Universidad Carlos III de Madrid
Escuela Politécnica Superior
Departamento de Informática

Tesis Doctoral

Sistema de ficheros paralelo escalable para entornos *cluster*

Autor: Luis Miguel Sánchez García
Directores: Félix García Carballeira
Jesús Carretero Pérez

Leganés, 2009

TESIS DOCTORAL

Sistema de ficheros paralelo escalable para entornos *cluster*

AUTOR: **Luis Miguel Sánchez García**
DIRECTORES: **Félix García Carballeira**
Jesús Carretero Pérez

FIRMA DEL TRIBUNAL CALIFICADOR

Firma

PRESIDENTE:

VOCAL:

VOCAL:

VOCAL:

SECRETARIO:

CALIFICACIÓN:

Leganés, a ..... de ..... de 2009

A Laura,
por el apoyo y cariño
prestados todo este tiempo.

A mis padres,
por toda una vida
dedicada a sus hijos.

Agradecimientos

He de dar las gracias a mucha gente que a lo largo de todo este tiempo me ha dado su apoyo y confianza. Empezaré por mis directores de Tesis, Félix y Jesús, que me animaron a comenzar esta aventura, así como a todos mis compañeros del grupo ARCOS que me han ayudado en todo momento.

También quiero agradecerselo a mis padres Emilio y Jacinta, por todos los años de sacrificio realizados. Y a mi hermano Emilio, por indicarme el camino a seguir. Por último a Laura, por su comprensión y apoyo durante todo este tiempo.

Muchas gracias a todos.

Luis Miguel Sánchez García.

Resumen

En la actualidad, las aplicaciones utilizadas en los entornos de computación de altas prestaciones (como por ejemplo simulaciones científicas o las dedicadas a la extracción de datos (*data-mining*)), manejan ingentes cantidades de información, por lo que necesitan enormes recursos de cómputo y memoria.

Las arquitecturas *cluster* se han convertido en la solución más común para ejecutar este tipo de aplicaciones. Destacan dos tipos: las construidas por la agregación de componentes heterogéneos y las basadas en el uso de recursos homogéneos. Las arquitecturas heterogéneas enfrentan a la problemática de combinar distintas tecnologías hardware y software, ésta reside en la dificultad de integrar diferentes sistemas de almacenamiento. Por otra parte, los grandes *clusters* lidian con el desequilibrio entre nodos de cómputo y de E/S del sistema de almacenamiento, ya que al ser mayor el número de nodos de cómputo, la E/S se convierte en un cuello de botella para las aplicaciones.

Las soluciones actuales para superar las dificultades que muestran estas arquitecturas *cluster* con respecto la adaptación de los nodos con el fin de eliminar la heterogeneidad en el primer caso, y la utilización de sistemas de ficheros paralelos así como el incremento en la infraestructura del sistema de almacenamiento en el segundo. Estas tendencias tienen enormes costes económicos y temporales en la adaptación y configuración de la infraestructura de E/S.

La presente tesis propone solucionar estos problemas, estableciendo los siguientes objetivos:

- Proporcionar un acceso homogéneo a los datos y usar tecnologías estándar de E/S para la construcción del sistema de almacenamiento en entornos heterogéneos.
- Equilibrar la carga de E/S producida por las aplicaciones y eliminar la sobrecarga de los sistemas de almacenamiento en entornos de gran escala.

Para alcanzar estos objetivos se han definido las siguientes soluciones:

- Un sistema de ficheros paralelo multiplataforma basado en el uso de tecnologías estándar que persigue: la formación de sistemas de almacenamiento para *clusters* heterogéneos y proporcionar una plataforma que homogenice el acceso a los datos de las aplicaciones.
- Una arquitectura de E/S basada en la ampliación de los esquemas de jerarquía de memoria al entorno de los grandes *clusters*, que incremente el número de nodos de E/S de los *clusters* para aumentar el paralelismo y reducir los accesos al sistema de almacenamiento.

A lo largo del presente documento se detallan las soluciones propuestas, así como las evaluaciones de las mismas.

Abstract

Nowadays, the applications used in environments high performance computing, such as simulations scientific applications dedicated to data extraction (*data-mining*), manage large amounts of information, needing huge computing and memory resources.

Cluster architecture is the most common solution for HPC applications. There are two kinds of cluster architectures: first, based on the aggregation of heterogeneous components and others, built with homogeneous components of large-supercomputers. Heterogeneous cluster architectures have a main problem, because it is built using different hardware and software technologies. There are no parallel file systems to adapt all of these diverse technologies available on these architectures. Moreover, homogeneous large-clusters have an I/O imbalance problem. This is due to the large number of compute nodes available compared to the few number of I/O nodes. This imbalance converts the I/O system on a bottleneck for HPC applications.

The most common approach to remove the heterogeneity of the clusters is the adaptation of the nodes integrating technology to allow compatibility with new systems. Moreover, in the case of large clusters, traditional solutions are the use of parallel file systems and include changes in the infrastructure of the storage system, such as increasing the number of I/O nodes. In both cases, the solutions have high economic and time costs in the adaptation and configuration of the I/O infrastructure.

This thesis proposes a solution for the problems presented above. The goals are the following:

- Providing uniform data access using standard I/O technologies with the purpose of constructing storage systems in heterogeneous environments.
- Balancing effective I/O load and eliminating the overhead of storage systems in large scale environments.

To achieve these objectives we designed the following solutions:

- A parallel file system platform based on the use of standard technologies for the formation of storage systems for heterogeneous clusters, providing further homogenice platform data access to applications.
- An I/O architecture based on the extension of the diagrams of the hierarchy of memory to the large clusters environment, increasing the number of I/O nodes of the clusters to improve the parallelism and to reduce the I/O access to the storage.

This document details the proposed solutions and shows the evaluations of them.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos.

Capítulo 1

Introducción

En el presente capítulo se realizará una introducción de la tesis y una exposición de los puntos más importantes que rodean a la misma. Primero se contextualizará y a continuación se detallará la motivación y la estructura del resto del documento.

1.1. Introducción

Actualmente, las tecnologías de la información, en múltiples y variados ámbitos, tales como, la industria, la banca, o entornos científicos, necesitan cada vez más, aplicaciones que requieren numerosos recursos, tanto de cómputo como de almacenamiento. Algunos ejemplos de estas aplicaciones las encontramos en: el análisis y gestión de grandes volúmenes de información de ámbito empresarial (como *data mining* o *data warehouse*), la simulación de fenómenos físicos (como el estudio de altas energías [1]) y el procesamiento de datos vía satélite, etc. Este problema se ha tratado de solucionar mediante la construcción y mejora de arquitecturas de computación adecuadas a sus necesidades, lo que ha llevado a la creación de una gran cantidad de computadores de enorme capacidad, denominados "supercomputadores".

El desarrollo de este tipo de arquitecturas se ha visto beneficiada por un incremento en la potencia de los procesadores y sistemas de almacenamiento. La computación de altas prestaciones ha evolucionado de los grandes y costosos supercomputadores a entornos formados por redes o *clusters* de estaciones de trabajo [2]. Así lo refleja el hecho de que en la lista de los 500 supercomputadores más potentes del mundo [3] cada vez es mayor el número de *clusters* que aparecen. En noviembre de 2008, el 82 % de los supercomputadores de la lista eran *clusters* frente a un 17 % de computadores MPP (*Massively Parallel Processing*). Esta tendencia se ha beneficiado de la reducción coste/prestaciones de las tecnologías usadas.

Las arquitecturas basadas en *clusters* se pueden clasificar en dos grandes grupos:

- *Clusters heterogéneos*: construidos a partir de la agregación de recursos que utilizan distintas tecnologías. Pueden encontrarse múltiples proyectos orientados en esta dirección [4, 5], principalmente debido al ahorro que supone la reutilización de componentes (sistemas de almacenamiento, nodos de cómputo, etc.) para formar computadores con mayores capacidades de cómputo, de almacenamiento, etc.

- *Clusters de gran escala*: grandes instalaciones donde predomina la homogeneidad de los elementos utilizados para su construcción. Supercomputadores de este tipo son: Marenostrum [6], IBM ASCI Purple [7], etc.

Las soluciones tradicionales para conseguir almacenamiento de altas prestaciones en los entornos *cluster* son:

- Servidores de almacenamiento de datos denominados *networked attached storage* (NAS) que disponen de *arrays* de discos. Utilizan sistemas de ficheros distribuidos como NFS [8] orientado a sistemas Unix o CIFS [9] para entornos Windows (véase la Figura 1.1).

- Redes de almacenamiento de datos llamadas *storage area networks* (SAN) (véase la Figura 1.1). Existen dos posibilidades en este tipo de arquitectura:

  - Disponer de servidores que administren los dispositivos almacenamiento.
  - *Network Shared Disk* (NSD), donde los clientes acceden a los datos de los dispositivos de almacenamiento directamente.

  En este tipo de modelo de almacenamiento es frecuente utilizar sistemas de ficheros paralelos (SFP) como GPFS [10] o PVFS [11] para mejorar las prestaciones del sistema o sistemas de ficheros compartidos como OCFS [12].

Entre dichas soluciones para las arquitecturas *cluster* la más utilizada es la que se basa en el uso de servidores de almacenamiento, por lo que la tesis se centrará en ésta.

1.2. Problemática

En la E/S de los entornos *cluster* encontramos distintos problemas:

- En *clusters* heterogéneos:

  - Dificultad en la agregación de distintas tecnologías de E/S para la formación de los sistemas de almacenamiento.
  - Falta de flexibilidad y generalidad en los sistema de almacenamiento: cualquier cambio en el sistema de ficheros o en la arquitectura de almacenamiento provoca una reconfiguración en los nodos de cómputo.

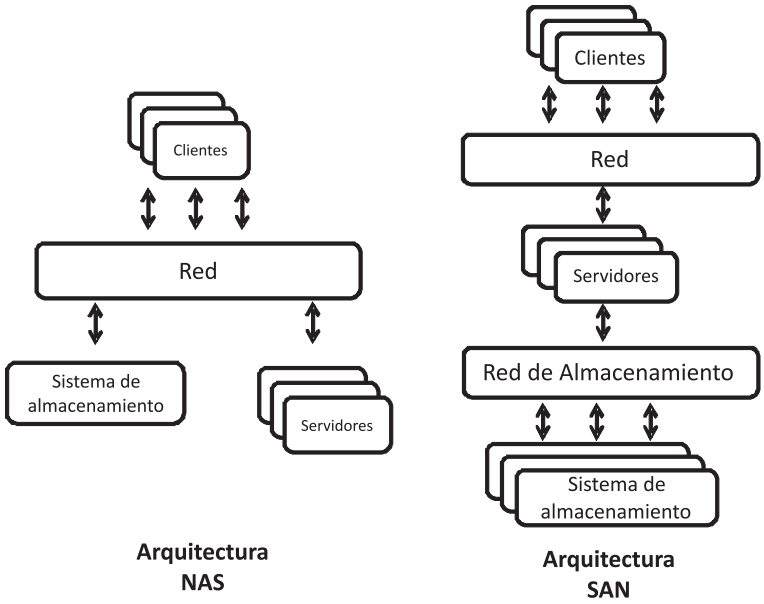

Figura 1.1: Modelos de almacenamientos NAS y SAN.

- Integración de sistemas de ficheros paralelos estándar en entornos heterogéneos lo que dificulta la portabilidad de estos sistemas de ficheros a distintos entornos.

- En *clusters* de gran escala:

  - Falta de escalabilidad de los sistemas de almacenamiento. El número de nodos de E/S del sistema de almacenamiento de los *cluster* es muy inferior al de procesadores y nodos de cómputo del sistema (tal y como se refleja en la Tabla 1.1). Este desequilibrio provoca que la carga de E/S generada por las aplicaciones desborde a los sistemas de ficheros usados para gestionar estos nodos de E/S.

    Tradicionalmente para solventar este problema se han utilizado sistemas de ficheros paralelos y se han incrementado en el número de nodos de E/S o/y las prestaciones de los sistema de E/S involucrados (discos duros, redes de datos, etc.).

Nombre	Número de procesadores por nodo de E/S	Número de nodos de cómputo por nodo de E/S
Marenostrum [6]	228,2	57,05
Magerit [13]	200	100
IBM ASCI Purple [7]	87,5	11

Tabla 1.1: Relación entre componentes y nodos de E/S.

Sin embargo estas soluciones presentan grandes costes en tiempo (administración, instalación, etc) y dinero por lo que se dificulta su implantación.

1.3. Objetivos

Ante el aumento de necesidades de E/S que encontramos en los *clusters*, esta tesis plantea como objetivo principal: *resolver los problemas de E/S presentes en las arquitecturas cluster, estableciendo mecanismos para eliminar la heterogeneidad en el acceso a los datos y mejorar la escalabilidad.*

La tesis estudiará las arquitecturas existentes en la actualidad y sus políticas de encaminamiento asociadas, con el objeto de determinar las posibles soluciones que permitan eliminar los problemas anteriormente detallados.

Se analizarán las arquitecturas *cluster* así como los distintos niveles de E/S presentes, para posteriormente diseñar las soluciones concretas a los problemas ya mencionados.

Objetivos más específicos de esta tesis son:

- En *clusters* heterogéneos:
  - Homogeneizar el acceso a los datos.
  - Permitir la integración de múltiples tecnologías para la construcción de un sistema de almacenamiento para un *cluster*.
  - Facilitar la administración del sistema de E/S heterogéneo permitiendo añadir nodos sin perjuicio del manejo del mismo por parte de las aplicaciones.
- En *clusters* de gran escala:
  - Incrementar el nivel de paralelismo de E/S de los sistemas de almacenamiento de los *clusters*.
  - Equilibrar la carga de E/S producida por las aplicaciones.
  - Reducir la carga sobre los sistemas de almacenamiento, mejorando la escalabilidad.

Se proponen dos soluciones distintas para cada caso:

- En *clusters* heterogéneos: diseñar una plataforma software que permita el acceso homogéneo a los datos y que utilice servidores de E/S estándares para conformar un sistema de almacenamiento del *cluster*.
- En *clusters* de gran escala: definir una arquitectura software de E/S que amplíe los esquemas de jerarquía de memoria al entorno de los grandes *clusters*.

1.4. Estructura del documento

El resto del trabajo se organiza en los siguientes capítulos:

- El Capítulo 2 analiza el estado de la cuestión en cuanto a las diferentes tecnologías existentes para almacenamiento en computación paralela y distribuida, así como las principales técnicas de E/S paralela utilizadas en dichos sistemas.
- El Capítulo 3 presenta de forma más detallada la motivación de la tesis, así como una introducción a las soluciones propuestas: el sistema de ficheros paralelo *Expand*, en el caso de los *cluster* heterogéneos, y la arquitectura basada en el uso de nodos intermedios de almacenamiento, para entornos de gran escala.
- El Capítulo 4 describe el sistema de ficheros paralelo *Expand*, presentando sus principales características.
- El Capítulo 5 presenta la arquitectura de E/S basada en el uso de nodos de almacenamiento intermedio para entornos de computación de gran escala.
- El Capítulo 6 presenta las evaluaciones realizadas y los resultados obtenidos del sistema de ficheros paralelo *Expand*, así como de la arquitectura de E/S propuesta para entornos de computación de altas prestaciones.
- Finalmente, el Capítulo 7 describe las principales conclusiones obtenidas del trabajo desarrollado, las principales líneas de trabajo futuro y las aportaciones de la tesis.

Por último se incluye la bibliografía junto a algunas referencias a diversas páginas Web utilizadas.

Capítulo 2

Estado de la cuestión

En este capítulo se presentarán aspectos y detalles de la tesis, así como trabajos relacionados para la consecución de la misma. Además, se contemplarán en profundidad las capas del sistema de E/S implicadas, usadas en entornos de computación de altas prestaciones (indicadas en la Figura 2.1):

- Topologías en los sistemas de almacenamiento.
- Sistemas de ficheros.
- Técnicas de optimización de la E/S.
- Bibliotecas e interfaces de E/S.
- Aplicaciones.

Por último, se mostrará un resumen con los puntos más importantes.

2.1. Topologías en los sistemas de almacenamiento

En [14] se define el concepto de topología aplicado a sistemas de almacenamiento como aquello que imprime un orden particular de los protocolos específicos de almacenamiento, mecanismos de transporte y conexiones físicas. Esto incluye discos duros y otros medios (como medios fijos o rotatorios magneto-ópticos) junto con su conexión, red de interconexión y técnicas *software* asociadas usadas para implantar estos dispositivos.

De la solución tradicional conseguida mediante la conexión directa de los sistemas de almacenamiento se ha evolucionado a un conjunto de soluciones, todas ellas basadas en el uso de una red de interconexión de alta velocidad que conecta

un conjunto de sistemas de almacenamiento con los computadores que acceden a los datos allí almacenados. Al conectar el sistema de almacenamiento directamente a una red, dicho sistema se separa de los servidores lo que permite, por ejemplo, añadir más espacios de almacenamiento sin paradas. La tendencia a "virtualizar" el almacenamiento (muestra una visión lógica a los usuarios libre de la solución física realmente usada) facilita la administración, ofreciendo una mayor flexibilidad.

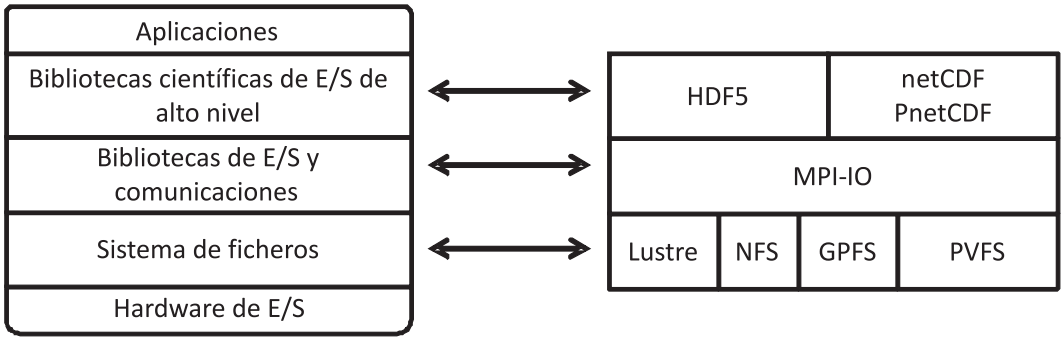

Figura 2.1: Capas del sistema de E/S.

