Switching Routing,*

Proceedings de la 4ª Conferencia Internacional HiPC (High Performace Computing), IEEE Computer Society Press, ISBN 0-8186-8067-9), Vol. 1, pp. 48-53, Diciembre de 1997, Bangalore (India).

Este fue uno de los primeros artículos publicados en un congreso internacional. Aquí presentamos, con la utilización de un simulador, cómo sería la codificación MPEG-2 en un sistema multicomputador. Se analiza la idoneidad de la topología del sistema así como la técnica de conmutación más adecuada para la red de interconexión.

- T. Olivares, P. Cuenca, F. Quiles and A. Garrido, *Parallelization of the MPEG coding Algorithm over a Multicomputer. A Proposal to Evaluate its Interconection Network*. Proceedings del IEEE Pacific Rim Conference on Communications, Computers andS ignal Processing (PACRIM´97) (IEEE Computer Society Press) Vol. 1, pp. 113-116, 1997, CANADA.

Este trabajo es una continuación del anterior, donde se van a detallar más ciertos aspectos de la codificación en paralelo del algoritmo MPEG-2

- T.Olivares, F.Quiles, P.Cuenca, L. Orozco y I. Ahmad, *Study of data distribution techniques for the implementation of an MPEG-2 video encoder*, Proceedings de la Conferencia Internacional del IASTED, PDCS'99 (Parallel and Distributed Computing and Systems) , ISBN: 0-88986-275-3, Vol. 1, pp. 537-542, 1999, Cambridge, Massachussets (USA)

Esta es la publicación que resume el trabajo realizado en Hong-Kong, la implementación de distintas técnicas de reparto de datos, tanto para el paralelismo espacial como para el temporal, y la comparación de ambas

- T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Tuning NFS on high speed networks with parallel multimedia applications*, Proceedings de la Conferencia Internacional del IASTED, PDCS'00 (Parallel and Distributed Computing and Systems), ISBN: 0-88986-304-0, Vol. 1, pp. 13- 17, Noviembre 2000, Las Vegas, Nevada (USA).

En esta publicación y las siguientes ya empieza a perfilarse de forma clara el objetivo principal de la tesis. Los pasos previos como ya hemos comentado, el análisis y ajuste de ciertos parámetros NFS, realizados ya sobre un cluster multimedia de alta velocidad.

- T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Performance study of NFS over Myrinet-based clusters for parallel multimedia applications*, Proceedings de la CCECE'01 (Canadian Conference On Electrical And Computer Engineering), IEEE Computer Society Press, Mayo 2001, Toronto (Canadá)

En este artículo se completan los resultados anteriores con la presentación de un análisis de resultados con distintas topologías de red

- F. Pelayo, F. Cuartero, V. Valero, D. Cazorla, T. Olivares, *Specification and Performace of the MPEG-2 Video Encoder by using the Stochastic Process*

Algebra: ROSA, Proceedings del 2001 UK Performance Engineering Worshop (UKPEW'2001), pp 105, Julio 2001, UK.

Este es un ejemplo de la aportación que ha supuesto la implementación del codificador paralelo, como aplicación multidisciplinar para el resto de miembros del departamento

- T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *A network File System evaluation over a high-speed multimedia clustered architecture,* Proceedings del V World Multiconference on Systemics, Cybernetics and Informatics (Invited Session: Multimedia on Clusters of Workstations: Mastering the future), Volume XII, Part II, pp.374-380, 22-25 Julio 2001, Orlando, Florida (USA).

En este trabajo se completa el análisis de NFS con la implementación de diferentes patrones de carga, representativos de distintas aplicaciones multimedia

- T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *The Need of Multicast Predictive NFS Servers for High-Speed Networks used as Parallel Multimedia Platforms,* Proceedings del ICPP'01 Worshop on Scheduling and Resource Management for Cluster Computing, ISBN: 0-7695-1260-7, pp.391-396, IEEE Computer Society Press, Septiembre 2001, Valencia (España)

Aquí se presenta el análisis de los tiempos de lectura por procesador, los efectos de las técnicas de *caching* y *prefetching* en NFS, y se empieza a plantear las primeras soluciones, como podemos ver, orientadas a la mejora del servidor NFS.

Publicaciones Nacionales:

- P. Cuenca y T. Olivares, *MPEG-2: Un Sistema de Compresión de Vídeo para la Televisión Digital*, Ensayos'1996, ISSN: 0214-4824, pp. 195-215, Universidad de Castilla - la Mancha.

Este trabajo resume los primeros estudios con la codificación MPEG-2

- T.Olivares, F.Quiles, P. Cuenca, *Evolución de las arquitecturas paralelas en visión artificial.* Actas de las VI Escuela de verano, ISBN: 84-89492-43-3, pp. 299- 308, 1997, Universidad de Castilla-La Mancha.

Con este trabajo se produjo el acercamiento al mundo del procesamiento paralelo, empezando, claro está, por las arquitecturas más interesantes.

- T.Olivares, P.Cuenca A.Garrido, F.Quiles y J.L.Sanchez, *Codificación en paralelo de vídeo MPEG-2 en un multicomputador. Análisis de la red de interconexión,* Actas de las VIII Jornadas de Paralelismo, 1997, Cáceres.