Las principales soluciones para la conexión de un sistema de almacenamiento son (ordenadas de menor a mayor complejidad):

- *Directly Attached Storage* (DAS): La arquitectura DAS es la solución tradicional de conexión directa al computador mediante tecnología IDE o SCSI. Para conectar dispositivos de almacenamiento más heterogéneos se dispone de las interfaces FireWire (IEEE 1394) y USB.

- *Networked attached storage* (NAS): La topología interna de una NAS consiste en un servidor de ficheros que mueve los datos entre un *array* de discos y una red. El sistema de interconexión se basa en las tecnologías Ethernet y Gigabit Ethernet, usando protocolos de Internet estándares para ofrecer un servicio orientado a servir ficheros.

- *Storage Area Networks* (SAN): La topología interna de una SAN contiene redes dedicadas y auto-gestionadas de dispositivos de almacenamiento que se unen a los servidores a través de redes de alta velocidad. Una SAN ofrece un servicio de bloques. En este caso existen dos posibilidades arquitectónicas para la disposición de los dispositivos de almacenamiento:

  - Se dispone de servidores de almacenamiento que regulan y administran los recursos de estos dispositivos. Es la solución tradicional en este tipo de arquitecturas.

- *Network Shared Disk* (NSD): donde los clientes acceden directamente a los datos disponibles en los dispositivos de almacenamiento. Esto incrementa el rendimiento de las aplicaciones. Sin embargo, presenta problemas como un gran coste en las conexiones de los nodos de cómputo con los sistemas de almacenamiento, en gestión de los recursos, etc.

En ambos casos para acceder a los datos se utilizan tecnologías como: iSCSI o fiberchannel (FC), aunque existen versiones de estas tecnologías sobre otros sistemas como Ethernet, que tiene un menor coste aunque una mayor latencia y menor ancho de banda.

La Figura 2.2 muestra una representación de los tres tipos de arquitecturas de almacenamiento.

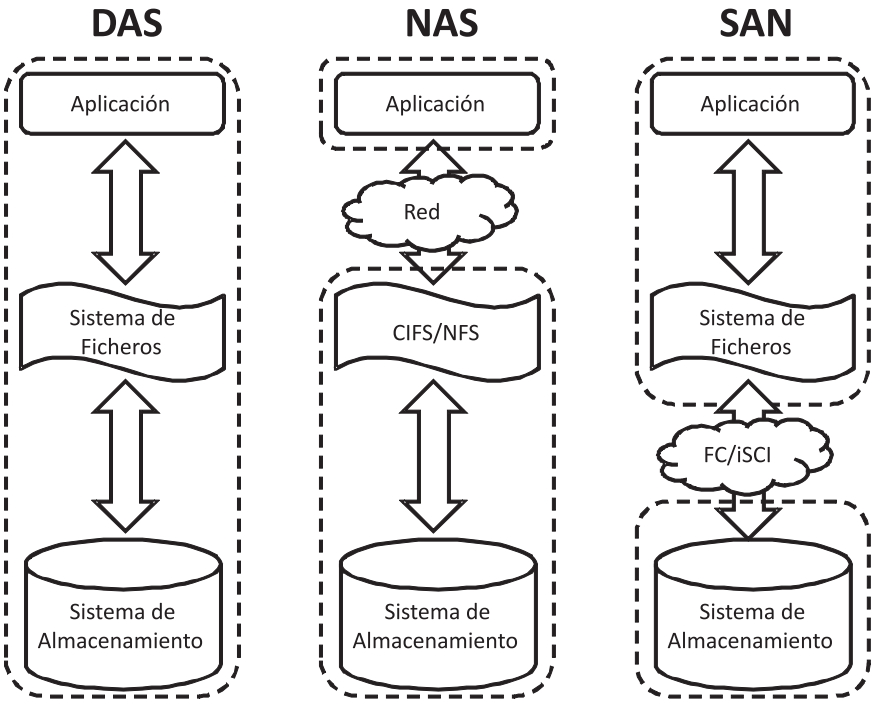

Figura 2.2: Topologías en los sistemas de almacenamiento.

2.2. Sistemas de ficheros

En computación, un sistema de ficheros es un método para el almacenamiento y organización, de ficheros de computadora y los datos que éstos contienen, con el fin de facilitar la tarea de encontrarlos y acceder a los mismos. Los sistemas de ficheros son usados en dispositivos de almacenamiento como discos duros y CD-ROM e involucran el mantenimiento de la localización física de los ficheros.

Más formalmente, un sistema de ficheros es un conjunto de tipo de datos abstractos que son implementados para el almacenamiento, la organización jerárquica, la manipulación, el acceso, el direccionamiento y la recuperación de datos.

La mayoría de las aplicaciones utilizan los ficheros como una forma de almacenamiento permanente de la información y como medio para compartir esta información con otras aplicaciones. El sistema de ficheros es la parte del sistema de E/S que se encarga de la gestión de los ficheros y proporciona un mecanismo de abstracción a los usuarios, de forma que oculta todos aquellos detalles relacionados con el almacenamiento y distribución de la información en los discos, así como el funcionamiento de los mismos.

Las tareas más importantes de un sistema de ficheros son:

- Organizar los dispositivos de almacenamiento del sistema.
- Controlar el conjunto de bloques libres de los dispositivos de almacenamiento.
- Permitir a los usuarios construir un espacio de nombres lógico.
- Mover los datos de forma eficiente entre los dispositivos de almacenamiento y la memoria.
- Coordinar el acceso múltiple a los recursos del sistema de ficheros.
- Ofrecer mecanismos de protección a los recursos del sistema del ficheros, como por ejemplo, privilegios sobre los recursos.
- Manejar la caché de los datos más usados.

Los sistemas de ficheros tradicionales o secuenciales son aquellos sistemas que se ejecutan en entornos monoprocesador y ofrecen sus servicios únicamente a los usuarios de dichos sistemas. Un ejemplo de sistema de este tipo lo constituye el sistema de ficheros de Unix (UFS: *Unix File System*). En Unix cada fichero se representa como una secuencia de bytes lineal y un puntero a partir del cual se realizan las operaciones de E/S. UFS proporciona un conjunto de operaciones básicas de acceso a ficheros, que consiguen acceder de forma contigua a los datos almacenados en discos locales. Esta interfaz no es adecuada para los requisitos de aplicaciones distribuidas y paralelas.

2.2.1. Sistemas de Ficheros Distribuidos

Con la aparición de las redes de interconexión, surgió la necesidad de compartir datos entre diferentes máquinas, problema que originó la creación de los sistemas de ficheros distribuidos. Los sistemas de ficheros distribuidos consiguen que procesos de múltiples computadores puedan acceder a un conjunto común de ficheros. Un sistema de este tipo garantiza la transparencia de acceso a los ficheros en un entorno distribuido, ofreciendo un espacio de almacenamiento global que permite a múltiples

clientes compartir los dispositivos de almacenamiento. Aunque un sistema de ficheros distribuido está formado por múltiples servidores y dispositivos de almacenamiento, el acceso a la información no se realiza en paralelo. **NFS** [15], **Coda** [16], **Ficus** [17], **Swift** [18], **xFS** [19], **AFS** [20], **Sprite** [21], **Coda** [22] **InterMezzo** [23, 24] son ejemplos de sistemas de ficheros distribuidos.

2.2.1.1. NFS

NFS (*Network File System*) fue diseñado originalmente por Sun Microsystems [25] en 1985 [15]. NFS permite el acceso a ficheros y directorios situados en máquinas remotas de forma transparente lo que hace posible utilizarlos como si fueran locales.

NFS sigue un esquema cliente-servidor. El nodo donde se almacenan los datos a ser accedidos se denomina *servidor NFS* y el que accede a los datos se *cliente NFS*. El servidor NFS sitúa directorios de su propio árbol de directorios a disposición de otros nodos que se encuentran conectados a través de una red. A la labor de poner a disposición un conjunto de directorios a un conjunto de nodos bajo una serie de opciones se denomina *exportar directorios*. Los clientes a fin de utilizar los directorios exportados previamente deben *montarlos* en algún directorio de su propio sistema de ficheros, que recibe el nombre el nombre de *punto de montaje*. Una vez montado el directorio, se accede a los ficheros y directorios de igual forma que a los ficheros locales. Lo habitual es que el administrador de la máquina cliente monte durante el proceso de arranque del sistema operativo, ya sea de forma automática o de forma manual, los puntos de montaje a utilizar durante el tiempo de funcionamiento del sistema.

Los clientes y servidores se comunican a través de la red mediante el uso de RPC (*Remote Procedure Call* [26]).

El cliente hace peticiones al servidor que incluyen toda la información necesaria para completar la operación, es decir, la operación es autocontenida. El protocolo NFS fue diseñado sin estado. Esta propiedad del servidor permite que sea más simple el proceso de recuperación ante un fallo, ya que si un cliente "cae" no afecta a las operaciones del servidor o de otros clientes. En el caso de que servidor falle, éste sólo necesita reiniciarse, los clientes sólo tienen que esperar hasta que el servidor vuelva a estar listo.

Junto con el protocolo NFS se proporcionan otros dos protocolos: NLM y NSM. El protocolo denominado NLM (*Network Lock Manager*) [27], permite gestionar cerrojos sobre los ficheros. El protocolo NSM (*Network Status Monitor*) [28] consigue monitorizar la caída abrupta de un servidor NFS.

NFS de Sun Microsystems se transformó en el estándar para compartir ficheros bajo plataformas UNIX. Actualmente existen implementaciones de NFS para otros sistemas operativos como Windows. Linux dispone de dos implementaciones diferentes del servidor NFS: un servidor NFS en espacio de usuario (ya prácticamente sin mantenimiento) y otro servidor NFS en el núcleo del sistema operativo [29].

Algunos sistemas también emplean servidores estándar como base de su fun-

cionamiento, principalmente NFS. El sistema Bigfoot-NFS[30], por ejemplo, también combina varios servidores NFS, sin embargo, utiliza el fichero como unidad de reparto, de manera que, todos los datos de un mismo fichero se almacenan en un único servidor. Aunque los ficheros y los directorios se distribuyen por varios servidores, no se ofrece acceso paralelo a los datos de un mismo fichero. Otro sistema similar es Slice[31]. Slice es un sistema de almacenamiento que utiliza un filtro de paquetes, denominado μproxy, que virtualiza a los servidores NFS presentando un volumen único de ficheros compartido. El sistema utiliza μproxy para distribuir las peticiones de los clientes sobre varios servidores. El principal problema de este sistema es que el filtro μproxy puede convertirse en un cuello de botella que afecte a la escalabilidad del sistema.

El sistema dNFSP [32] es un sistema similar a PVFS [11], se basa en un conjunto de servidores de datos y un servidor de metadatos que actúa como un servidor NFS entre los clientes y los servidores de datos. Todas las operaciones pasan por el servidor de metadatos que redirige las peticiones a los servidores de datos correspondientes. Para las operaciones de lectura los servidores de datos responden directamente a los clientes. En el caso de las operaciones de escritura todas pasan por el servidor de metadatos. Con el fin de evitar que éste sea un cuello de botella, se replica el servidor de metadatos distribuyendo la carga de los clientes sobre ellos. La coherencia de la información de los servidores de metadatos se mantiene mediante un protocolo relajado.

Por su parte, el sistema pNFS [33] constituye una extensión de NFS4 que permite a los clientes acceso directo a los dispositivos de almacenamiento. Se trata de un sistema similar a GPFS [10] pero ofreciendo el protocolo NFS4.

2.2.1.2. GFS

GFS (*Global File System*) [34] es un sistema distribuido que permite a múltiples nodos acceder y compartir discos y dispositivos de cintas en una red de almacenamiento. GFS distribuye las responsabilidades del sistema de ficheros a través de los nodos de proceso, el almacenamiento a través de los dispositivos y los recursos del sistema de fichero a través de todos los nodos.

La primera versión de GFS se conoció como GFS-1 y apareció en el verano de 1995. En dicha versión, se utilizó tecnología Fibre Channel a fin de procesar grandes datasets científicos en equipos Silicon Graphics. GFS fue por tanto implementado en el sistema operativo IRIX de esta compañía.

Esta implementación utilizaba discos SCSI paralelos y mandatos para llevar a cabo la sincronización, esto bloqueaba completamente los dispositivos, haciendo por tanto imposible acceder de forma simultánea a la metainformación. Este cuello de botella fue resuelto en la segunda versión, denominada GFS-2, a través de la implementación de grano fino de los cerrojos, que no bloqueaban todo el dispositivo. Esta solución no era escalable, debido a la contención de la red. Además, requería el uso de grandes ficheros para obtener un buen rendimiento, debido a que no había caché en los clientes ni para la metainformación ni para los datos de los ficheros. En

1998 se empezó a portar el código al sistema operativo Linux, integrando GFS en el kernel. De esta forma, se logró construir un sistema de ficheros de propósito general escalable, que permite compartir dispositivos a través del uso de una red heterogénea y se posibilita el almacenamiento temporal de metainformación y ficheros (*caching*). Esta versión se denominó GFS-3.

2.2.1.3. SMB

Server Message Block es un protocolo de red (que pertenece a la capa de aplicación en el modelo OSI) que permite compartir ficheros, impresoras, y demás recursos entre nodos de una red. Se usa principalmente en computadoras con Windows y DOS.

Fue desarrollado por IBM, pero el que actualmente se usa es la versión modificada por Microsoft, que lo renombró a CIFS (Common Internet File System) en 1998. Entonces se incluyó el soporte para enlaces simbólicos y duros y la posibilidad de mayores tamaños de fichero.

Para las comunicaciones utiliza un protocolo de servicios propio denominado NetBIOS.

Existe una versión libre equivalente para sistemas operativos basados en UNIX/LINUX denominado Samba [35].

2.2.1.4. Google File System

El sistema de fichero Google File System (GFS) es un sistema de almacenamiento basado en las necesidades de Google [36], por lo que no es un sistema de fichero de propósito general. Los objetivos de su diseño son:

- Gran tolerancia a fallos.
- Administración de grandes volúmenes de información (del orden de GB).
- Soporte ante la modificación de ficheros. Esta modificación se considera que sólo se puede producir debido a la adición de nuevos datos, ya que se considera que rara vez se produce la modificación de los datos. Aun así, se permite la modificación aunque no se realice eficientemente.
- La interfaz proporcionada por el sistema de ficheros debe estar muy integrada con las aplicaciones que hacen uso del sistema.
- El sistema debe facilitar el acceso concurrente a los datos.
- El acceso a los datos debe optimizarse para el caso de uso más frecuente: grandes lecturas de datos y pequeñas lecturas aleatorias.
- El sistema debe proporcionar un gran ancho de banda en el acceso a los datos.

La arquitectura se compone de un servidor maestro, que es el encargado de mantener todos los metadatos del sistema de ficheros y de múltiples servidores de datos. Los datos se encuentran divididos en bloques de tamaño fijo identificados por un único número (global e inmutable) de 64 bits, denominado *chunk handle*. Este identificador es asignado por el servidor maestro cuando se almacena el bloque de datos del fichero. Para mantener la fiabilidad, cada bloque está replicado en distintos servidores de datos. Estos servidores de datos se encuentran ejecutando en el espacio de usuario lo que facilita su uso en entornos Linux. No existe ningún tipo de caché de datos en el cliente.

El nodo maestro gestiona metadatos como el espacio de nombres, la información para el control de acceso, el mapa con la distribución de ficheros y la localización de los bloques de datos. Contar solo con un servidor maestro que tiene un conocimiento global del sistema simplifica el diseño y permite tener sofisticados métodos de emplazamiento y replicación de los bloques de datos. Con este diseño se pretende minimizar las lecturas y escrituras para evitar los cuellos de botella. El servidor maestro redirige los accesos a los datos que se encuentran almacenados en los servidores de datos y los distribuye a los clientes a través de los distintos servidores que dispongan de los datos solicitados.

El servidor maestro no almacena un registro persistente de localización de réplicas, sino que los servidores de datos indican esta información al iniciarse. Además, puede actualizar su información mediante la gestión y la monitorización de los servidores de datos.

Otras funciones del servidor maestro son la recolección de basura, la replicación en caso de fallos en los servidores de datos y la migración de bloques para el equilibrio de carga.

El tamaño usado por parte de los bloques de datos en los que se dividen los ficheros es de 64 MB. Este tamaño de bloque presenta varias ventajas:

- Reduce las interacciones del cliente con el servidor maestro. Las lecturas y escrituras en el mismo trozo sólo requieren una petición inicial al servidor maestro para obtener la localización del trozo.

- Al disponer de un tamaño de bloque grande, es muy probable que se puedan realizar muchas operaciones en un trozo. Esto reduce la sobrecarga en la red y permite el mantenimiento una conexión TCP persistente con el servidor de datos durante un periodo de tiempo.

- Se reduce la cantidad de metadatos almacenados en el servidor maestro debido a que el tamaño de bloque disminuye el número total de bloques del sistema. Esto permite almacenar los metadatos en memoria.

El mayor problema que presenta el uso de un tamaño de bloque de datos tan grande es la posibilidad de fragmentación interna de los datos.

2.2.2. Sistemas de ficheros paralelos

En la arquitectura genérica de un sistema de E/S distribuido-paralelo pueden distinguirse diferentes niveles funcionales [37]:

- Nodo de cómputo: realiza las peticiones de E/S. También se denomina cliente.
- Nodo de E/S: ejecuta las operaciones de E/S sobre ficheros y dispositivos lógicos. Se le denomina también servidor.
- Controlador: transfiere bloques físicos desde/hacia los dispositivos físicos.
- Dispositivo: almacena los datos en bloques físicos.