Este trabajo resume toda una primera etapa de trabajo con un simulador, elaborado en la Politécnica de Valencia, para estudiar los efectos del tráfico MPEG-2 en un multicomputador

- T.Olivares**,** *Codificación de vídeo MPEG-2 sobre una red de estaciones de trabajo*. Actas de las IX Escuela de verano, ISBN: 84-89958-82-3, pp. 139-162, 1999, Universidad de Castilla-La Mancha. Albacete

En esta publicación nos trasladamos de lleno al mundo de los cluster "en vivo", y empezamos a experimentar con el codificador implementado en la Universidad de Hong-Kong

- T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Evaluación de un sistema de ficheros NFS en un cluster de altas prestaciones con aplicaciones multimedia,* Actas de las XI Jornadas de Paralelismo, Septiembre 2000, Granada.

En la presentación de esta publicación en Granadan en las Jornadas de Paralelismo, a las que asistimos anualmente, presenté la primera evaluación del sistema de ficheros NFS sobre un cluster de alta velocidad.

- T. Olivares y otros, *Linux, guía de instalación y administración,* ISBN: 84-481-2891-5, 2001, McGraw-Hill, Madrid.

En este libro he podido plasmar mis conocimientos en sistemas de ficheros, en general, y en los sistemas de ficheros en red, en particular. Ha sido un trabajo conjunto realizado por varios miembros del departamento y del que he aprendido mucho.

7.3. TRABAJOS FUTUROS

Teniendo en cuenta la situación en la que nos encontramos, después de una etapa intensiva de trabajo, nos sorprende la cantidad de cosas interesantes que podríamos realizar como continuación de este trabajo. Queríamos demostrar que el adelantamiento de datos a través de ficheros en el cliente, iba a suponer una reducción significativa en los tiempos de lectura, y eso, ya lo hemos demostrado. Ahora lo que vamos a proponer son tareas puntuales para continuar todo el trabajo realizado, partiendo de la idea general de que el diseño de NFS necesita algunas actualizaciones en la forma en la que se realizan determinadas operaciones y procedimientos, para acercarlo más a la realidad de la alta velocidad.

- Una de las primeras cosas que habría que hacer es el mantenimiento del grafo de accesos, con el que contamos para descubrir el fichero que se debe leer a continuación. No solo nos referimos a la estructura de datos como tal, que durante nuestro trabajo anterior se han implementado distintas posibilidades, sino la gestión del grafo, para los casos en los que agote los requisitos de espacio en memoria. Se podría presentar como una estructura de un tamaño limitado y configurable, en el cual se iría eliminando la información, siguiendo algún criterio como el del tiempo, por ejemplo. Por lo tanto, habría que definir otras políticas de sustitución de la información almacenada en el grafo que pudiesen optimizar el acceso y mantenimiento del mismo.

- Preparar todo el código nuevo para que forme parte de un patche del cliente NFS, que se pueda compilar en el kernel y a partir de ese momento utilizarlo y sacar todas las ventajas del adelantamiento de datos en la cache

- Trasladar todo lo implementado a la nueva versión de NFS, la versión 4, que va a suponer una verdadera revolución en los sistemas de ficheros. Pasar por una etapa previa de conocimiento del código y después, portar nuestra técnica de prefetching a través de ficheros, a esta nueva versión.

- Implementar técnicas de multicast en el envío de la información por parte del servidor, en la nueva versión 4. En esta nueva versión al servidor se le dota de cierto estado, para controlar algo más las peticiones (qué se pide y quién lo pide), de ahí que sería posible pensar en este tipo de técnicas para la versión 4.

- NFS sobre Ipv6. No hay nada hecho acerca de este tema. Se podría trasladar todo lo hecho en IPv6, y aprovechar las ventajas de las mejoras en la pila TCP/IP

Señalar por último que el trabajo presentado en esta Tesis Doctoral, y el que proponemos como trabajo futuro a realizar, se ha desarrollado y se va a desarrollar en el marco de los proyectos CICIYT siguientes:

TIC97-0897-C04-02, Desarrollo de una Red de Estaciones de Trabajo de Altas Prestaciones y Bajo Coste, proyecto coordinado entre la Universidad de Castilla-La Mancha, la Universidad Politécnica de Valencia y la Universidad de Murcia (Junio de 1997 - Diciembre de 2000)

TIC2000-1151-C07-02, Redes de Estaciones de Trabajo de Altas Prestaciones para Aplicaciones Multimedia, proyecto coordinado entre la Universidad de Castilla-La Mancha, la Universidad Politécnica de Valencia, la Universidad de Murcia, la Universidad de Valencia y la Universidad Jaime I (Enero de 2001- Diciembre de 2003)

BIBILIOGRAFÍA

[1] B. Alarcos, **MPEG Vídeo**, Departamento de Automática, Universidad de Alcalá, http://www.aut.alcala.es/~alarcos/docente/at_itt/mpeg/vídeo.htm

[2] F.J. Alfaro, J.L. Sánchez, J.Duato y C.R. Das, *A strategy to computethe InfiniBand arbitration tables*, En Proceedings del Congreso Internacional: Parallel and Distributed Processing Symposium (IPDPS). IEEE Computer Society. Abril 2002.

[3] T.E. Anderson, D.E. Culler, D.A. Patterson y otros, ***A case for NOW (Network of Workstations***, IEEE Micro, Febrero 1995, pp. 54-64.

[4] *Apple Seed, Personal Parallel Computing* http://exodus.physics.ucla.edu/applesed/applesed.html

[5] V. Bhaskaran y K. Konstantinides, **Image and Video Compression Standards. Algorithms and Architectures**, 2ª Edición, Kluwer Academic Publishers, 1997.

[6] U. Black. **Tecnologías emergentes para redes de computadoras**. Prentice Hall, segunda edición, 1997.

[7] R. Buyya, **High Performance Cluster Computing**, Volume 1, Architectures and Systems, Prentice Hall, 1999.

[8] R. Buyya, *The Application of PDC (Parallel and Distributed Computing) and Technique in E-commerce*, PDCAT'2000 Panel, Hong-Kong, Mayo de 2000.

[9] Página web de *Cabot Communications*, http://www.cabot.co.uk/

[10] J.M. Canelada, *Tiendas virtuales en Linux***, Linux Actual, nº 16.**

[11] N. Celandroni y otros, ***MPEG-2 Coded Video Traces Transmitted over a Satellite Link: Scalable and Non-Scalable Solutions in Rain Fading Conditions***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 10, Nº 1, 2000

[12] *Computing Cluster & Cybercafe, Computing Cluster & Classrooms*, Universidad de Chicago, http://intech.uchicago.edu/ccc/crerar.html

[13] T. S. Chua y otros, ***A Replication Strategy for Reducing Wait Time in Video-on-Demand Systems***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 15, Nº 1, 2001

[14] *Clemson Dedicated Cluster, Parallel Computing*, http://www.parl.clemson.edu/cdcpc.html

[15] *Cluster Computing White Paper*, versión 1.01, Julio del 2000, IEEE Computer Society Task Force on Cluster Computing. http://homer.csm.port.ac.uk/other-activities/tfcc/

[16] M. Conti, ***Modeling MPEG Scalable Sources,*** Multimedia Tools and Applications, Kluwer Academic Publishers, Volumen 13, Nº 2, 2001

[17] P.Cuenca Castillo, **Codificación y Transmisión Robusta de Señales de Vídeo MPEG-2 de Caudal Variable sobre Redes de Transmisión Asíncrona ATM**. Tesis Doctoral, Universidad Politécnica de Valencia, Noviembre 1998.

[18] *Davic, Digital Audio Visual Council*, http://www.davic.org/

[19] *Multimedia/Hypermedia Interchange Standards*, http://www.diffuse.org/moving.html

[20] General Instruments Corporation, *DigiCipher™ HDTV System Description*, Agosto 1991.

[21] VITA Standards Organizations, *Myrinet-on VME Protocol Specifications*, Draft Standard VITA 26-199x, Draft 1.1, Agosto 1998.

[22] J. Duato, S. Yalamanchili y L. Ni, **Interconnection Networks, an Engineering aproach**, IEEE Computer Society Press, 1997.

[23] ITU-T Recommendation BT.601, *Encoding Parameters of Digital Television for Studios*, 1982

[24] L.Fernández, J.M. García. ***Una aproximación a las redes de altas prestaciones***, X Jornadas de paralelismo, 13-15 septiembre 1999, pp.225-233.

[25] Documentos del MPI Forum. http://www.mpi-forum.org/docs/docs.html

[26] D. García y W. Watson, ***ServerNet II***, Proceedings CANPC Workshop, Lecture Notes in Computer Science 1362, 1998.

[27] E. Gavilan, **MPEG-2: pieza clave de la elevisión digital**, Instituto Oficial de Radio Televisión Española, 1995.

[28] D. Ghose, H. J. Kim, ***Scheduling Video Streams in Video-on-Demand Systems: A survey***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 11, Nº 2, 2000

[29] Gigabit Ehernet Alliance***, Gigabit Ethernet, accelerating the standard for speed***, White paper, 1998. http://www.gigabit-ethernet.org

[30] 10 Gigabit Ethernet Alliance, ***10 Gigabit Ethernet, Technology Overview***, White Paper. http://www.10gea.org/10GEA_Whitepaper_0901.pdf

[31] IEEE P802.3ae 10Gb/s Ethernet Task Force http://grouper.ieee.org/groups/802/3/ae/

[32] K.L. Gong, **Parallel MPEG-1 Video Encoding**, Tesis Doctoral, Departamento de EECS, Universidad de California, Berkeley, Mayo 1994. Technical Report.

[33] B. Hamidzadeh y Tsun-Ping J. To, ***Prioritized Admision Strategies to Improve User-Perceived Performance in Interactive VoD Servers***, Multimedia Tools and Applications, Kluwer Academic Publishers, nº 13, 2001.

[34] O. Hernández. ***Descripción del estándar MPEG-2.*** Universidad central de Venezuela. http://neutron.ing.ucv.ve/revista-e/No1/Mpeg2.htm

[35] K. A. Hua y S. Sheu, ***An Efficient Periodic Broadcast Technique for Digital Video Libraries,*** Multimedia Tools and Applications, Kluwer Academic Publishers Vol. 10, Nº 2/3, 2000

[36] A.J. Huttunen e I. Defée, ***Broadband MPEG-2 Cñient with Network Configuration capability,*** Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 11, Nº 3, 2001

[37] K. Hwang, Z. Xu. **Scalable Parallel Computing**. McGraw-Hill.