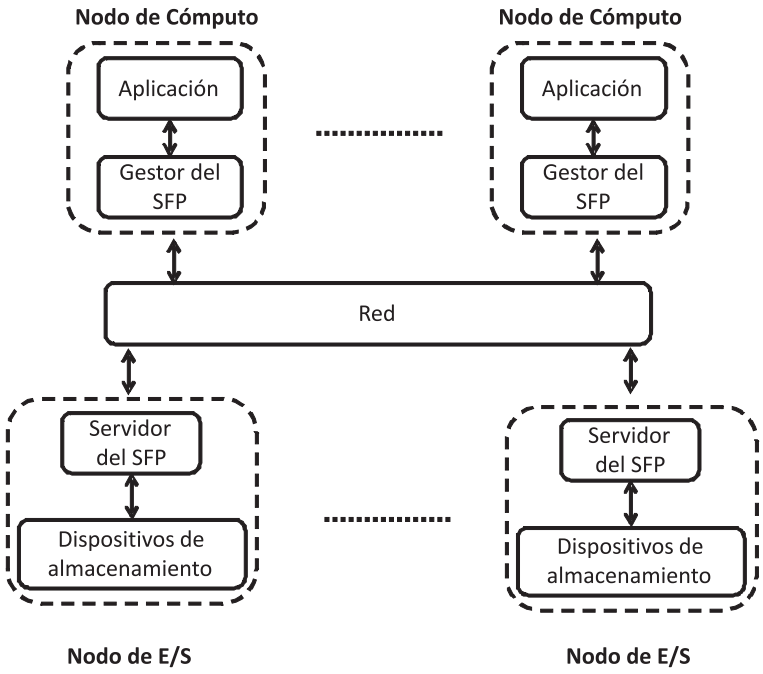

Figura 2.3: Arquitectura genérica de un Sistema de Ficheros Paralelo.

La arquitectura tradicional de un sistema de ficheros paralelo se puede describir partiendo de la definición anterior y de forma genérica como se muestra en la Figura 2.3. Dicha arquitectura está formada por un conjunto de clientes (que son los nodos de proceso) y un conjunto de servidores interconectados por una red. Estos servidores son nodos de entrada y salida a los que se conectan diferentes dispositivos a través de distintos controladores.

Los servidores trabajan conjuntamente para ofrecer a los clientes la visión de un único sistema de almacenamiento [38] uniendo la capacidad de almacenamiento que aporta cada uno. Los clientes demandan, a través del conjunto de servidores, operaciones de almacenamiento y recuperación de datos en el espacio de almacenamiento ofrecido por los servidores. Al contrario que los servidores, a priori, un cliente no tienen conocimiento de la existencia de los demás. Cada cliente opera de

forma independiente, aun cuando sean diferentes procesos de una misma aplicación paralela.

Los servidores, además de gestionar el sistema de almacenamiento ofrecido a los clientes, han de asegurar la consistencia de los datos cuando múltiples clientes acceden a una misma porción de datos almacenados.

Actualmente hay numerosas soluciones que utilizan la E/S paralela. Algunas de estas soluciones usadas para entornos comerciales son: Lustre [39], Panasas [40], GPFS [10] e IBRIX Fusion [41]. Estos sistemas de ficheros se utilizan en un alto porcentaje en los *clusters* que aparecen en el Top 500 [3].

Algunos sistemas de ficheros usados en entornos académicos para investigación son: PVFS (versión 1 y 2) [11], Clusterfile [42], Galley [43], PPFS [44], Scotch [45], Vesta [46, 47], Parfisys [48, 49], PIOUS [50], Armada [51], etc.

Esta sección empieza describiendo algunos de los sistema de ficheros paralelos más extendidos en la actualidad.

2.2.2.1. Lustre

Lustre es un sistema de ficheros paralelo de código abierto para el entorno Linux desarrollado por la compañía Cluster File Systems Inc. y HP. Se encuentra ámpliamente implantado en los laboratorios nacionales de Estados Unidos de Norte America, entre los que se incluyen: el laboratorio nacional Lawrence Livermore (LLNL), el laboratorio nacional Pacific Northwest (PNNL), los laboratorios nacionales Sandia (SNL), la administración nacional de seguridad nuclear (NNSA), el laboratorio nacional de Los Alamos National (LANL), y el NCSA.

Lustre es un sistema de ficheros paralelo orientado a objetos. Estos objetos encapsulan los datos de los usuarios como por ejemplo los atributos de los datos. Lustre mantiene i-nodos únicos para cada uno de los recursos del sistema de ficheros como ficheros, directorios, enlaces simbólicos, etc.

Los metadatos se almacenan en los servidores de metadatos denominados MDS (*Metadata Server*), usando un sistema de equilibrio de carga. Además, los datos almacenados en los servidores de metadatos se encuentran replicados para soportar cierta tolerancia a fallos, y se lleva a cabo un registro de las operaciones que conlleven cambios en el sistema de ficheros.

Por otra parte, los datos se reparten entre los distintos servidores de datos del sistema. Estos servidores de datos se denominan OST (*Object Storage Target*), y almacenan cada uno de los objetos del sistema. Además, se encargan de la gestión de la E/S de los distintos clientes, una vez que conocen qué servidores contienen los objetos necesarios para realizar la operación de E/S.

Lustre mantiene una semántica fuerte de los datos de los ficheros, mediante un sistema de bloqueos que gestiona de forma independiente cada uno de los servidores de datos. Por último añadir que Lustre mantiene un control completo de los recursos de almacenamiento denominados en Lustre como OBD (*Object-Based Disk*), mediante un soporte directo a los sistemas de ficheros de tipo *journaling* o de registro

como ext3, ReiserFS o XFS.

2.2.2.2. GPFS

El sistema de ficheros GPFS (*General Parallel File System*) [10, 52] ha sido desarrollado por IBM como sistema de ficheros paralelo de disco compartido. Actualmente funciona en entornos AIX y Linux, y se encuentra muy extendido en entornos de cómputo de altas prestaciones como por ejemplo el *cluster* MareNostrum [6] del Barcelona Supercomputing Center(BSC) o el supercomputador ASCI Purple [7] del instituto nacional Lawrence Livermore (LLNL).

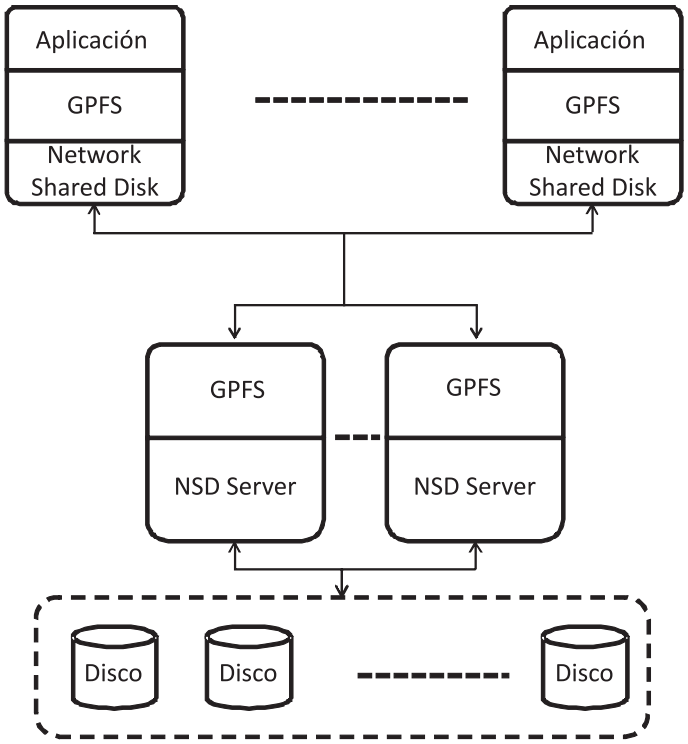

Figura 2.4: Arquitectura de GPFS usando servidores de datos.

En GPFS, podemos encontrar dos tipos de arquitecturas de E/S bien diferenciadas. En la primera los nodos de cómputo están conectados a los nodos de E/S del sistema, los cuales a su vez se conectan a los discos compartidos a través de tecnologías de comunicación de datos como Fibre Channel o iSCSI, tal y como se puede apreciar en la Figura 2.4. En el otro tipo de arquitectura de almacenamiento, como se puede ver en la Figura 2.5, todos los nodos de cómputo realizan funciones de nodos de E/S, y usan directamente los dispositivos de almacenamiento mediante las tecnologías de comunicación anteriormente comentadas.

La arquitectura de GPFS usa un sistema de bloqueo de datos distribuido que garantiza una semántica POSIX en el sistema. Los bloqueos se pueden realizar sobre regiones concretas de los datos lo que permite un mayor control de los mismos. GPFS por defecto mantiene una caché distribuida en los nodos de cómputo, aunque también

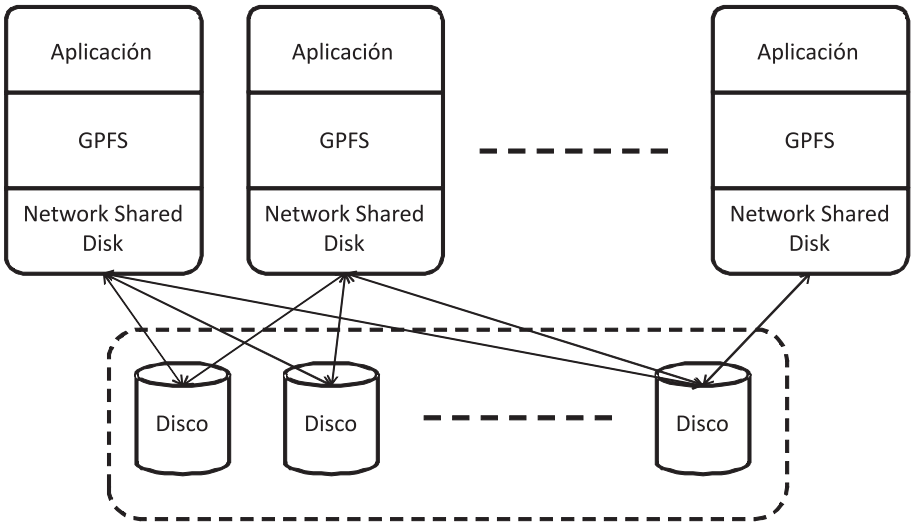

Figura 2.5: Arquitectura de GPFS usando discos de forma directa.

soporta un modo degradado de gestión de los datos denominado *data shipping*, donde la caché sólo se mantiene en el lado del servidor. Este método se utiliza en entornos donde para mantener la coherencia de los datos son accedidos por diversos clientes a la vez sobre un mismo bloque, con tamaños menores al tamaño de bloque.

Existe una implementación específica de MPI-IO [53] desarrollada con estas modificaciones.

2.2.2.3. PVFS

Sistema de ficheros paralelo desarrollado por la universidad de Clemson (USA) para entornos Linux. Se distribuye como software libre y no requiere de ningún hardware especial para que funcione. Su uso se encuentra muy extendido en entornos académicos.

Se basa en la distribución de los datos a lo largo de distintos nodos de E/S. Para conseguir un alto rendimiento o un gran ancho de banda en las operaciones de lectura y escritura concurrentes, PVFS distribuye los datos en múltiples nodos del cluster, denominados nodos de E/S. De manera que los clientes poseen diferentes rutas hacia los datos, se mejora el ancho de banda para múltiples clientes, y se eliminan los cuellos de botella.

Actualmente existen dos versiones del mismo con grandes diferencias de diseño. En la versión 1 hay un único servidor centralizado de metadatos (ver Figura 2.6). En la versión 2 es posible incluir múltiples servidores de metadatos, donde se distribuyen los mismos (ver Figura 2.7). Además, en esta versión se pueden incorporar distintas políticas de reparto, que permiten añadir patrones fácilmente y nuevos componentes al sistema, gracias a su novedosa estructura modular.

Para el acceso a los datos se puede usar tanto la interfaz POSIX como la nativa del sistema. Para usar esta última es necesario incorporar la biblioteca del sistema

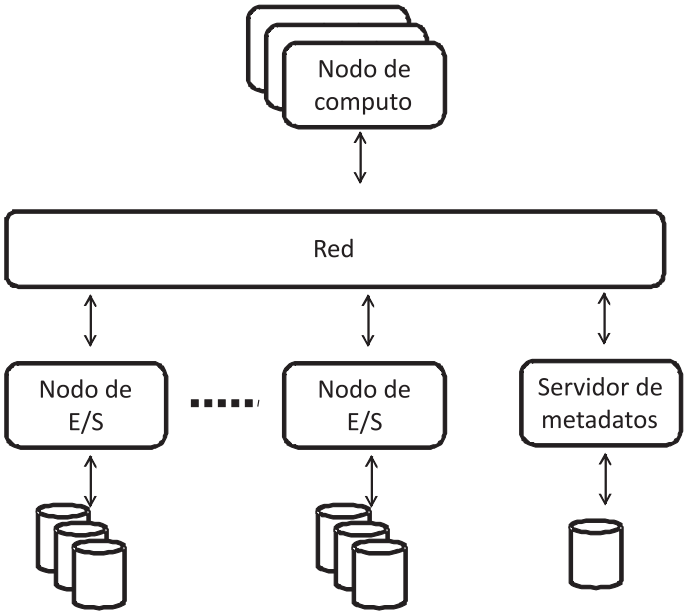

Figura 2.6: Arquitectura de PVFS versión 1.

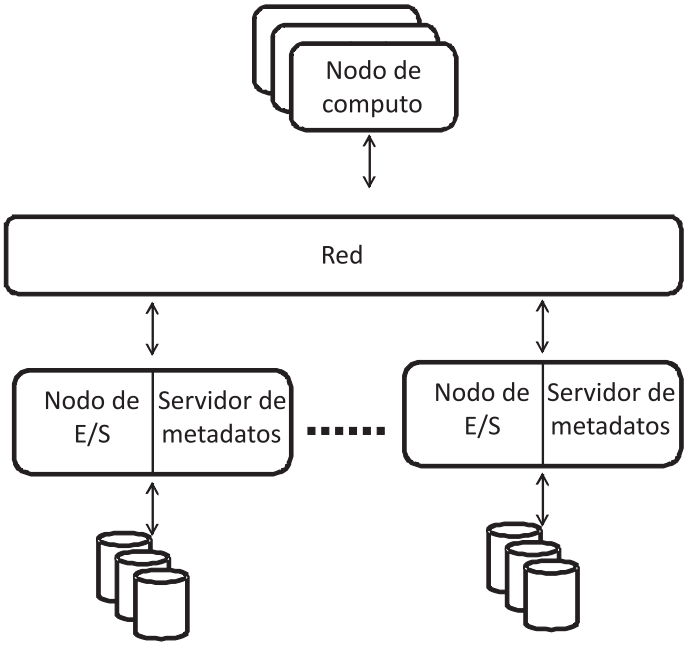

Figura 2.7: Arquitectura de PVFS versión 2.

de ficheros paralelo en la aplicación que haga uso de éste. Este es el caso de la integración con MPI-IO donde además se han desarrollado técnicas de optimización que serán detalladas en secciones posteriores.

2.3. Técnicas de optimización en la E/S

Esta sección introduce las técnicas usadas para la optimización de la E/S. Se divide en dos partes: la primera trata técnicas genéricas utilizadas en los sistemas de ficheros, como las de almacenar los datos en memoria (*caching*) y la de lectura adelantada (*prefetching*), y en la segunda detalla las técnicas de optimización de la E/S paralela.

2.3.1. *Caching* y *prefetching* de la E/S

El empleo de una caché o almacenamiento intermedio en un sistema de E/S permite beneficiarse de la localidad temporal y espacial de los datos, y de un acceso más rápido a los mismos, lo que alivia el problema de la crisis de la E/S [54]. Una caché almacena una copia de los datos recientemente utilizados en un dispositivo de almacenamiento más rápido que aquél en el que se encuentran los mismos. El uso de caché en el sistema mejora su rendimiento ya que reduce el número de accesos a los dispositivos, la carga en los nodos de E/S y la contención en la red de interconexión al poder acceder a una gran parte de los datos de forma local.

Respecto al tipo de datos que almacena la caché en el sistema, se distinguen dos: datos correspondientes a bloques del fichero, caché de bloques, y metadatos correspondientes a la información que necesita utilizar el sistema de ficheros para la gestión de los mismos, caché de metadatos.

La caché de bloques mejora el rendimiento de tres formas:

1.- Utiliza el principio de proximidad de referencias en los accesos a un fichero, tanto temporal como espacial. En un sistema de ficheros paralelo la proximidad temporal sobre un fichero es una situación poco frecuente, debido a que predomina el acceso de tipo secuencial. Sin embargo, se da una alta proximidad espacial, debido al gran número de peticiones de E/S de tamaños pequeños [55, 56].

2.- Realiza la lectura adelantada (*prefetching*) de bloques de datos antes de que sean solicitados por las aplicaciones. Esto mejora el rendimiento de las operaciones de lectura, sobre todo en el caso de uso de patrones de tipo secuencial.

3.- Permite el uso de políticas de escritura diferida, lo que retrasa la escritura de los datos modificados en los dispositivos de almacenamiento secundario. De este modo, se mejora el rendimiento de las operaciones de escritura.

Los parámetros para el diseño de la caché de bloques en sistemas de ficheros paralelos y distribuidos [57] son:

- **Localización de la caché:** se puede situar en los nodos de E/S, pero eso supone el uso de un esquema centralizado y, por tanto, no escalable, ya que no reduce el acceso a recursos compartidos, tales como la red, la caché de

los nodos de E/S y los dispositivos. No obstante, resuelve el problema de la coherencia de caché.