[38] InfiniBand™ Trade Association, *InfiniBand Architecture Specifications, vol.1, release 1.0,* Octubre de 2000.

[39] Página web de *InfiniBand™ Trade Association,* http://infinibandta.com

[40] ISO/IEC 1318-2 Draft International Standard, **Generic Coding of Moving Pictures and Associated Audio**. Part 2: Video, 1994

[41] J.H. Jeon y otros, ***Real Time MPEG-2 Video Codec Systems Using Multiple Digital Signal Processors***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 11, Nº 2, 2000

[42] H. Jin, **High Performance Mass Storage And Parallel I/O**, IEEE press

[43] C.Koelbel y otros. **The High Performance Fortran Handbook.** The MIT Press, Massachussets, 1994.

[44] S.D. Lago y otros, ***ARMIDA: Multimedia applications across ATM-based networks accessed via Internet navigation***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 5, Nº 2, 1997

[45] Página web de LAM, http://www.lam-mpi.org/

[46] P.W.K. Lie y otros, ***Threshold-Based Dynamic Replication in Large-Scale Video-on-Demand Systems***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 11, Nº 1, 2000

[47] *Myrinet Link Specifications*, http://www.myri.com/open-specs/link- history/index.html.

[48] Myricom, *Product List and Prices*, http://www.myri.com/myrinet/product_list.html

[49] A. Lombardo y otros, ***Traffic Specifications for the Transmission of Stored MPEG Video on the Internet***, IEEE Transactions on Multimedia, Volumen 3, Nº 1, marzo 2001.

[50] C. Marchisio y P. Marchisio, ***Media Touch: A Native Authoring Tool for MHEG-5 Applications,*** Multimedia Tools and Applications, Kluwer Academic Publishers, nº 14, pp5-22, 2001.

[51] ISO/IEC JTC1/SC29/WG12, *Multimedia and Hypermedia information coding Expert Group*, (MHEG), http://dual.km.giti.waseda.ac.jp/WG12/

[52] C. Montaner. ***El Sistema MPEG-****2*. Televisión Española. CINEVÍDEO 20, Nº 154, 1998.

[53] Página web de *Mosix*, http://www.mosix.org

[54] *The MPEG Home Page*. http://mpeg.telecomitalialab.com

[55] Página web de MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/

[56] Página web de MPI-FM, http://www-csag.ucsd.edu/projects/comm/mpi-fm.html

[57] Página web del MPI forum http://www.mpi-forum.org/

[58] Página web del estándar MPI, http://www-unix.mcs.anl.gov/mpi/index.html

[59] Página web de *MPI Software Technology*, http://www.mpi-softtech.com

[60] Página web de WMPI, http://dsg.dei.uc.pt/wmpi

[61] Página web del supercluster NT, http://archive.ncsa.uiuc.edu/SCD/Hardware/NTCluster/

[62] *Network of Workstations*, http://www.lri.fr/~fci/RS-Anglais.html

[63] *The Berkeley NOW Project*, http://now.cs.berkeley.edu/

[64] R.O. Onvural, **Asynchronous Transfer Mode Networks. Performance Issues**, Artech House, 2ª Edición, 1995.

[65] *Myrinet Overview* http://www.myri.com/myrinet/overview/index.html

[66] P.S. Pacheco, **Parallel Programming with MPI**, Morgan kaufman Publishers, 1997.

[67] *Myrinet Products*, http://www.paralline.com/produits/myrinet/product_list.en.html

[68] J. Peltoniemi, ***Video-on-Demand Overview***, Universidad de Tecnología de Tampere, Laboratorio de Telecomunicación. Tampere (Finlandia) 1995.

[69] R.Puigjaner, ***De las redes locales a las redes ATM***, IX Escuela de Verano de Informática, Tendencias en redes de altas prestaciones, Julio 1999, pp.1-44.

[70] Página web de PVM, http://www.epm.ornl.gov/pvm/pvm_home.html

[71] B. Raghavan y otros**, *A Gbyte/s Parallel Fiber-Optic Network Interface for Multimedia Applications,*** IEEE Network Nº 13, 1999.

[72] K.R. Rao, J.J. Hwang, **Techniques & Standars for Image, Video & Audio Coding,** Prentice Hall, 1996

[73] R. Seifert, **Gigabit Ethernet, Technology and Applications for high-Speed LANs**, Addison Wesley, 1998.

[74] D. Santo Orcero, ***Supercomputadores paralelos bajo Linux***, Linux Actual, nº 16, marzo 2000.

[75] D. Santo, *Supercomputadores clase Mosix: arquitectura y construcción*, Linux Actual nº13, Diciembre 1999.

[76] *Sistemas de Codificación de Vídeo*, Grupo de Procesamiento de la Señal, Universidad Politécnica de Cataluña http://gps-tsc.upc.es/imatge/Main/TEI/10_standards_vídeo/sld016.htm

[77] R. Shah, *Inifiniband set to emerge as high-speed PCI bus standard*, SunWorld, http://www.sunworld.com/swol-03-2000/swol-03-infiniband.html

[78] EEE Computer Society's, *A Listing of Parallel Computing Sites*, http://www.computer.org/parascope/#parallel

[79] W. Stallings, **ISDN and broadband ISDN with Frame Relay and ATM**, Prentice Hall, 3ª Edición, 1995

[80] W. Stallings, **Comunicaciones y redes de computadoras**, Prentice Hall, 6 Edición, 2000.

[81] B. Veeravalli, G. Barlas, ***Access Time Minimization for Distributed Multimedia Applications***, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 12, nº 273, 2000.

[82] V. Lo. *A beginners guide for MPEG-2 Standard*. Universidad de Hong Kong http://www.fhfriedberg.de/fachbereiche/e2/telekomabor/zinke/mk/mpeg2beg/beginnzi.htm

[83] A. Wittmann, *¿Hay un ASP en su futuro?*, Global Communication, Octubre de 2000.

[84] Página web del cluster Kaláka, *Distributed system for high performance parallel computing*, http://www.scitec.auckland.ac.nz/~peter/kalaka.html

[85] P. Piamsa-nga y otros, *A Parallel Model for Multimedia Database on Cluster System Environment*, Proceedings del congreso Internacional IEEE: International Synposium on Industrial Electronics (ISIE'98), Pretoria, Suráfrica, Julio, 1998

[86] F. Sijstermans, J.van der Meer, *CD-I Full Motion Video Encoding on a Parallel Computer*. Com. ACM 34, 4 (Abril.1991), pp. 81-91

[87] ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/encode

[88] K. Shen, L.A. Rowe, and E.J. Delp, *A Parallel implementation of an MPEG1 encoder : Faster than Real-Time*, Proceeding de la conferencia Internacional: SPIE Conference of Digital Video Compression: Algorithm and Technology 5-10 Feb. 1995. San José-California, pp. 407-418.

[89] N. Kralevich, *The NOW Split-C MPEG Encoder*, Internal Report Final Project CS267, Universidad de California en Berkeley. Technical Report.

[90] Y. Yu and D. Anastassiou, *Software implementation of MPEG2 video encoding using socket programming in LAN*, Proceedings de la conferencia Internacional: SPIE Conference of Digital Video Compression on Personal Computer, Algorithm and Technology 7-8 Feb. 1994. San José-California, pp. 229-240.

[91] Luis, F. Martínez. *An efficient ISO/MPEG-1 Layer II Encoder Using Parallel Processing Techniques.* Proyecto de investigación de la Universidad de Miami, Florida 1995

[92] R.J. Gove, *The MVP : A Highly-Integrated Video Compression Chip*, Proceedings de la Conferencia Internacional IEEE: Data Compression, 28-31 March. 1994. Utah, pp. 215-224.

[93] S.M. Akramullah, I. Ahmad, M.L. Liou. *Performance of Software-based MPEG-2 video encoder on parallel and distributed systems*. IEEE Transactions on Circuits and Systems for Video Technology, 1997.

[94] Página web proyecto BIP, http://www.ens-lyon.fr/LIP/RESAM/index_bip.html

[95] Página web del software y documentación de Myrinet, http://www.myri.com/scs/

[96] *System Administration Guide*. Sun Microsystems. http://docs.sun.com

[97] Amir Afzal, **Introducción a Unix. Un enfoque práctico**. Prentice Hall, 1997.

[98] J. Tackett Jr, D. Gunter y L. Brown, **Linux, Edición Especial**, Prentice Hall Hispanoamericana, S.A.1996.

[99] J.A. Martínez, *El sistema de ficheros virtual (VFS) de Linux*, Linux Actual, Nº 10.

[100] U. Vahalia, **UNIX Internals, the new frontiers**, Prentice Hall, 1996.

[101] J.H. Howard, M. L. Kazar, S.G. Menees y otros, *Scale and Performance in a Distributed File System*, Proceedings de la Conferencia Internacional ACM: Transactions on Computer Systems, Vol.6, Nº 1, Febrero 1988, pp. 51-81.

[102] *White Paper: The NFS Distributed File Service*, Marzo 1995, http://www.sun.com/software/white-papers/wp-nfs/

[103] *NFS Administration Guide*, Sun Microsytems, Agosto 1997, http://docs.sun.com

[104] H. Stern, **Managing NFS and NIS**, O'Reilly & Associates, Inc.1992.

[105] B. Callaghan, **NFS Illustrated,** Addison Wesley, 2000

[106] RFC 1094: *NFS: Network File System Protocol Specification*

[107] RFC 1831: *RPC: Remote Procedure Call Protocol Specification Version 2*

[108] RFC 1832: *XDR: External Data Representation Standard*

[109] J. S. Vitter and P. Krishnam, *Optimal Prefetching via Data Compression*, Journal of the ACM, 43(5), September 1996, 771-793. Proceedings del 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS'91), San Juan, Puerto Rico, October 1991.