Por otra parte, existe la posibilidad de ubicarla en los nodos de cómputo. Esto supone una solución más escalable, porque disminuye la contención en la red de interconexión, en los nodos de E/S y en los dispositivos, ofreciendo un mayor paralelismo en el acceso a los datos, pero complica la coherencia de la caché.

- **Granularidad de la caché:** se refiere al tamaño de los datos almacenados en la misma. El aumento de tamaño mejora la probabilidad de aciertos en accesos posteriores y disminuye la utilización de la red de interconexión, pero trae consigo un problema, el incremento de la latencia en las operaciones de E/S. Por esta razón, no es adecuado para aplicaciones paralelas que accedan a datos no contiguos en el fichero.

- **Tamaño de la caché:** viene determinado por los patrones de E/S utilizados e incide en la tasa de aciertos de dicha caché. Para patrones secuenciales, basta una caché de tamaño pequeño, ya que la reutilización de bloques es muy pequeña. Para otros, puede ser necesaria la utilización de una de mayor tamaño. Es necesario conocer el comportamiento de las diferentes aplicaciones que utilicen el sistema de ficheros para calcular el tamaño óptimo de la caché y de esta manera mejorar la relación coste-rendimiento.

- **Política de reemplazo:** es la utilizada para decidir qué bloques se eliminan de la caché cuando se llena. Hay diferentes políticas, entre las que destacan FIFO (*First In First Out*), aleatoria, LRU (*Least Recently Used*), MRU (*Most Recently Used*), etc. Entre los factores que determinan la elección de una u otra se encuentran los patrones de acceso a ficheros, las características físicas de los discos utilizados, la arquitectura multiprocesador o la prioridades.

- **Política de actualización:** determina cómo y cuándo volcar los bloques modificados al disco. Existen varios tipos:

  - **Escritura inmediata (*write-through*):** los bloques son escritos a disco en cuanto alguna aplicación los modifica.
  - **Escritura diferida (*write-back*):** los datos se mantienen en la caché hasta que se requieren bloques libres por la demanda de nuevas peticiones. Variantes de esta política son: *write-on-close*, que vuelca todos los bloques modificados a disco cuando se cierra el fichero y *write-full*, que retrasa la escritura a disco hasta que el *buffer* de la caché se llena.

    Reduce la latencia y permite eliminar las escrituras a disco de ficheros temporales de corta duración. No obstante, introduce un problema de fiabilidad en el sistema de ficheros.

- **Lectura adelantada:** también llamada *prefetching*. Consiste en la lectura por adelantado de bloques antes de que sean solicitados por las aplicaciones. De

este modo, se puede lograr solapar el tiempo de E/S con el tiempo de cómputo de las aplicaciones.

La caché de metadatos almacena la información del sistema de ficheros y la de cada fichero en particular (nombre, atributos e información de direccionamiento del mismo).

La metainformación puede ser persistente, si se localiza en disco (ej: bloques de datos), o transitoria, si existen únicamente cuando el fichero está siendo accedido (ej: puntero de posición). Por otra parte, algunos metadatos son visibles a las aplicaciones (ej: longitud de un fichero) y otros ocultos (ej: distribución del disco).

Las principales cachés de metadatos que proporciona un sistema de ficheros son:

- **Caché de particiones**, permite un rápido acceso a la información que describe cada partición. Ejemplo: superbloque de UNIX.
- **Caché de atributos de ficheros**, almacena los atributos de aquellos ficheros que están siendo utilizados. Ejemplo: i-nodo de UNIX y tabla de i-nodos.
- **Caché de nombres**, optimiza la decodificación de nombres en un sistema de ficheros.
- **Caché de información de direccionamiento**, almacena información que permite acceder a los bloques que forman el fichero. Ej: i-nodo en UNIX, que se complementa con el uso de bloques indirectos.

Las técnicas de *caching* y *prefetching* de ficheros se han estandarizado en el campo de los sistemas de ficheros secuenciales y distribuidos [58, 59, 60]. En este tipo de sistemas, la técnica de *prefetching* más comúnmente utilizada consiste en leer secuencialmente el fichero de forma adelantada. Esto implica detectar si una aplicación accede de forma secuencial sobre un fichero y en caso afirmativo, leer los bloques de datos secuencialmente [61, 62].

Muchos sistemas de ficheros distribuidos proporcionan dos niveles de caché, algunos de ellos son Sun NFS, Sprite, AFS o CODA.

Por otro lado, las técnicas de *caching* y *prefetching* también se han implantado en los sistemas de ficheros paralelos con el fin de optimizar el rendimiento de las operaciones de E/S [63, 64, 65].

Existen algunos trabajos que utilizan compresión de datos para el diseño de estrategias de *prefetching* óptimas [66, 67, 68]. Otros trabajos definen estrategias de *prefetching* a través de un modelo probabilístico de las secuencias de peticiones [69].

En [64] se describe un modelo teórico que permite caracterizar un sistema, con el fin de aplicar las técnicas de *caching* y *prefetching*, mientras que en [70] se estudia la combinación de éstas.

2.3.2. Técnicas para la mejora de la E/S paralela

En esta sección describiremos algunas de las técnicas más utilizadas actualmente para la mejora del rendimiento en sistemas de ficheros. Se pueden dividir en tres categorías dependiendo del tipo de accesos que la aplicación realiza: E/S no contigua (*Noncontiguous I/O*), E/S colectiva (*Collective I/O*) o caché cooperativa (*cooperative caching*).

2.3.2.1. *Noncontiguous I/O*

Numerosas aplicaciones científicas utilizan grandes conjuntos multidimensionales de datos que deben ser almacenados a lo largo de su ejecución. Típicamente este almacenamiento de los datos se usa para visualización, captura de los datos en un momento dado, *checkpointing*, *out-of-core computation*, post-proceso de datos [71, 72, 73], etc. Entre estas aplicaciones hay algunas cuyos patrones de acceso son no contiguos, como por ejemplo las aplicaciones IPARS [74] y FLASH [75]. El acceso a los datos mediante la interfaz básica POSIX provoca que los datos no contiguos tengan que ser accedidos independientemente, lo que ocasiona múltiples accesos pequeños al sistema de almacenamiento. Esto reduce el rendimiento de las operaciones de E/S.

Se han propuesto múltiples soluciones que mejoran ostensiblemente el rendimiento de las aplicaciones que realizan este tipo de accesos. A continuación se detallarán las técnicas denominadas *list I/O*, *datatype I/O* y *data sieving*, que son las más utilizadas para este tipo de patrones de acceso.

List I/O Es una interfaz diseñada para soportar los accesos no contiguos a los datos. Permite la descripción de datos no contiguos tanto en memoria como en disco mediante el uso de una lista de pares compuestos por la posición inicial del dato y el tamaño solicitados. De esta manera es posible realizar una única operación de E/S en el sistema de ficheros, como se puede ver en la Figura 2.8. Con esta técnica se puede describir una operación de E/S en una única llamada y de esta manera optimizar y reducir el número de operaciones de E/S necesarias para llevar a cabo la operación solicitada.

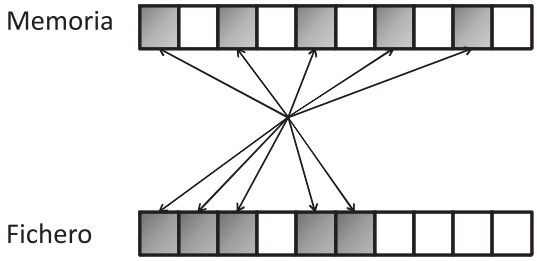

Figura 2.8: *List I/O*.

Sin embargo, esta técnica necesita de la creación y procesamiento de largas lis-

tas de peticiones, además de la transmisión de las mismas a los servidores de datos que dan soporte al sistema de ficheros paralelo. Los tamaños descritos en las listas de peticiones deben ser acordes a los tamaños de transmisión de la red. Así, es conveniente usar esta técnica cuando el número de elementos de la lista de peticiones no es muy elevado y los accesos tienen un componente irregular en su comportamiento (otras técnicas como *Datatype I/O* describen mejor el comportamiento regular del acceso a los datos).

List I/O se encuentra implementada en la interfaz MPI-IO. Algunos sistemas de ficheros paralelos, como PVFS [76], dan soporte a esta interfaz mediante la incorporación de nuevas llamadas a la interfaz de E/S.

Datatype I/O Se utiliza para patrones de acceso de tipo regular, donde el tamaño y el desplazamiento de los datos solicitados se pueden describir mediante un tipo de datos especial (que es una adaptación de los tipos de datos intercambiados entre las aplicaciones MPI) que indica de forma unívoca el patrón de acceso a los datos. Este tipo de patrones de acceso son frecuentemente utilizados en las aplicaciones científicas (como por ejemplo, en el acceso a los elementos de una matriz).

El tipo de datos utilizado para describir el patrón de acceso a los datos se obtiene de una lista de peticiones de menor tamaño que describe el número de datos consecutivos, el desplazamiento inicial de los datos, y el desplazamiento existente entre ellos. Esto reduce el número de peticiones que se tienen que comunicar al servidor.

Una versión de esta técnica se encuentra implementada usando PVFS [77] en una implementación de MPI-IO denominada ROMIO [78].

Data Sieving La mayoría de los sistemas de almacenamiento (discos duros, etc) se comportan mejor cuánto mayor sea la cantidad de datos distribuida físicamente de forma continua.Teniendo en cuenta esto, la técnica transforma múltiples accesos pequeños en un menor número de ellos de mayor tamaño [79], mediante el uso de un espacio de almacenamiento en memoria temporal. Este espacio almacena los datos involucrados en la lectura o escritura. Así, los datos se extraen o se sobrescriben en el *buffer* utilizado, para mas tarde sustituir, en caso de una escritura, los datos almacenados en el fichero (véase Figura 2.9).

Mediante esta técnica se pretende reducir el número de accesos a los sistemas de almacenamiento, de manera que se trabaje el mayor tiempo posible con los datos replicados que se encuentran en memoria. Así, las variables a tener en cuenta para utilizar esta técnica son: el reparto de los datos solicitados (es decir, si se encuentran concentrados o poco esparcidos), la cantidad, y por último, el tamaño utilizado en memoria para almacenar estos datos de forma temporal.

Esta técnica presenta varios problemas:

- La utilización masiva de la memoria del computador, ya que es necesaria una mayor cantidad de datos en memoria para minimizar el número de posibles

accesos al sistema de almacenamiento.

- Sólo es eficiente si los datos con los que se trabaja se encuentran en el *buffer* temporal, puesto que si los datos solicitados se encuentran muy dispersos, es muy posible que no estén disponibles en el *buffer* temporal, por lo que el número de accesos al sistema de almacenamiento aumenta.

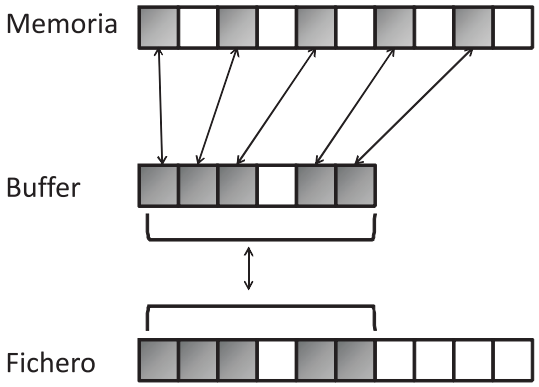

Figura 2.9: *Data sieving.*

Por otro lado, esta técnica requiere de cierto control de concurrencia. Un proceso no puede modificar un dato mientras exista algún proceso lector accediendo a éste, puesto que podría originar una inconsistencia.

2.3.2.2. *Collective I/O*

Tal y como se ha visto en los apartados anteriores, las aplicaciones paralelas acceden frecuentemente a multitud de datos no contiguos de pequeño tamaño. Las técnicas que se han presentado hasta ahora se realizan de forma individual. A veces, las aplicaciones realizan operaciones colectivas, cuando esto ocurre, las llamadas colectivas de E/S tienen como cometido la fusión de múltiples peticiones individuales en una gran petición, de manera que se optimice la E/S del conjunto de aplicaciones involucradas. Esta técnica de fusión tiene distintos comportamientos dependiendo de dónde se realice esta unión de peticiones:

- *Two-phase I/O* [80, 81]: el conjunto de peticiones se fusionan en los nodos de cómputo o en nodos intermedios dedicados a tal efecto.

- *Server-directed I/O* [82, 83]: la mezcla de las peticiones se realiza en los nodos de E/S. Cuando las peticiones se agrupan en la controladora del disco (o discos), esta técnica se denomina *Disk-directed I/O*.

Two Phase I/O La Figura 2.10 ilustra el método de fusión realizado por parte de las aplicaciones mediante llamadas colectivas[84]. Esta técnica identifica a unos

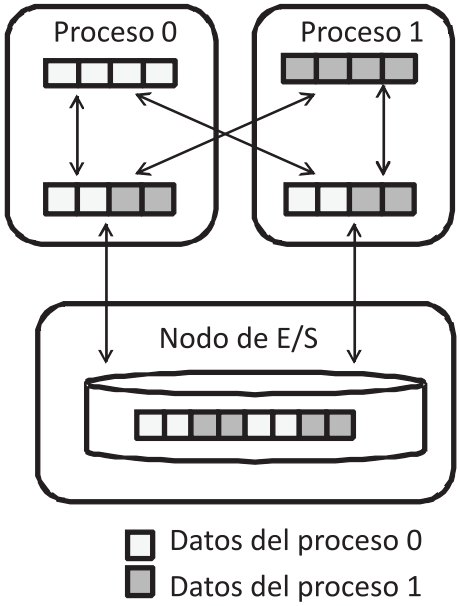

Figura 2.10: *Two-phase I/O.*

procesos de la aplicación paralela como procesos agregadores. Estos se caracterizan por ser los responsables del manejo de una porción de los datos.

De forma similar a la técnica *data sieving*, los datos se almacenan en un *buffer* temporal, si bien a diferencia de la técnica anterior, este *buffer* temporal se encuentra distribuido a lo largo de los procesos agregadores.

La gran ventaja de esta técnica de E/S es la agregación de múltiples accesos pequeños y no contiguos sobre los datos, fusionándolos en un conjunto pequeño de accesos de gran tamaño. Uno de los problemas que acarrea el uso de esta técnica es la necesidad de una sincronización de todos los procesos involucrados, lo que significa que es una técnica que penaliza a aquellos procesos que tienen una carga de trabajo menor.

Por otra parte, si la red sobre la que se comunican los distintos procesos no es suficientemente rápida (con baja latencia y gran ancho de banda), el movimiento de los datos entre los procesos agregadores puede llevar a una pérdida significativa del rendimiento, sobre todo en comparación con otro tipo de técnicas no colectivas como *list I/O* o *datatype I/O*.

Server-directed I/O* y *Disk-directed I/O La implementación de *server-directed I/O* se realiza en el servidor de almacenamiento y la de *disk-directed I/O* en los discos. La mayor diferencia entre ambas técnicas es el nivel de abstracción sobre el cual se trabaje: por un lado *disk-directed I/O* maneja datos a nivel de bloque, y por el otro *server-directed I/O* utiliza la abstracción de fichero sustentada por el sistema de ficheros local al servidor de E/S.

La Figura 2.11 muestra un ejemplo de lectura mediante el uso de la técnica *disk-directed I/O*. Los nodos de cómputo envían las peticiones directamente a los

nodos de E/S. Estos mezclan y ordenan las peticiones, transfiriéndolas ordenadas a los dispositivos de E/S (por ejemplo discos). Los datos obtenidos son almacenados en *buffers* temporales y finalmente enviados a los nodos de cómputo.

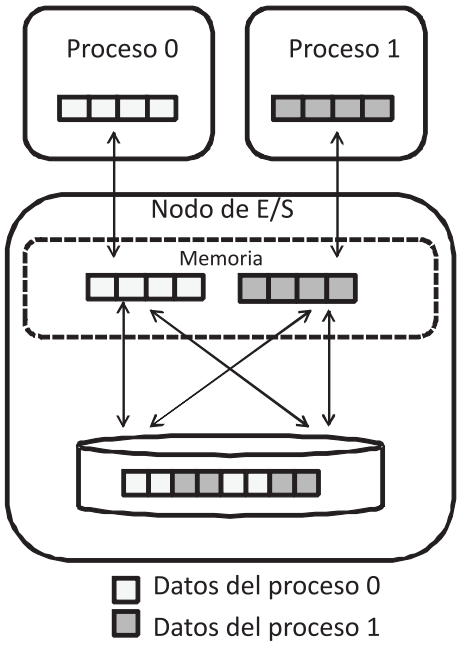

Figura 2.11: *Disk-Directed I/O*.

La técnica *server-directed I/O* se encuentra implementada en la biblioteca de E/S Panda [83]. En ésta, los nodos de cómputo envían al servidor maestro de E/S una descripción de los datos solicitados.

Ambas técnicas tienen ventajas sobre *two-phase I/O*. Las más destacables son: los datos son sólo enviados una vez y no es necesario el uso de memoria adicional por parte de los nodos de cómputo. El mayor problema sigue siendo el manejo de numerosas peticiones pequeñas que pueden llegar a colapsar al sistema de E/S, reduciendo considerablemente el rendimiento.

2.3.2.3. *Cooperative caching*

En un entorno de computación de alto rendimiento, cada nodo de cómputo utiliza parte de la memoria como caché para almacenar los bloques de datos de los ficheros manejados por las aplicaciones. La caché de estos sistemas está gestionada por el sistema de ficheros del nodo de cómputo. Cuando la caché se encuentra llena, algunos bloques modificados por las aplicaciones, son enviados al disco.

Esta técnica se basa en el uso de los recursos potenciales de los nodos de cómputo (como son la memoria o el sistema de almacenamiento local) para almacenar bloques de datos, estableciendo una caché virtual de almacenamiento. De esta manera, se pueden establecer políticas de reemplazo y de escritura de datos.