[110] T. Mitra, C. Yang y T. Chiweh, *Application-Specific File Prefetching for Multimedia Programs*, Proceedings de la Conferencia Internacional IEEE: Multimedia and Expo, ICME'2000

[111] F. H. P. Fitzek and M. Reislein, *A Prefetching Protocol for Continuous Media Streaming in Wireless Environments*, Technical Report TKN-00-05, Telecommunications Network Group, Technical University Berlin.

[112] W. Richard Stevens, *TCP/IP Illustrated, Vol.1 The Protocols* http://lib.nevalink.ru/tcp_stevens/index.html

[113] Olaf Kirch, *Guía de Administración de Redes con Linux*, Traducción proyecto Lucas, Julio 1997.

[114] RFC 1813: *NFS Version 3 Protocol Specification*

[115] *NFS: Network File System Version 3 Protocol Specification*, Sun Microsystems, 1993. http://docs.sun.com

[116] Bob Friesenhahn, *A file system for the Web*, Byte Magazine, Noviembre 1996.

[117]Brent Callaghan, *WebNFS, the filesystem for the Internet*, abril 1997, Sun Mycrosystems. http://www.sun.com/software/white-papers/

[118] RFC 2054: *WebNFS Client Specification*

[119] *NFS version 4 protocol Internet-draft*, NFS version 4 working group, Febrero 2000. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-06.txt

[120] *NFS Version 4, Technical Brief*, Octiber 1999, Sun Microsystems

[121] *Network File System Version 4*, Transport Area Group, IETF, Mayo de 2000. http://www.ietf.org/html.charters/nfsv4-charter.html

[122] RFC 3010: *NFS version 4 Protocol*

[123] *Linux as File Server*, http://ayesha.phys.virginia.edu/~bryan/Linux/

[124] Hongjiu Lu, *NFS server in Linux: past, present and future*, VA Linux Systems, 1999.

[125] V. Srinivasan y otros, *Spritely NFS: Experiments with Cache-Consistency protocolos*, Proceedings del vigésimo Simposio: Operating System Principles, Arizona, Diciembre 1989

[126] D. Nagle, G. Ganger, J. Butler, G. Goodson y C. Sabol, *Network Support for Network-Attached Storage*, Proccedings del Hot Interconnects, Agosto 1999.

[127] G.A. Gibson y R.V. Metter, ***network Attached Storage Architecture,*** Communications of the ACM, Noviembre del 2000, Vol 43, nº 11

[128] J. S. Chase y otros, N*etwork I/O with Trapeze*, Proceedings del Hot Interconnects, Agosto 1999

[129] *Coda File System***,** http://www.coda.cs.edu/doc/html/index.html

[130] M. Donald Dahin, **Serverless Network File System,** Tesis Doctoral, Universidad de California en Berkeley, 1995

[131] *The Now project*, http://now.cs.berkeley.edu

[132] T. E. Anderson y otros, *Serverless Netwok File System*, Operating System Principles, ACM Transactions on Computer Systems, 1995

[133] C. Thekkath, T. Mann, y E. K. Lee. *Frangipani: A scalable distributed file system.* Proceedings del 16th ACM Symposium on Operating Systems Principles, pages 224-237. ACM Press, Octubre 1997.

[134] R. Eckstein y otros, **Using Samba**, O'Reilly & Associates, Inc.1ª edición, 1999

[135] T. V. Veschler, ***La unión hace la fuerza,*** Linux Actual, Jlio/Agosto 1999

[136] *The Parallel Virtual File System*, http://parlweb.parl.clemson.edu/pvfs/

[137] A. Vyas, **NFS Extensions for Transparent Access to Remote Devices,** MTech Thesis, http://www.cse.iitk.ac.in/research/mtech1999/9911107.html

[138] V. Sharma, **Desig and Implementation of a Portable and Extensible FTP-to-NFS Gateway,** Mtech Thesis, http://www.cse.iitk.ac.in/research/mtech1999/9911156.html

[139] T. Olivares, V. López y otros, **Linux, Guía de Instalación y Administracion**, McGraw-Hill, 2000.

[140] M. Beck y otros, **Linux Kernel Internals**, 2ª edición, Addisson-Wesley

[141] Página web de Red Hat, http://www.redhat.com/

[142] T.Olivares, F. Quiles, A. Garrido, P.J. García y L. Orozco-Barbosa, *Evaluación del sistema de ficheros NFS en un cluster de altas prestaciones con aplicaciones multimedia,* Actas de las XI Jornadas de Paralelismo, Septiembre de 2000, Granada.

[143] T. Olivares, P. Cuenca, F. Quiles, A. Garrido, J. Sanchez and J.Duato, *Interconnection network behavior on a multicomputer in the parallelization of the MPEG coding algorithm. Worm-hole vs Packet-Switching Routing*, Proceedings de la cuarta Conferencia Internacional: High Performace Computing (HiPC'97), IEEE Computer Society Press, ISBN 0-8186-8067-9, Vol. 1, pp. 48-53, Diciembre 1997, Bangalore (India).

[144] T. Olivares, P. Cuenca, F. Quiles and A. Garrido, *Parallelization of the MPEG coding Algorithm over a Multicomputer. A Proposal to Evaluate its Interconection Network.* Proceedings del IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM´97), IEEE Computer Society Press, Vol. 1, pp. 113-116, 1997, CANADA.

[145] T.Olivares, F.Quiles, P.Cuenca, L. Orozco y I. Ahmad, *Study of data distribution techniques for the implementation of an MPEG-2 video encoder*, Proceedings de la Conferencia Internacional IASTED: Parallel and Distributed Computing and Systems (PDCS'99), ISBN: 0-88986-275-3, Vol. 1, pp. 537-542, 1999, Cambridge, Massachussets (USA)

[146] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Tuning NFS on high speed networks with parallel multimedia applications,* Proceedings de la Conferencia Internacional IASTED: Parallel and Distributed Computing and Systems (PDCS'2000), ISBN: 0-88986-304-0, Vol. 1, pp. 13- 17, Noviembre 2000, Las Vegas, Nevada (USA).

[147] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Performance study of NFS over Myrinet-based clusters for parallel multimedia applications*, Proceedings de la Canadian Conference On Electrical And Computer Engineering, (IEEE Computer Society Press), Mayo 2001, Toronto (Canadá)

[148] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *A network File System evaluation over a high-speed multimedia clustered architecture*, Proceedings de la V World Multiconference on Systemics, Cybernetics and Informatics (Invited Session: Multimedia on Clusters of Workstations: Mastering the future), Volume XII, Part II, pp.374-380, 22-25 Julio 2001, Orlando, Florida (USA).

[149] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *The Need of Multicast Predictive NFS Servers for High-Speed Networks used as Parallel Multimedia Platforms*, Proceedings del ICPP'01 Worshop on Scheduling and Resource Management for Cluster Computing, ISBN: 0-7695-1260-7, pp.391-396, IEEE Computer Society Press, Septiembre 2001, Valencia (España)

[150] P.Cuenca, T.Olivares, A.Garrido, F.Quiles and L.Orozco-Barbosa, *Techniques to Increase MPEG-2 Error Resilence in the VBR Video Transmission over ATM Networks*. Proceedings del IEEE International Conference on Communications (ICC'98), ISBN 0-7803-4788-9, Vol. 1, pp. 869-873, 1998, Atlanta (USA).

[151] F. Pelayo, F. Cuartero, V. Valero, D. Cazorla, T. Olivares, *Specification and Performace of the MPEG-2 Video Encoder by using the Stochastic Process Algebra: ROSA*, Proceedings del 2001 UK Performance Engineering Worshop (UKPEW'2001), pp 105, Julio 2001, UK.

[152] *NFS Server Performance and Tunning Guide*, http://docs.sun.com

[153] S.M.Akramullah, I.Ahmad, M.L.Liou, M. Kafil. *A scalable off-line MPEG-2 video encoding scheme using a multiprocessor system*. Parallel Computing. Julio de 2000.

[154] J. S. Vitter and P. Krishnam, *Optimal Prefetching via Data Compression*, Journal of the ACM, 43(5), September 1996, 771-793.

[155] R. D. Barve, M. Kaldahalla, P. J. Varman and J. S. Vitter, *Competitive parallel Disk Prefetching and Buffer Management*", Proceedings del 5th Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS), San Jose, California, 1998.

[156] *The TPIE project*, http://www.cs.duke.edu/TPIE

[157] D. E. Vengroff and J.S. Vitter, *I/O-Efficient Algorithms and Environments*, ACM computing Surveys, 28(4), Diciembre 1996.

[158] *Transparent Informed Prefetching and Caching: Overview*, http://www.pdl.cs.cmu.edu/TIP/TIP.html

[159] T. M. Madhyastha, G. A. Gibson and C. Faloutsos, *Informed Prefetching of Collective I/O request*, Proceedings del SC'99.

[160] G.A. Gibson et al., *A Cost-Effective High-Bandwith Storage Architecture*, Proceedings de la 8th Conference on Architectural support for Programming Languages and Operating Systems, 1998.

[161] G. A. Gibson and R. Van Meter, *Network Attached Storage Architecture*, Communications of the ACM, Vol.43, No. 11, Noviembre de 2000.

[162] K. Amiri, G. A. Gibson, R. Golding, *Highly Concurrent Shared Storage*, Proceedings de la International Conference on Distributed Computing Systems, Taipei, Abril de 2000.

[163] F. Chang and G. A. Gibson, *Automatic I/O hint generation through Speculative Execution*, Proceedings del 3rd Symposium on Operating Systems Design and Implementations (OSDI), Febrero de 1999.

[164] J. Pâris, S. W. Carter and D. E. Long, *A reactive Broadcasting Protocol for Video on Demand*, Proceedings de la Multimedia Computing and Networking Conference, San José, SPIE, Enero de 2000.

[165] W. Freeman and E. Miller, *Design for A Decentralized Security System for Network Attached Storage,* 8th Goddard Conference on Mass Storage Systems and Technologies, College Park, MD, Marzo de 2000.

[166] T. Kroeger and D. E. Long, *The Case for Efficient File Access Pattern Modeling*, Proceedings del 7th Workshop on Hot Topics in Operating Systems (HotOS-VII). Rio Rico, Arizona, Marzo de 1999.

[167] T. Kroeger, D. E. Long and J. C. Mogul, *Exploring the Bounds of Web Latency, Reduction from Caching and Prefetching*, USENIX Symposium on Internet Technologies and Systems, 1997.