Así por ejemplo, el sistema de ficheros paralelo PAFS [85, 86], establece una

arquitectura de caché mediante tres componentes: servidores de disco, caché y uno de reparto de datos. En este caso, los datos se distribuyen entre los distintos servidores de discos, mientras que los servidores-caché se encargan de la gestión de los datos entre los clientes y los servidores de datos. Por último, el servidor de reparto gestiona la distribución de los datos.

En este PAFS, cada servidor-caché se encarga de manejar un conjunto de ficheros definido, de tal manera que, todas las peticiones relacionadas con el mismo fichero serán redirigidas al servidor-caché correspondiente. De esta manera no es necesario ningún tipo de política de coherencia de caché, lo que mejora el rendimiento del sistema. Sin embargo, aunque los servidores-caché tienen distribuidos los datos físicamente a través de los nodos de almacenamiento dispuestos, pueden llegar a ser un cuello de botella si varias aplicaciones solicitan datos de un mismo servidor-caché.

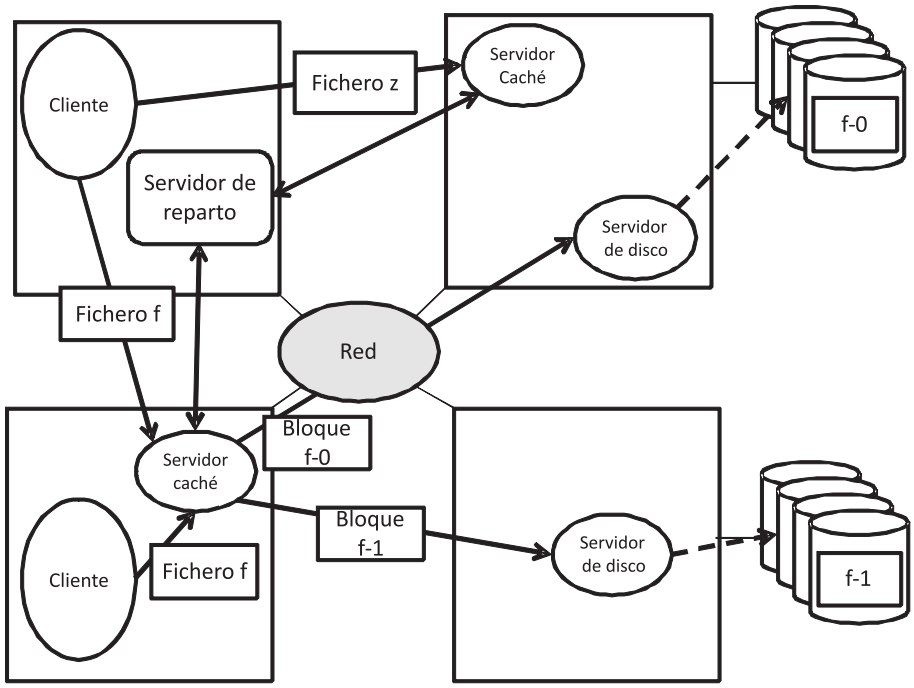

Figura 2.12: Arquitectura de PAFS.

El sistema DAche [87, 88], proporciona una caché global distribuida para procesos de una misma aplicación paralela. Esta caché se encuentra integrada en ROMIO lo que proporciona compatibilidad con aplicaciones usadas con MPI-IO. En este caso, los procesos almacenan bloques de tamaño fijo en el espacio de memoria del cliente y se reparten entre los distintos procesos de forma cíclica. Esta caché mejora el rendimiento de una aplicación paralela, pero no es adecuada para múltiples aplicaciones que manejen los mismos datos, ya que no pueden coordinarse.

Otro ejemplo similar de caché usada en los procesos de una aplicación paralela se encuentra en el sistema de ficheros paralelo Clusterfile[42]. Al igual que en el sistema DAche, los datos son repartidos entre los diversos nodos de una aplicación paralela, si bien en este caso los datos son distribuidos según se establezca en la

visión de los procesos.

2.4. Bibliotecas e interfaces de E/S

En la comunidad científica se han establecido dos grandes paradigmas de programación para entornos de computación de alto rendimiento: paso de mensajes y paralelismo en los datos. En la siguiente sección se realiza un breve repaso a los representantes destacados: *Message Passing Interface* (MPI) y *High Performance Fortran* (HPF), respectivamente.

2.4.1. Message Passing Interface(MPI)

MPI [89] es una especificación estándar para bibliotecas basada en el paradigma de paso de mensajes. Se utiliza para resolver problemas científicos y de ingeniería en entornos de alto rendimiento.

Desarrolla el modelo SPMD (*Single Program Multiple Data*), en el que se escribe un único programa con el que se genera un ejecutable mediante un compilador de MPI. Este ejecutable considera dos parámetros básicos para su funcionamiento: el número total de procesos utilizados en la aplicación paralela, y el identificador interno dispuesto para cada uno de los procesos (este identificador es un número comprendido entre 0 y n-1, siendo n el número total de procesos de la aplicación paralela). Este identificador es único para cada uno de los procesos de la aplicación paralela y determina en ocasiones el flujo de control en cada uno de los procesos.

El modelo utilizado en MPI establece que cada uno de los procesos de una aplicación gestiona los recursos locales (memoria, etc.), y que los distintos procesos de una aplicación paralela pueden comunicarse entre sí mediante el mecanismo denominado *paso de mensajes*. Estos pasos de mensajes pueden ser punto a punto (usando primitivas del tipo *send* o *receive*) o colectivos (usando primitivas como *broadcast* o *reduce*).

Para poder comunicar los distintos procesos de una aplicación paralela se utiliza un ente denominado *comunicador*. En MPI es posible crear múltiples comunicadores, a los cuales, los procesos se pueden adscribir.

En MPI, es posible describir los patrones de acceso a datos distribuidos en memoria o en un fichero. Estos patrones permiten expresar accesos regulares o irregulares a los datos, teniendo en cuenta también si estos accesos son contiguos o no. Mediante este mecanismo es posible realizar optimizaciones para la transmisión de los datos entre los distintos procesos.

2.4.1.1. MPI-IO

MPI-IO [90] es una interfaz estándar propuesta para el acceso a los datos por parte de una aplicación paralela. Está basada en el estándar MPI y utiliza varios de sus componentes, como los tipos de datos o los comunicadores.

En MPI-IO un fichero es abierto por el conjunto de procesos que se encuentra adscrito a un comunicador. Así, todas las llamadas colectivas implican a todos los miembros de este comunicador.

MPI-IO provee de múltiples funciones para las operaciones de E/S sobre un fichero, así por ejemplo, es posible usar:

- desplazamientos implícitos o explícitos sobre los datos manejados,
- operaciones colectivas o individuales,
- punteros de fichero individuales o compartidos por parte de los distintos procesos de una aplicación,
- u operaciones bloqueantes o no sobre los datos.

También es posible declarar las vistas disponibles en un proceso sobre un fichero, de esta manera, se realiza una separación entre la representación lógica de los datos y su distribución física a lo largo del fichero. Estas funciones o mecanismos proveen de una gran flexibilidad al programador a la hora de manejar los datos, pudiendo adaptarse a los requerimientos de la aplicación. A su vez, el uso de operaciones específicas como las colectivas permiten la optimización interna de las mismas.

2.4.2. High Performance Fortran

HPF (*High-Performance Fortran*) [91] es una extensión del lenguaje de programación Fortran que permite al programador indicar al compilador cómo distribuir los datos, así como paralelizar el código. HPF asume la existencia de un único hilo de control y una única memoria global. Las ejecuciones de las instrucciones no son sincronizadas a lo largo de los distintos procesadores. Sin embargo, las operaciones realizadas sobre los elementos de una matriz manejada por los distintos procesadores, se ejecutan de forma simultánea para evitar incoherencias en los datos manejados.

A diferencia de MPI, donde el programador distribuye los datos a lo largo de los distintos procesos de una aplicación paralela de forma explícita, en HPF estos datos se reparten usando directivas que reducen la complejidad de la programación y declaran el patrón de la distribución (regular o irregular).

Cada una de las dimensiones (de tamaño N) de una matriz puede ser distribuida sobre P procesos de la aplicación mediante una de las siguientes etiquetas en la directiva:

- * : sin distributión de los datos.
- BLOCK(n) : distribución de los bloques de forma equitativa (por defecto n=N/P).
- CYCLIC(n) : reparto de bloques de igual tamaño de forma cíclica. (por defecto n=1)

La Figura 2.13 muestra distintos ejemplos de distribución de los datos de una matriz, siguiendo las etiquetas citadas.

Por otra parte, es posible repartir los datos de forma dinámica mediante directivas específicas.

2.5. Formatos portables de datos y bibliotecas para el manejo de datos

Las interfaces de E/S de bajo nivel, como por ejemplo POSIX, describen los datos como una secuencia de bytes. Sin embargo, niveles superiores de abstracción donde las aplicaciones o usuarios manejan tipos de datos complejos en lugar de flujos de datos, requieren de otro tipo de bibliotecas. Estas aplicaciones se ejecutan en múltiples plataformas lo que requiere una portabilidad de los datos, sin necesidad de ningún tipo de cambio en la aplicación. Por último, estas bibliotecas permiten el acceso paralelo a los datos ya que las aplicaciones suelen ser utilizadas en entornos paralelos. Esta sección describe dos bibliotecas de datos muy utilizadas en entornos científicos: netCDF y HDF5.

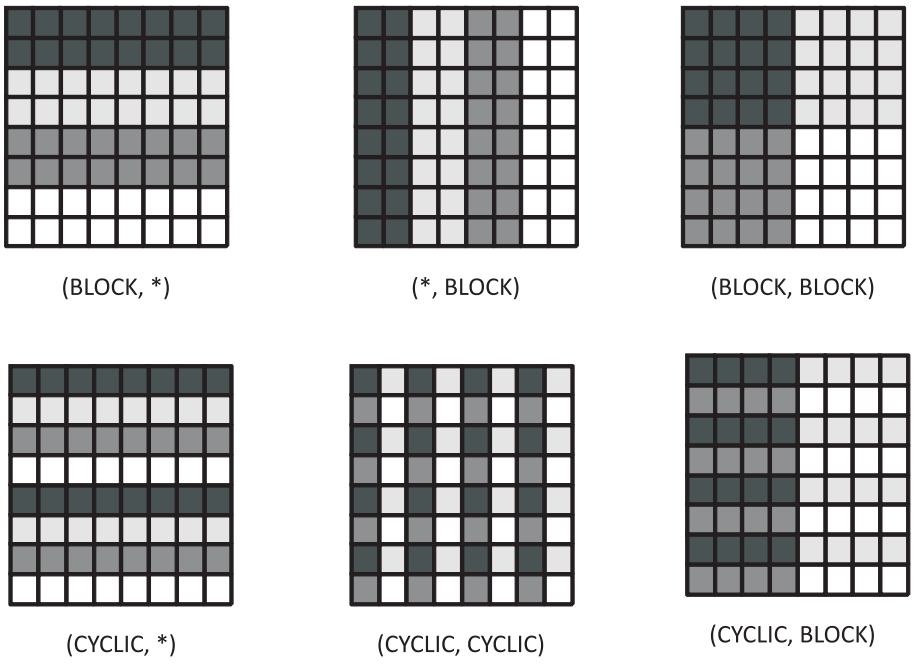

Figura 2.13: Ejemplos de distribución de datos en HPF.

Antes de presentar una descripción detallada del diseño de cada una de estas bibliotecas, presentaremos un breve análisis de los métodos de acceso por parte de las aplicaciones a las bibliotecas. La primera estrategia para el acceso a los datos se encuentra descrita en la Figura 2.14a donde un único proceso recopila y distribuye los datos mediante una interfaz secuencial para el acceso a los mismos. El inconveniente de este enfoque es que se crea un cuello de botella al usar un único proceso para

esta tarea, por lo que se sobrecarga la memoria del equipo en el que se encuentre este proceso.

Para evitar este problema, otro método alternativo para acceder a los datos de los procesos es el uso de una interfaz secuencial proporcionada por la biblioteca, que sólo tiene el inconveniente de que sólo puede usarse para ficheros distintos. De esta manera, tal y como se refleja en la Figura 2.14b todas las operaciones de E/S pueden ser concurrentes. De forma que la gestión de los conjuntos de datos es más difícil, lo que implica un coste mayor de integración con las aplicaciones.

La última estrategia consiste en proporcionar una interfaz para el acceso paralelo a los datos, tal y como se puede apreciar en la Figura 2.14c. Ésta evita los cuellos de botella y permite la incorporación de optimizaciones de E/S.

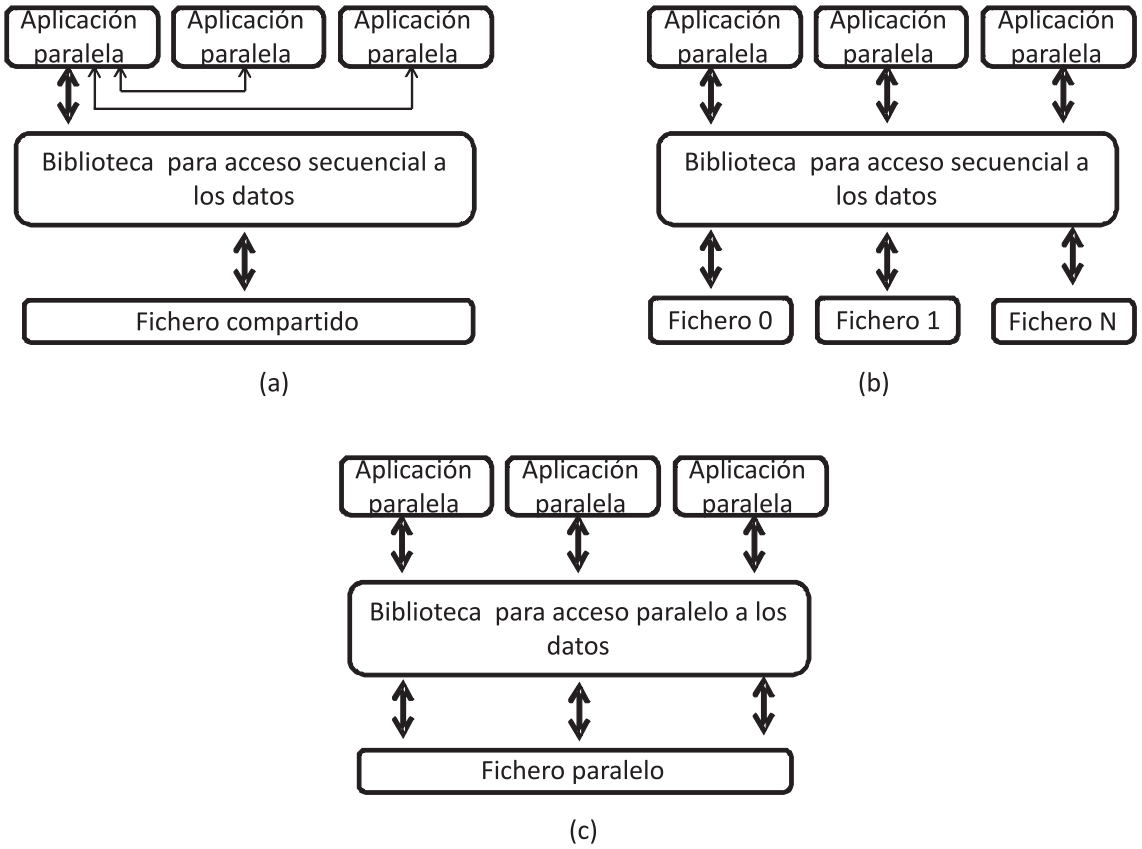

Figura 2.14: Esquemas de acceso a los datos mediante una biblioteca de datos.

2.5.1. netCDF

NetCDF [92], desarrollado por el UPC (*Unidata Program Center*), provee a las aplicaciones con múltiples métodos para el almacenamiento de conjuntos de dato estructurados. Así por ejemplo, las aplicaciones científicas utilizadas para el análisis del clima usan la biblioteca NetCDF como estándar en la gestión de una gran variedad de tipos de datos: series temporales, mallas, imágenes de satélite, etc.

NetCDF almacena matrices de datos que contienen dimensiones, atributos y variables. Físicamente, el fichero con el conjunto de datos se encuentra dividido en dos partes: una cabecera con los metadatos y un conjunto de datos que representan a los datos de la matriz almacenada. Los metadatos contienen los atributos asociados a los datos como dimensiones de la matriz, atributos, etc.

En el origen, netCDF únicamente disponía de una interfaz para acceso a los datos de forma secuencial. Una nueva versión denominada PnetCDF [93] desarrollada por la NA (*Northwestern University*) y el ANL (*Argonne National Laboratory*), añade una interfaz para el acceso paralelo que aumenta el rendimiento de forma significativa. Para la implementación de esta versión paralela se ha utilizado MPI-IO (ver Figura 2.15), lo que permite realizar todas las optimizaciones indicadas en secciones anteriores, como *two-phase I/O* o *data sieving*.

Con PnetCDF, en las aplicaciones paralelas las operaciones de apertura, manejo y cierre de un fichero se realizan de forma conjunta, pero la actualización de los metadatos del fichero se efectua en un único proceso.

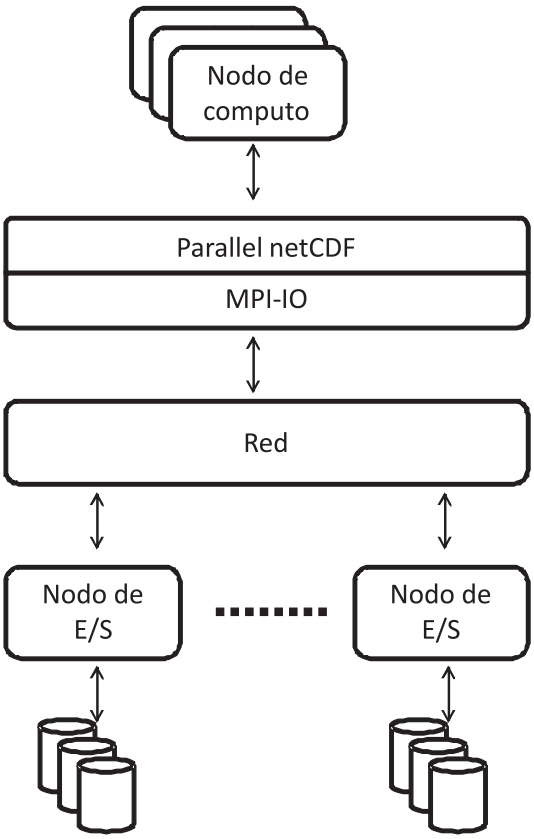

Figura 2.15: Diseño de PnetCDF.