[168] P. Rodriguez, K. W. Ross and E. Biersack, *Distributing Frequently Changing Documents in the web: Multicasting or Hierarchical caching*, Computer Networks and ISDN Systems, Vol. 30, Noviembre 1998

[169] J. Kangasharju, F. Hartanto, M. Reisslein and K. W. Ross, *Distributed layered Encoded Video through Caches*, IEEE Infocom 2001.

[170] M. Reisslein, K. W. Ross and V. Verillotte, *Descentralized Prefetching for VBR Video on Demand*, Proceedings of ECMAST'98, Berlin, Germany, Mayo de 1998.

[171] M. Reisslein and K. W. Ross, *High-Performance Prefetching Protocols for VBR Prerecorded Video*, IEEE Networking Magazine, Nov/Dic. 1998.

[172] T. Cortés and J. Labarta, *Linear Aggresive Prefetching: a way to increase the Performance of Cooperative Caches*, International Parallel Processing Symposium (IPPS'99), San Juan, Puerto Rico, Abril de 1999.

[173] T. Cortés and J. Labarta, *PAFS: a new generation in Cooperative Caching, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, CSREA press, Julio 1998

[174] C. A. Gerlhot and A. Kemper, *A Multi-Threaded Architecture for Prefetching in Object Bases*, Proceedings de la 4th International Conference Extending Database Technology (EDBT), Marzo 1994

[175] P. Ranganathan, V. S. Pai, H. Abdel-Shafi, S.V. Adve, *The Interaction of Software Prefetching with ILP Processors in Shared Memory Systems*, Proceedings del 34th Annual International Symposium on Computer Architecture, 1997

[176] M. M. Müller, T. M. Warschko and W. F. Tichy, *Prefetching on the Cray-T3E*, Proceedings de la International Conference on Supercomputing, Julio de 1998.

[177] F. Kozodoy, *Simple Client Prefetching Analysis*, http://www.cs.bu.edu/faculty/best/res/projects/ClientProfiling/Home.html

[178] C. L. Chee, H. Lu, H. Tang and C. V. Ramamoorthy, *Improving I/O responses times via Prefetching and Storage System Reorganization*, Proceedings de COMPSAC'97, 21st International Computer Software and Applications Conference, 1997

[179] M. Eldridge, *Prefetching in a Texture Cache Architecture*, Proceeding on the Worshop on Grphics Hardware, Lisboa, Portugal, 1998

[180] C. Chi and J. Yuan, *Design Considerations of High Performance Data Cache with Prefetching*, Lectures Notes in Computer Science 1685, p. 1243. Agosto de 1999

[181] D. Revel, D. Mc Namee, D. Steere and J. Walpole, *Adaptative Prefetching for Device-Independent File I/O*, Proceeding de Multimedia computing and Networking, 1998

[182] C. Vaill, **A Workgroup Model for Cache Prefetching and Recommendation**, MS thesis proposal 1999, http://www.cs.columbia.edu/~cvaill/proposal/proposal.html

[183] *The Latency Tolerance/Reduction for www Systems Project* http://www.ii.uib.no/~markatos/arch-vlsi/www.html

[184] V. N. Padmanabhan and J. C. Mogul, *Using Predictive Prefetching to Improve World Wide Web latency*, Proceeding de la ACM SIGCOMM Conference, Standford, CA, 1996

[185] *WWW Collector- The Prefetching Proxy Server for www*, http://shika.aist-nara.ac.jp/products/wcol/wcol.html

[186] E. P. Markatos, *Main Memory Caching of Web Documents*, Proceedings de la 5th International WWW Conference, 1998.

[187] T. Mitra, C. Yang and T. Chiweh, *Application-Specific File Prefetching for Multimedia Programs*, Proceedings de la IEEE International Conference on Multimedia and Expo, ICME'2000

[188] F. H. P. Fitzek and M. Reislein, *A Prefetching Protocol for Continuous Media Streaming in Wireless Environments*, Technical Report TKN-00-05, Telecommunications Network Group, Technical University Berlin.

[189] J. B. Weissman, M. Marina, M. Gingras, *Optimizing Remote File Access for Parallel and Distributed Network Applications*, Journal of Parallel and Distributed Computing, Vol. 61, Nº 11, Noviembre de 2001.

[190] T.M. Kroeger y D. D.E. Long, *Design and Implementation of a predictive File Prefetching Algortihm*, Proccedings de la conferencia Técnica Anual USENIX, Junio de 2001, Boston.

[191] M. K. McKsick y otros, **The Network File System**, capítulo 9 del libro The Design and Implementation of the 4.4BSD Operating System, Addison-Wesley Publishing Company, Inc. 1996.

[192] T.V. Veschler Cox, *La unión hace la fuerza*, Linux Actual, Julio/agosto 1999.

[193] H.Lei y D. Duchamp, *An Analitical Approach to File Prefetching*, Proceedings de la conferencia Técnica Anual USENIX, Anaheim, California, 1997.

[194] J.L. Hennessy & D.A. Patterson, Computer Architecture a Quantitative Approach, Morgan Kaufmann Publishers, 3ª edición, 2002