2.5.2. HDF5

HDF (*Hierarchical Data Format*) es un formato portable de datos desarrollado por la NCSA (*National Center for Supercomputing Applications*). Esta diseñado para almacenar, recuperar, analizar, visualizar y convertir datos científicos. La versión más utilizada de este formato es HDF5 [94], la cual maneja matrices multidimensionales. HDF5 usa una estructura jerárquica que gestiona los datos, lo que permite a los programadores indicar múltiples maneras de organizar los datos. A su vez, permite el acceso paralelo a los datos.

La estructura de HDF5 tiene dos clases de componentes distintos: grupos y conjuntos de datos. Un grupo contiene instancias de grupos o conjuntos de datos, mientras que un conjunto de datos representa a una matriz multi-dimensional que contiene los datos finales. Todo conjunto de datos o grupo, tiene asociado un conjunto de metadatos, como número de elementos, etc. Uno de las mayores diferencias con respecto a netCDF consiste en la posibilidad de especificar matrices con dimensiones ilimitadas.

HDF5 soporta el acceso a porciones escogidas de datos lo que permite trabajar sobre espacios de datos definidos por el usuario previamente, de forma análoga a las vistas en MPI-IO. Proporciona una interfaz para el acceso secuencial a los datos y otra paralela. El acceso paralelo a los datos se realiza a través de MPI-IO lo que posibilita realizar optimizaciones en las operaciones de E/S, como las vistas en secciones anteriores. Cada proceso que intervenga en el acceso a los datos define el espacio en el cual va a trabajar, de forma que se pueden efectuar operaciones de tipo colectivo o individual sobre los datos.

2.6. Caracterización de los accesos de E/S de las aplicaciones en entornos HPC

Las aplicaciones utilizadas en entornos de alto rendimiento tienen un comportamiento bien conocido debido a los múltiples estudios realizados . Un subconjunto muy importante de este tipo de aplicaciones son las utilizadas en entornos científicos, las cuales usan entornos paralelos para solventar la necesidad de gran cantidad de memoria, numerosos recursos de computación y suelen realizar grandes demandas de E/S. Estas operaciones de E/S pueden disponer de patrones muy complejos [95].

Es posible clasificar a las aplicaciones en categorías, según su comportamiento de E/S [73]:

1.- Aquellas aplicaciones cuya operación de E/S es obligada, por ejemplo las que realizan operaciones de *data-mining*.

2.- *Checkpoint*, que permiten salvar el estado en un instante dado de la aplicación. Así, en caso de fallo es posible continuar la aplicación a partir del momento en el que se obtuvo.

3.- Aplicaciones que imprimen instantáneas para conocer su progreso.

4.- Aplicaciones que debido a la sobrecarga de la memoria principal utilizan una fuente externa para almacenar partes de la información usada.

5.- Por último, aplicaciones que imprimen los resultados obtenidos para visualización o post-procesamiento.

Las categorías 2 y 5 son las más habituales en entornos *cluster*, siendo las operaciones de escritura de datos las más comunes.

Por otra parte, tal y como se ha ido detallando a los largo del capítulo, los sistemas de ficheros paralelos, tienen como prioridad proporcionar a las aplicaciones un acceso eficiente a los datos. A lo largo del tiempo se han estudiado numerosos patrones de acceso en este tipo de entornos de E/S [96, 97, 98].

A continuación se presenta a modo resumen, un conjunto de observaciones realizadas por parte de distintos investigadores:

- Con frecuencia, los datos de un fichero son compartidos por parte de múltiples aplicaciones. En una aplicación paralela no es habitual [96, 99] el uso compartido de los datos, por lo que los accesos a los datos por múltiples procesos pertenecientes a una misma aplicación paralela no se suelen superponer.
- Los datos de los ficheros son distribuidos a lo largo de múltiples discos para incrementar el rendimiento de las operaciones de E/S, de forma que se pueden efectuar operaciones sobre distintos discos de forma concurrente.
- Los procesos de una aplicación paralela no siempre acceden a los datos de forma secuencial, sino que pueden realizar saltos de forma entrelazada [100]. Estos accesos no contiguos a nivel global pueden implicar accesos dispersos en los discos.
- Las investigaciones sugieren que las aplicaciones generan un enorme número de pequeñas peticiones de E/S [96].
- Los accesos de E/S paralelos no siempre se distribuyen de forma proporcional sobre los distintos nodos de E/S, lo que puede producir contención sobre algunos, reduciendo a su vez el paralelismo en los accesos.
- Las aplicaciones paralelas suelen usar patrones de acceso a los datos de forma entrelazada. Este tipo de acceso suele ocurrir por ejemplo con matrices multi-dimensionales, las cuales son divididas entre los distintos procesos de la aplicación [96].

Todos estos motivos indican que las soluciones actuales para los sistema de almacenamiento no son las más adecuadas ya que no se adaptan a las características particulares que tiene cada aplicación, lo que fomenta una perdida de rendimiento tanto en las aplicaciones como en los sistemas de almacenamiento.

2.7. Resumen

En este capítulo se ha presentado una visión completa de la problemática de la E/S en entornos HPC: arquitecturas de E/S, sistemas de ficheros distribuidos y paralelos más extendidos, técnicas de optimización de E/S, bibliotecas y paradigmas de programación usados en aplicaciones HPC, los formatos utilizados, y las características generales del comportamiento de E/S en este tipo de aplicaciones .

Las arquitecturas de E/S y los sistemas de almacenamiento utilizados en los entornos HPC suelen ir a la par, por lo que un cambio en la arquitectura de E/S suele tener como consecuencia principal un cambio en este tipo de sistemas. Este cambio puede traer consigo desde una reconfiguración hasta un despliegue completamente nuevo del mismo en la nueva arquitectura. Esto provoca un coste, a nivel económico, ya que muchos componentes no se pueden reutilizar, lo que involucra un mayor coste en hardware o software, pero también, y como efecto secundario, un coste temporal (necesario para la reconfiguración, instalación, etc. de los nuevos componentes).

En el siguiente capítulo se expondrán las distintas soluciones a los problemas planteados.

Capítulo 3

Motivación

En el presente capítulo se detallará con más profundidad la problemática y la motivación de la tesis.

3.1. Introducción

Las arquitecturas *cluster* han pasado a ser la solución perfecta para los problemas de cómputo de índole científico y empresarial. Concretamente, el 82 % de los computadores presentes en el Top 500 [3] son *clusters*.

Los *clusters* pueden clasificarse según los elementos que lo componen:

- Heterogéneos: donde los elementos son diferentes en hardware y/o software (incluso de sistema operativo). Normalmente se basa en la agregación de computadores y sistemas de red o almacenamiento que poseen distintas tecnologías. Tienen grandes problemas en la integración de las distintas tecnologías disponibles.
- Homogéneos: los elementos de los que se compone son iguales en todos los aspectos (hardware y software) lo que facilita la construcción de grandes instalaciones de cómputo, como el Marenostrum [6].

Numerosas aplicaciones se ejecutan en cada uno de estos *cluster*, generando múltiples operaciones de E/S: tanto de escritura (para ficheros de salida) como de lectura (para aplicaciones de tipo *data mining*). Esto establece el sistema de almacenamiento como un importante punto a tener en cuenta para el rendimiento de las aplicaciones y del sistema en general. Las prestaciones del sistema de almacenamiento son primordiales: rendimiento, facilidad de administración, escalabilidad, etc.

Así que la motivación fundamental por la que se desea realizar la presente tesis es: *la mejora de las prestaciones de los sistemas de E/S en entornos cluster.*

A continuación se detallarán las problemáticas propias de cada uno de los tipos de *cluster*, así como la solución diseñada para cada caso.

3.2. *Clusters* heterogéneos

Como ya se indicó anteriormente, los *cluster* heterogéneos tienen su mayor problema en la integración de distintas tecnologías. Este problema se ve agudizado cuando se comparten los sistemas de almacenamiento los distintas tecnologías (hardware y software). Existen sistemas de ficheros en red que permiten hacer esto, como SMB [35] o NFS [15].

Sin embargo, se presentan distintos problemas en los sistemas de E/S:

- El acceso a los datos no es homogéneo, debido al uso de distintos sistemas hardware y/o software por parte de los nodos de cómputo y/o del sistema de almacenamiento.
- No son sistemas escalables. No permiten la construcción de grandes sistemas a partir de la agregación de tecnologías ya existentes.
- No es posible el uso de tecnologías conocidas para el aumento de prestaciones de los sistemas de almacenamiento, como por ejemplo de los sistemas de ficheros paralelos, debido a que no funcionan en todo tipo de plataformas hardware o software.
- Los sistemas de almacenamiento incrementan su complejidad de gestión según aumentan las prestaciones de los mismos, lo que dificulta el manejo de este tipo de sistemas.

Por estos motivos, se desea aliviar esta problemática mediante el uso de *una plataforma software que permita el acceso homogéneo a los datos y que utilice servidores de E/S estándares para conformar los sistemas de almacenamiento para arquitecturas cluster.*

La plataforma propuesta en esta tesis se denomina Expand (*Expandable parallel file system*). Con ella se pretende ofrecer un sistema de almacenamiento de altas prestaciones con servidores y protocolos estándares, como NFS [101], FTP o GridFTP [102]. Esto facilita su integración en sistemas heterogéneos, la reutilización y agregación de recursos existentes y el acceso paralelo a los datos.

La Figura 3.1 muestra la arquitectura global de Expand. En esta figura se puede apreciar el uso de varios servidores de datos estándar para construir un sistema de ficheros paralelo. Los datos de los ficheros se distribuyen a través de todos los servidores de datos, utilizando bloques de diferente tamaño como unidad de reparto. Los procesos en los nodos clientes utilizan Expand para acceder a los datos de las particiones distribuidas.

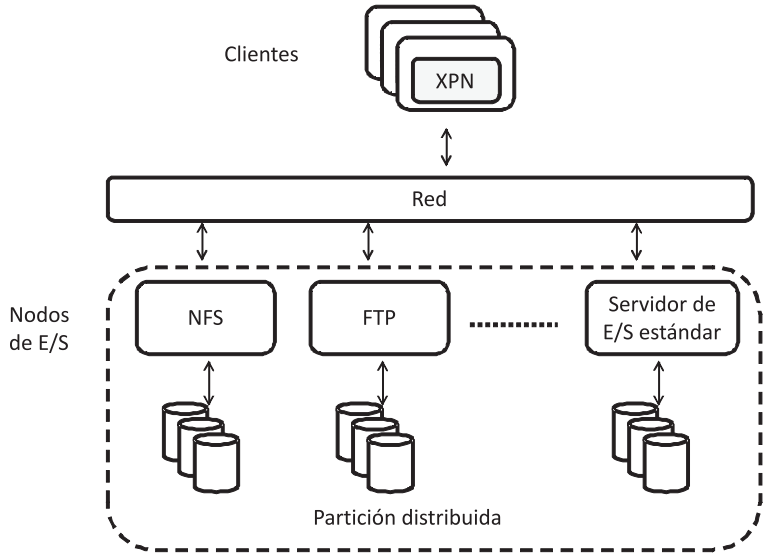

Figura 3.1: Descipción de la arquitectura Expand.

3.2.1. Objetivos

Para el diseño del sistema de ficheros paralelo se pretenden cumplir los siguientes objetivos.

3.2.1.1. Uso de sistemas estándar de almacenamiento para su construcción

Para la construcción de Expand se utilizarán servidores de datos estándar. Todos los aspectos del diseño y operaciones de Expand se deben implementar en los clientes. Esta característica independiza a Expand de la infraestructura de almacenamiento subyacente y simplifica la construcción del sistema de ficheros, puesto que todas las operaciones se implementan en el cliente. Este enfoque es completamente diferente al empleado en todos los sistemas de ficheros paralelos existentes, que implementan un servidor de E/S propio, lo que dificulta su integración en entornos heterogéneos.

3.2.1.2. Multiplataforma

Expand debe permitir el empleo de servidores con arquitecturas y sistemas operativos diferentes, para facilitar su uso en entornos heterogéneos. Tiene que ser independiente del sistema operativo utilizado en el cliente y en el servidor. Todas las operaciones han de implementarse mediante el protocolo del servidor de datos usado. Así, por ejemplo, cuando se emplé NFS como servidor, se deben utilizar llamadas a procedimientos remotos (RPC).

3.2.1.3. Facilitar la gestión del sistema de ficheros

Tiene que permitirse una gestión sencilla del sistema de ficheros. Esto es posible al usar protocolos y servidores estándar ampliamente utilizados, se facilita la configuración de estos sistemas para su uso en el sistema de ficheros Expand. Así, por ejemplo, cuando se emplea NFS, el servidor solo necesita exportar los directorios apropiados y los clientes solo necesitan un pequeño fichero de configuración que describa la partición distribuida.

3.3. Grandes instalaciones

Las grandes instalaciones *cluster* se han convertido en una de las principales plataformas para la ejecución de aplicaciones científicas, empresariales, etc. con grandes necesidades de cómputo. Estas aplicaciones generan o manejan grandes volúmenes de información durante su ejecución.

Así por ejemplo, las aplicaciones de minería de datos (*data mining*) realizan operaciones de búsqueda y clasificación sobre registros que se encuentran almacenados en grandes volúmenes de información. Otras aplicaciones, como las utilizadas en simulaciones científicas generan una gran cantidad de datos en un corto espacio de tiempo. Un claro ejemplo de este tipo de aplicaciones es la denominada GADGET-2 [103], utilizada en la simulación del cosmos realizada en el supercomputador Marenostrum [104, 6]. Una simulación de este programa realizada en este supercomputador ha necesitado de 512 procesadores en paralelo durante 447 horas (equivalentes a más de 29 años de cómputo secuencial), generando 8.6 terabytes de datos distribuidos en 135 ficheros (*snapshots*) de 64 gigabytes cada uno, los cuales actualmente están siendo procesados para su estudio. Como se puede apreciar, la fase de E/S tiene una gran importancia en el rendimiento de las aplicaciones, por lo que una reducción del rendimiento del sistema de almacenamiento tiene como efecto colateral una reducción del rendimiento global de las aplicaciones del sistema.

Los sistemas de almacenamiento de estas grandes instalaciones presentan distintos problemas:

- Existe una disparidad muy grande entre el número de nodos de cómputo y el número de nodos de E/S. El número de nodos de cómputo es mayor al número de nodos de E/S disponibles en el sistema de almacenamiento, lo que puede provocar un colapso del sistema en caso de que se produzca una gran sobrecarga en el sistema de almacenamiento.

- Las arquitecturas de almacenamiento son estáticas, poco adaptables y flexibles a los requisitos de las aplicaciones.

- Las aplicaciones usadas en los entornos *cluster* son heterogéneas, utilizan patrones y tamaños de acceso muy diversos. La solución tradicional a este problema es la adaptación de las aplicaciones al entorno *cluster*, reduciendo el rendimiento de las aplicaciones.

- Los sistemas de ficheros no aprovechan oportunidades de mejora en estos entornos evitando accesos o copias de datos innecesarias. Se debe a que las aplicaciones de este tipo de entornos tienen comportamientos bien conocidos, por ejemplo usando conjuntos de datos compartidos o generando ficheros temporales que no son utilizados posteriormente (por ejemplo los ficheros de *checkpointing*).

Un ejemplo de desequilibrio de carga se puede observar en la Figura 3.2 que representa los tiempos obtenidos ejecutando la aplicación paralela denominada FLASH-IO [105] en el *cluster* Cacau [106, 107], perteneciente al instituto de investigación HLRS de Stuttgart (Alemania).

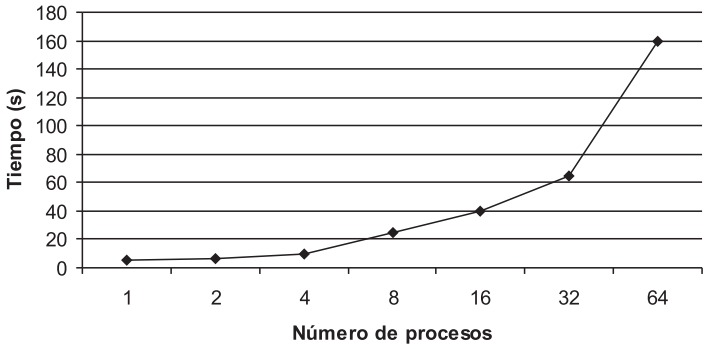

Figura 3.2: Evaluación de la aplicación FLASH-IO en un *cluster*.

La aplicación FLASH-IO simula las operaciones de E/S realizadas por la aplicación FLASH, generando diversos ficheros mediante accesos no contiguos en disco. El incremento en el número de procesos, implica un aumento del número de operaciones de E/S por parte de la aplicación. Para la prueba realizada en el *cluster* Cacau, se dispuso de hasta 64 nodos de cómputo y del sistema de almacenamiento compuesto por dos nodos de E/S. En la Figura 3.2 se puede apreciar como aumentando el número de procesos usados en la aplicación FLASH-IO, el tiempo necesitado para completar las operaciones de E/S sigue una progresión geométrica, es decir, el sistema de almacenamiento no escala al aumentar el número de operaciones de E/S. En este caso el sistema de almacenamiento se ha convertido en un cuello de botella que impide el uso masivo del mismo.

Para resolver estos problemas se propone emplear una arquitectura de E/S basada en el uso de sistemas intermedios de almacenamiento. La Figura 3.3 muestra el esquema de la arquitectura propuesta. En ella se puede ver como un conjunto de sistemas intermedios de almacenamiento realizan la mayor parte de las operaciones de E/S solicitadas por las aplicaciones, mientras estos sistemas intermedios son los que acceden a los datos del sistema de almacenamiento del *cluster*.

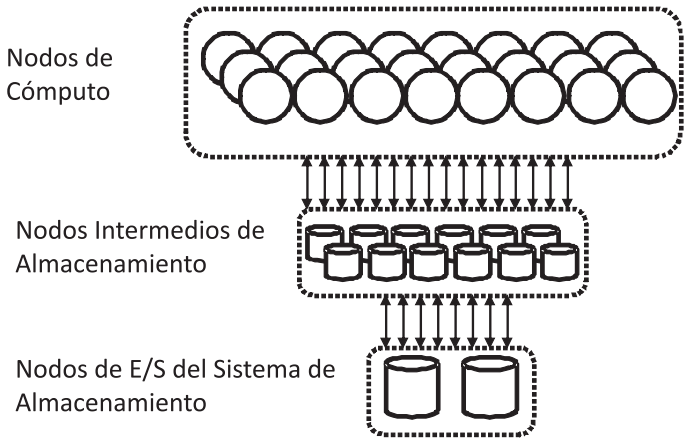

Figura 3.3: Esquema de la arquitectura propuesta en grandes *clusters*.

3.3.1. Objetivos

Los objetivos propuestos para diseñar esta arquitectura son los siguientes:

3.3.1.1. Flexibilidad y escalabilidad

La arquitectura de E/S tiene que proporcionar al usuario y al administrador del sistema la posibilidad de adaptar los distintos parámetros disponibles en la misma, para flexibilizar la arquitectura a los distintos requisitos de las aplicaciones (como el tamaño de reparto, políticas de reparto, etc.). De esta manera, se pueden obtener grandes rendimientos de una aplicación sin necesidad de cambiarla. Además, al poder adaptar el número de sistemas intermedios de almacenamiento al de nodos de cómputo, es posible tener tantos nodos de almacenamiento como aplicaciones se ejecuten en el sistema, lo que permite una gran escalabilidad del sistema de E/S.

3.3.1.2. Acceso paralelo a los datos

La arquitectura debe proporcionar mecanismos para el acceso paralelo a los datos, mejorando el rendimiento de las aplicaciones gracias a la posibilidad de adaptar el grado de paralelismo de las mismas.

3.3.1.3. Arquitectura software

El diseño de la arquitectura ha de realizarse teniendo en cuenta que el sistema de E/S no puede verse modificado en su hardware, esto es posible mediante el desarrollo de la misma vía software. Esta arquitectura debe integrarse en el sistema de ficheros de los nodos de cómputo, convirtiéndose en un intermediario de las peticiones de las aplicaciones.

3.3.1.4. Localización

La arquitectura utilizará los recursos disponibles en los nodos de cómputo, como son memoria, dispositivos de almacenamiento, etc. Las aplicaciones se encuentran más cerca de los datos que en las arquitecturas de almacenamiento tradicionales. De manera que estos se pueden localizar en los propios nodos de cómputo donde se situan las aplicaciones que los usan. Esta estrategia mejora el rendimiento del sistema al reducir las comunicaciones para la obtención de datos.

Por otra parte, los nodos de cómputo disponen generalmente de una red interna de comunicación de datos, como Myrinet, caracterizada por bajas latencias y grandes anchos de banda. Los sistemas intermedios de almacenamiento pueden utilizar estas redes para reducir los tiempos de las comunicaciones.

3.3.1.5. Transparencia

La arquitectura debe premitir el acceso transparente a los datos. Esto es posible mediante un sistema gestor de peticiones [11], la incorporación de ésta en el sistema de ficheros [108] o la integración de la misma en una biblioteca a nivel de usuario, como MPI [109].

Además el espacio de nombres de los datos no puede verse afectado.

3.3.1.6. Reducción de los cuellos de botella

Las aplicaciones pueden generar cuellos de botella durante las fases de E/S. La arquitectura propuesta debe reducir este problema mejorando el rendimiento del sistema de las siguientes formas:

- Mediante una gestión de las peticiones de E/S enviadas al sistema de almacenamiento. Por ejemplo, agrupando múltiples peticiones de pequeño tamaño en pocas de gran tamaño.

- Desplazando la carga de las operaciones de E/S a los sistemas de almacenamiento intermedio.

3.3.1.7. Oportunidades de mejora del rendimiento

La arquitectura debe proporcionar un sistema de almacenamiento que permita el acceso a datos compartidos por múltiples aplicaciones. Esto reduce los tiempos de acceso a los datos, lo que incrementa el rendimiento de las operaciones de E/S. Cabe destacar que en el caso de las arquitecturas *multicore*, donde múltiples aplicaciones ejecutan sobre un mismo nodo de cómputo, se puede reducir considerablemente el número de accesos al sistema de almacenamiento del *cluster*.

3.3.1.8. Incremento del rendimiento percibido por las aplicaciones

Las aplicaciones tienen fases de cómputo y de E/S. En las fases de E/S, los nodos de cómputo no realizan ningún otro tipo de operación, lo que provoca una reducción del rendimiento en las aplicaciones. La arquitectura propuesta traslada la carga de E/S generada por las aplicaciones a los sistemas intermedios de almacenamiento, lo que reduce el número de tiempos muertos.

Además, debido al reducido número de los nodos de E/S en un *cluster*, se produce una contención en las operaciones de E/S de las distintas aplicaciones. Al posicionar un número de estos nodos intermedios de almacenamiento mayor que el número de nodos de E/S, el grado de paralelismo en las operaciones de E/S aumenta. Por otra parte, utilizando políticas de planificación para la transferencia de los datos entre los nodos intermedios y el sistema de almacenamiento, se mejora el rendimiento global del sistema. Este es el caso de los ficheros temporales o de *checkpoint*, que pueden no almacenarse en el sistema o transferirse usando políticas de escritura diferída.

3.4. Resumen

Este capítulo presenta la motivación de la tesis: *la mejora de los sistemas de E/S de los entornos cluster*. Se centrá en dos entornos: los *clusters* heterogéneos, donde la multitud de tecnologías dificultan el uso de sistemas de E/S específicos, y los grandes *clusters*, donde la cantidad de elementos de cómputo con respecto a los recursos de E/S disponibles produce un desequilibrio.

Para resolver ambos problemas se han planteado las siguientes soluciones:

- *Clusters* heterogéneos: se propone el uso de un sistema de ficheros paralelo multiplataforma denominado Expand que utilice tecnologías estándar para su desarrollo y que proporcione acceso paralelo a los datos.
- Grandes *clusters*: se plantea el uso de una nueva arquitectura software de E/S basada en el uso de sistemas intermedios de almacenamiento. Esta arquitectura utiliza los recursos de los nodos de cómputo estableciendo espacios de almacenamiento para los datos manejados por las aplicaciones.

En siguientes capítulos se definirán con mayor detalle los diseños de las soluciones propuestas.

Capítulo 4

Arquitectura, diseño e implementación de Expand

En este capítulo se describe la arquitectura, diseño e implementación del sistema de ficheros paralelo *Expand*. Este sistema de ficheros paralelo permite eliminar la heterogeneidad en el acceso a los datos.

En las siguientes secciones se describen los principales aspectos relativos al diseño de Expand.

4.1. Introducción

Los *clusters* heterogéneos disponen de nodos de cómputo con distintos sistemas operativos, etc; lo que limita la cantidad de sistemas de ficheros adecuados para este tipo de sistema al uso de los distribuidos como NFS o CIFS, que se encuentran estandarizados en múltiples entornos de computación. Dicha dificultad para incorporar sistemas de ficheros paralelos para este tipo de entornos, impide el uso de técnicas de acceso paralelo a los datos y disminuye las prestaciones de los sistemas de almacenamiento.

Afrontando esta problemática se ha diseñado el sistema de ficheros paralelo Expand, que utiliza servidores de datos estándar (como NFS) para formar sistemas de almacenamiento paralelos.

En los siguientes apartados se detallarán las características del sistema de ficheros paralelo.

4.2. Definiciones

Un fichero (F) tiene una proyección en *Expand* (F_P) mediante una función f_e. Estos ficheros paralelos (F_P) se definen como:

$$f_e(F) \rightarrow F_P : F_P \equiv (D, M) \tag{4.1}$$

Donde D representa los datos de usuario distribuidos en el espacio de almacenamiento definido en Expand y M el conjunto de metadatos asociados al fichero paralelo F_P. Los datos D y los metadatos M se distribuyen a lo largo de varios espacios de almacenamiento EA, que se agrupan en particiones distribuidas (P) (véase la Figura 4.1) y cada EA se compone de un servidor de datos (EA_D) y un directorio exportado (EA_E).

$$EA_i \equiv (EA_{Ei}, EA_{Di}) \tag{4.2}$$

$$P \equiv \bigcup_{i=1}^{n} EA_i \tag{4.3}$$

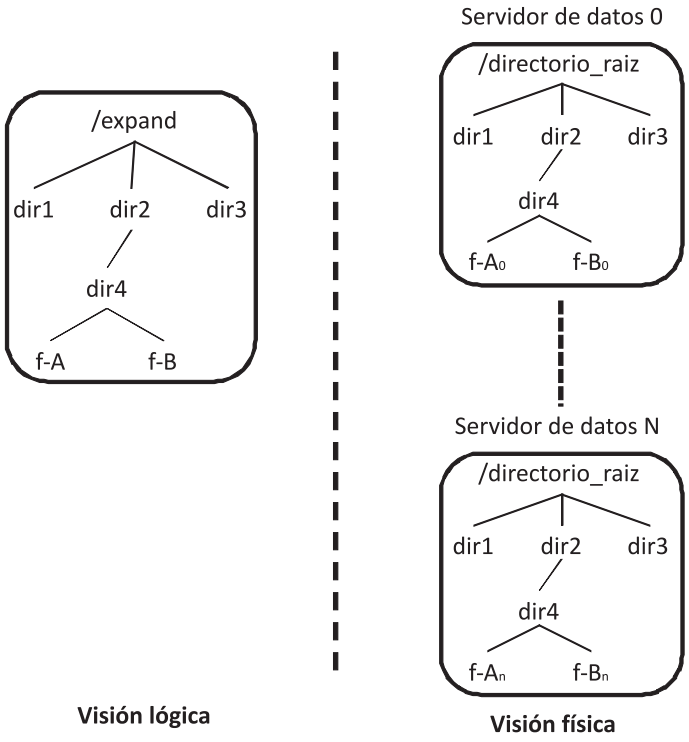

Figura 4.1: Estructura de los ficheros y directorios en Expand.

Los datos (D) de los ficheros paralelos (F_P) se distribuyen a través de los espacios de almacenamiento (EA) mediante la subdivisión en bloques (B) de igual tamaño, permitiendo el acceso paralelo a los mismos.

Los bloques de datos (B_i) se representan como el conjunto de tuplas de la forma ($desplazamiento, valor$), donde el *desplazamiento* identifica de forma unívoca la tupla con respecto al resto de tuplas del fichero, y *valor* es el dato que ha de ser almacenado en esa tupla. El desplazamiento permite la localización de un valor, formándose una secuencia ordenada de tuplas.

$$B_i \equiv \{(d_i, v_i)\} \tag{4.4}$$

$$D \equiv \bigcup B \equiv \{(d_1, v_1), ..., (d_n, v_n)\} \tag{4.5}$$

Estos bloques de datos (B) se agrupan en subficheros (S) almacenados en los distintos espacios de almacenamiento (EA). Un subfichero (S_i) se compone de uno o varios bloques de datos ordenados (B_{ij}), donde i representa el subfichero i y j el orden interno en el mismo. Por otra parte $\|B_i\|$ representa el número de bloques almacenados en S_i. Cada espacio de almacenamiento solo dispone de un subfichero por cada fichero paralelo de Expand.

$$\forall EA_i \in P, \exists S_i : S_i \subset EA_i \tag{4.6}$$

$$S_i \equiv \bigcup_{j=1}^{n} B_{ij} : n \in (N) \rightarrow \|B_i\| \subset S_i \tag{4.7}$$

$$\forall EA_i, EA_j \in P, i \neq j : EA_i \cap EA_j = \emptyset \tag{4.8}$$

La distribución de los datos se establece mediante una función de distribución $f()$, la cual determina la disposición de los bloques de datos a lo largo de los distintos subficheros de datos distribuidos en los espacios de almacenamiento. Mediante una función de distribución $f()$ es posible determinar dónde se proyecta un bloque de datos (B) en un subfichero (S_i) y un bloque de datos perteneciente a él (B_k) de forma unívoca. Para que el acceso a los datos de un fichero paralelo sea eficiente, es deseable que la función de distribución a emplear ofrezca una distribución de los datos del usuario con un uso equilibrado del espacio libre, un rendimiento máximo y la mejor fiabilidad posible.

$$\forall B_{ij} \exists f : f(B) \rightarrow (S_i, B_k) \tag{4.9}$$

En cuanto a los metadatos, contendrán la información relativa al fichero paralelo como la unidad de reparto (tamaño de bloque), la fecha de creación, la fecha de modificación, etc. La gestión de los metadatos se detallará más adelante para su estudio más detallado.

Las particiones en Expand se definen utilizando un pequeño fichero de configuración, en formato XML [110], con la siguiente estructura:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xpn_conf>
<partition name="<partition1>" type="<options>" bsize="<block size>">
 <data_node url="<protocol>:/<server>/path/"/>
 ...
</partition>
 ...

 <partition name="<partitionN>" type="<options>" bsize="<block size>">
 <data_node url="<protocol>:/<server>/path/"/>
  ...
 </partition>
</xpn_conf>
```

donde la etiqueta *partition* permite la definición de la estructura de una partición de almacenamiento. Esta partición se caracteriza por cuatro elementos básicos:

- el nombre de la partición (N_P), lo que identifica de forma única a la partición declarada,
- un tipo de partición (T_P), usado en caso de definir distintos tipos de políticas a utilizar en la distribución de los datos, que determinan la función (f) de distribución,
- un conjunto de espacios de almacenamiento usados en la partición (EA_P),
- y por último, un tamaño de bloque (S_P), utilizado por defecto para distribuir los datos de los ficheros.

$$P = \{N_P, T_P, S_P, \|EA_P\|, \bigcup_{i=1}^{n}\{EA_i\}\} \tag{4.10}$$

Por cada partición, se define uno de los servidores utilizados para formar la partición. Los servidores se definen mediante una URL la cual es única en una partición, no pudiéndose utilizar la misma URL más de una vez por partición. La URL se compone de todos los elementos necesarios para poder manejar el servidor de datos:

- un protocolo de comunicaciones (C_E),
- la dirección del servidor (I_E), así como un puerto y usuario si es necesario,
- y un directorio (D_E), donde se almacenarán los datos de los usuarios.

$$EA = \{C_E, I_E, D_E\} \tag{4.11}$$

Así, por ejemplo, el siguiente fichero de configuración define dos particiones distribuidas:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xpn_conf>
<partition name="xpn1" type="NORMAL" bsize="8k">
 <data_node id="id1" url="nfs://node0/export/home1/"/>
 <data_node id="id2" url="nfs://node1/export/home2/"/>
 <data_node id="id3" url="nfs://node2/export/home3/"/>
 <data_node id="id4" url="nfs://node3/home/"/>
</partition>

<partition name="xpn1" type="NORMAL" bsize="64k">
 <data_node id="id5" url="gridftp://node5/scratch/"/>
 <data_node id="id6" url="gridftp://node6/export/scratch/"/>
</partition>
</xpn_conf>
```

La primera utiliza cuatro espacios de almacenamiento (*EA-0*, *EA-1*, *EA-2* y *EA-3*) con protocolo NFS, versión 2. La segunda partición utiliza dos espacios de almacenamiento con protocolo GridFTP. En ambos casos se utilizan, por defecto, bloques de 8 KB como unidad de reparto. El directorio `/xpn1` constituye el directorio raíz para la primera partición y `/xpn2` el directorio raíz para la segunda. Utilizando estos archivos, el fichero Expand `/xpn1/dir/foo` se proyecta sobre los siguientes subficheros de datos como se puede ver en la Figura 4.2.

Este esquema permite incluso disponer de una partición distribuida con diferentes protocolos, lo que favorece el uso de múltiples servidores heterogéneos para formar particiones distribuidas de forma transparente a los usuarios.

4.3. Distribución de datos

Tal y como fue detallado anteriormente, cada fichero en Expand consta de varios *subficheros* (S_i) de datos, uno por cada partición utilizada en cada espacio de almacenamiento (EA). Un S_i es un fichero independiente manejado por un servidor dispuesto en el EA, de forma totalmente transparente a los clientes y usuarios de Expand. Sobre una partición distribuida, el usuario puede crear ficheros paralelos con diferentes políticas de distribución. Como por ejemplo:

- Distribución cíclica. Esta distribución, que es la que se muestra en la Figura 4.3, es la más utilizada (f_c) en la mayoría de los sistemas de ficheros paralelos.

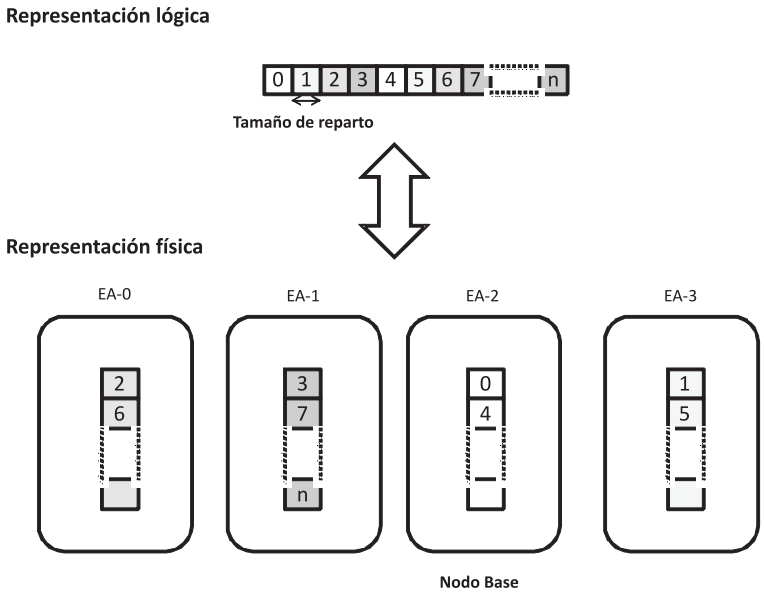

Figura 4.2: Ejemplo de proyección de los datos.

Este sistema de distribución necesita un espacio de almacenamiento *base* (EA_b) a partir del cual se inicia la distribución de los bloques de datos.

$$f_c(EA_b, B_{ij}) \rightarrow EA_j : EA_j \in P \tag{4.12}$$

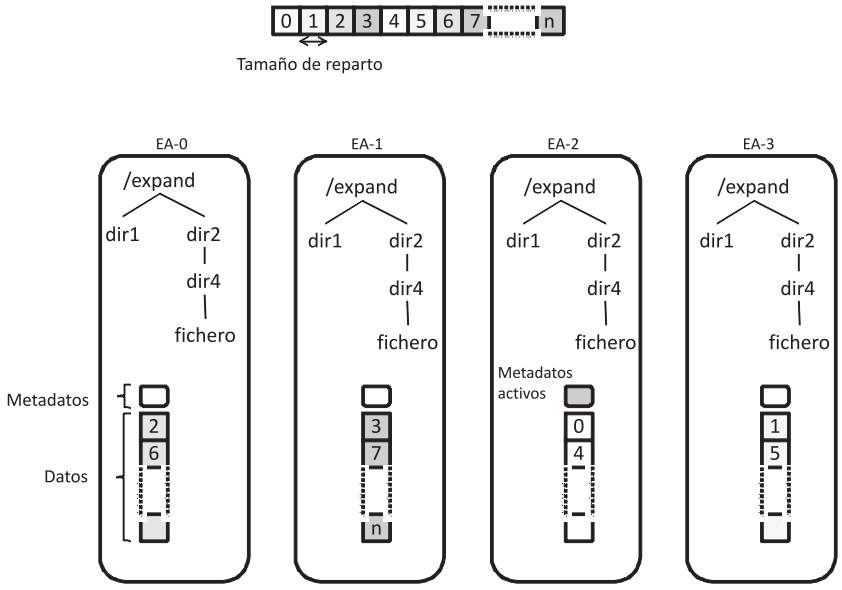

Figura 4.3: Distribución de los datos en Expand.

- Distribución aleatoria. Donde se utiliza una función de distribución f_a. En este esquema se crea un fichero semilla que se utilizará para la generación de números pseudoaleatorios. El rango de estos números estará comprendido

entre 1 y el número total de espacios de almacenamiento ($\|EA\|$) usados en la partición (P). Así, el bloque B_i del fichero se encontrará en el S_i almacenado en el EA indicado en el i-ésimo número pseudoaleatorio.

$$x \in \mathbb{N} : 1 \leq x \leq \|EA\| \in Pf_a(x, B_{ij}) \rightarrow EA_j : EA_j \in P \tag{4.13}$$

- Distribución indicada por el usuario. El usuario puede indicar la función de distribución de bloques (f_i) en la creación del fichero, al igual que se puede hacer en PVFS [11]. Así, por ejemplo, si el número de EA es 8 y el usuario indica el patrón $1-3-4-5$, el fichero se distribuirá cíclicamente utilizando solo estos 4 EA.

$$f_i(\bigcup_{i=a}^{b}\{EA_i)\}, B_{kj}) \rightarrow (EA_k) : EA_k \in P \tag{4.14}$$

Se han estudiado otras políticas de distribución de datos basadas en el espacio disponible de los distintos nodos [111]. Para este estudio se estableció un servicio gestor de nodos denominado *broker* que se comunica con los nodos de E/S y las aplicaciones. El esquema de la arquitectura, denominada AdExpand, se puede apreciar en la Figura 4.4. Para evitar posibles cuellos de botella o puntos de fallo, el servicio *broker* se puede encontrar replicado, de manera que se establecen comunicaciones periódicas entre ellos para poder tener una visión común del sistema. En la arquitectura AdeExpand, los *brokers* determinan el patrón de distribución de los datos en base al espacio disponible en los nodos de E/S en el momento de la creación de un fichero en Expand. Los nodos se añaden o eliminan del sistema a través de los *brokers* lo que permite dar una gran flexibilidad al sistema.

Las políticas definidas en base al espacio disponible en los nodos son las siguientes:

- NNREP (n servidores sin repetición). La aplicación cliente que pretende crear un nuevo fichero en una partición de Expand realiza una petición a un servicio *broker* descrito anteriormente para solicitar N servidores de datos entre todos los servidores existentes. Los servidores devueltos son aquellos con mayor porcentaje de espacio libre disponible. Esto permite que servidores pequeños con menor capacidad atiendan menos peticiones, manteniendo de esta forma el sistema más equilibrado.

- NREP (n servidores con repetición). La aplicación cliente realiza una acción similar a la indicada anteriormente, con la diferencia de que un nodo puede aparecer más de una vez en el conjunto de nodos devuelto. Los N servidores devueltos por el *broker* se seleccionan de la siguiente forma: en primer lugar se añaden al conjunto los servidores con más del 90 % de espacio libre, a continuación los servidores con más de un 75 % de espacio libre, y finalmente los servidores con más de un 50 % de espacio libre. Si no hay nodos con más de un 50 % de espacio libre se utiliza la política NNREP. La política NREP

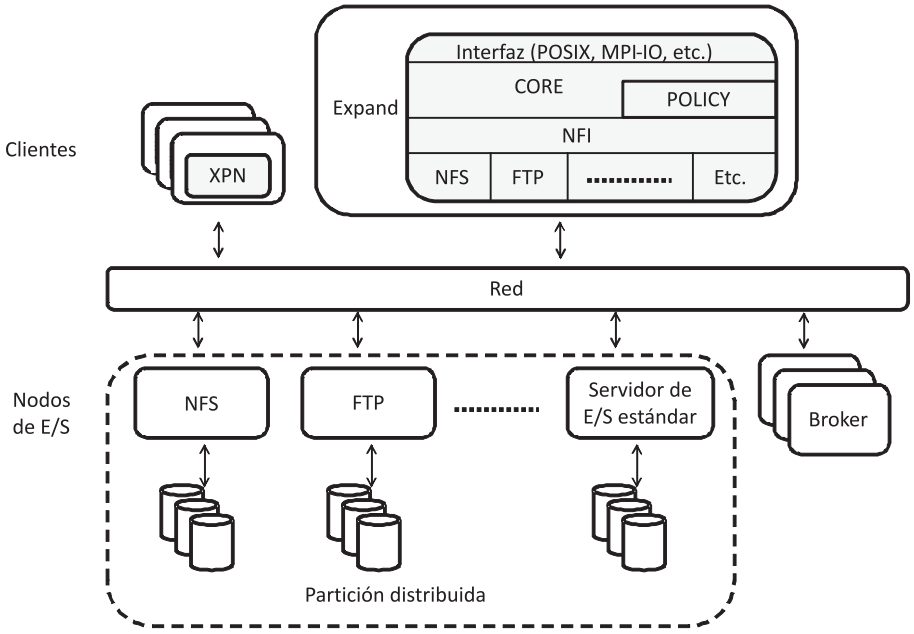

Figura 4.4: Arquitectura de Expand.

permite que un servidor aparezca varias veces en el conjunto de nodos donde se van a distribuir los datos de un fichero. La razón de esta política es que aquellos servidores con mayor capacidad libre atiendan más peticiones.

4.4. Nombrado y gestión de metadatos

En Expand es posible distinguir dos tipos de metadatos dependiendo de a qué nivel se realiza la gestión de los mismos:

- Metadatos de bajo nivel (M_E), pertenecientes a los subficheros generados por Expand en cada uno de los espacios de almacenamiento. Se encuentran gestionados por cada uno de los servidores de forma independiente y transparente a Expand. Este tipo de metadatos incluye, entre otros, el propietario del fichero, los permisos de acceso y la información sobre la localización de los bloques utilizados para almacenar los diferentes subficheros en los dispositivos de almacenamiento.

- Metadatos de alto nivel (M_P), que involucran al fichero de Expand. Son gestionados directamente por Expand. Incluyen, entre otra, la siguiente información: el tamaño de la unidad de reparto para cada fichero (una misma partición puede almacenar ficheros con tamaños de unidad de reparto diferentes), el espacio de almacenamiento *base* (que identifica el espacio de almacenamiento donde se almacena el primer bloque del fichero y el patrón de distribución a utilizar), etc.

De esta manera, los metadatos (M) de un fichero paralelo F_P se componen de:

$$M = \{M_P, \bigcup_{i=1}^{n}\{(M_{Ei})\}\} : M_{Ei} \in E_i \tag{4.15}$$

Para el diseño de los metadatos de alto nivel se han considerado diversas opciones:

- Servicio de gestión de metadatos: sistema empleado por PVFS [11] o GPFS [10], donde un servicio externo se encarga de gestionar los metadatos, lo que puede provocar un cuello de botella en el sistema. Otros sistemas de ficheros como PVFS2 [112] han descentralizado la gestión de los metadatos para evitar este tipo de problemas. Por otra parte, tener un servicio exclusivo para el manejo de los metadatos puede aumentar la complejidad del sistema.

- Cabecera de metadatos junto a los datos (B_M): en este caso se puede acceder a los metadatos de la misma manera que los datos, almacenándolos como un bloque de datos más en un subfichero S_M, lo que facilita la implementación del sistema de ficheros.

  $$S_M \in EA_M : \forall\{B_i\} \cup B_M \in S_M \tag{4.16}$$

  Estos metadatos pueden estar replicados en cada uno de los ficheros o distribuidos según algún tipo de función de distribución. Además, al estar incluidos junto a los datos, se puede acceder rápidamente a ellos ya que se facilitaal sistema de ficheros local del servidor almacenarlos en la caché de la memoria. Aparecen dos problemas en este mecanismo de gestión, por un lado se ha de establecer un método para el mantenimiento de la coherencia de los metadatos, y por otro, al incluir información dentro de los subficheros, los metadatos no pueden crecer, lo que dificulta la incorporación de nueva de información en los metadatos.

- Subfichero con el contenido de los metadatos (S_M): al igual que el caso anterior, los metadatos se encuentran almacenados como un dato, lo que facilita su manejo por parte del servidor de datos. Además esta solución permite separar los datos de los metadatos, situación que no era posible en el esquema anterior. Este esquema favorece una mayor facilidad en la creación y gestión de esquemas de replicación y tolerancia a fallos. Por otra parte, permite crecer a los metadatos de forma ilimitada, facilitando la inclusión de más información en los metadatos. Por contra, al igual que en el anterior caso, se han de establecer mecanismos para asegurar la coherencia de los metadatos.

$$F_P = \bigcup_{i=1}^{\|EA\|}\{S_i\} \cup S_M \in P \tag{4.17}$$

- Metadatos precalculados: esta última solución propuesta es la más ágil de las estudiadas hasta este momento para la gestión de los metadatos. Esto se debe a que no es necesario almacenar los metadatos físicamente en ningún dispositivo, ya que se generan mediante la configuración del sistema y la información de cada uno de los subficheros, que representan los datos del fichero paralelo. Así por ejemplo, el tamaño de un fichero paralelo se puede obtener a partir de la suma de cada uno de los tamaños de los diferentes subficheros, que conforman el fichero paralelo y están definidos en el fichero de configuración (véase Algoritmo 4.1). Este mecanismo de gestión de metadatos tiene algunas deficiencias, como la imposibilidad de mantener información individualizada de los ficheros, o la dificultad generada en algunas las operaciones de gestión de los ficheros, tales como renombrar o mover.

```
1 size(file) {
2         ∀S_i ∈ F_P(file) : ∑ f_size(S_i)
3 }
```

Algoritmo 4.1: Operación para el cálculo de tamaño de un fichero en Expand.

En Expand se ha optado por el mecanismo de gestión de metadatos basado en el uso de un subfichero externo. Este subfichero externo contiene los metadatos del fichero paralelo que se almacena en uno de los espacios de almacenamiento de la partición distribuida (véase la Figura 4.1). De manera que se incorpora el uso de metadatos precalculados para el manejo de metadatos tales como, el tiempo de creación, modificación o el tamaño del fichero, usando para ello algoritmos diseñados para la obtención de los mismos a partir de la información de los metadatos de bajo nivel de cada uno de los subficheros de datos.

El subfichero de metadatos para un fichero de Expand se almacena en uno de los espacios de almacenamiento de la partición denominado nodo *master*. Este nodo puede ser diferente del nodo *base*, donde se almacena el primer bloque del fichero. Para simplificar el proceso de nombrado y reducir los posibles cuellos de botella, Expand no utiliza un gestor de metadatos, sino que la gestión de metadatos se realiza de forma distribuida utilizando los servidores de datos de la partición.

En cuanto a la estructura de directorios, la Figura 4.1 muestra la jerarquía de directorios en Expand y cómo se proyecta sobre los servidores utilizados. En Expand, el árbol de directorios se replica en todos los EA para acelerar las operaciones de búsqueda de ficheros y directorios.

Para obtener los metadatos de un fichero es necesario acceder al subfichero de metadatos que reside en uno de los EA de la partición distribuida denominado nodo *master* (EA_M). Para conocer el EA donde se almacena este subfichero, se utiliza un método similar al empleado en el sistema de ficheros paralelo Vesta [46], mediante una función de localización f_M:

$$i \in \mathbb{N} : 1 \leq i \leq \|EA\| \in P \tag{4.18}$$

$$f_M(filename, \|EA\|) = EA_i \tag{4.19}$$

En el prototipo actual, la función de distribución (f_M) es:

$$f_M() = \sum_{i=1}^{n} (filename[i]) mod(\|EA\|) \tag{4.20}$$

Número de nodos de la partición	Desviación típica
8	0.56
16	0.39
32	0.23
64	0.15
128	0.11

Tabla 4.1: Distribución (desviación típica) de nodos *master* en diferentes particiones distribuidas

Esta función es sencilla y además ofrece una buena distribución de EA_M. La Tabla 4.1 muestra la distribución de EA_M a través de varios EA y la desviación típica que se obtiene al distribuir los ficheros de un sistema de ficheros real compuesto por 256.400 ficheros entre particiones con diferente tamaño. Los resultados indican que el empleo de la función anterior permite una distribución de los EA_M de forma bastante equitativa entre todos los EA de la partición, ofreciendo un buen equilibrio en el uso de los EA.

Debido a que la determinación del EA_M se basa en el nombre del fichero, cuando un usuario efectua una operación de renombrado, cambia el EA_M para ese fichero (ver Algoritmo 4.2). Este proceso se muestra en la Figura 4.5. La única operación necesaria para mantener coherente los metadatos es la transferencia del subfichero de metadatos (S_M) del antiguo EA_M al nuevo.

```
1  rename(oldname, newname) {
2          f_M(oldname, ||EA||) = EA_M
3          f_M(newname, ||EA||) = EA'_M
4          { mover S_M de EA_M a EA'_M }
5          ∀S_i ∈ F_P : { renombrar S_i a newname }
6  }
```

Algoritmo 4.2: Operación de renombrado de ficheros en Expand.

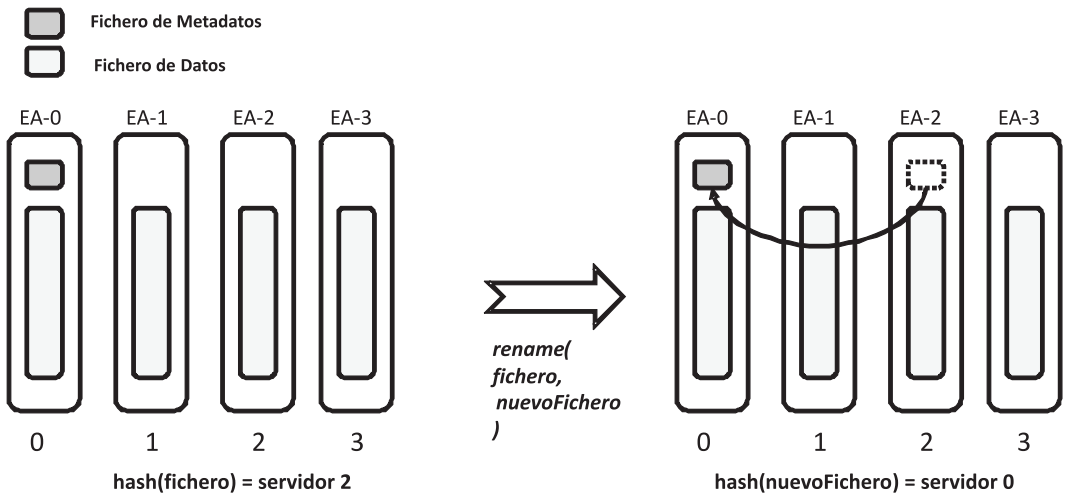

Figura 4.5: Proceso de renombrado de ficheros en Expand.

4.5. Acceso paralelo

Todas las operaciones en Expand utilizan un *manejador virtual*. Un manejador virtual (F) se define como:

$$FH \in F_P = \bigcup_{i=1}^{\|EA\|} fh_i \in S_i \tag{4.21}$$

donde fh_i es el manejador utilizado por el espacio de almacenamiento E_i, para acceder al subfichero de datos S_i del fichero Expand. Cada E_i utilizado proporciona su propio manejador fh_i, que es dependiente del protocolo y servidor utilizados.

Para abrir un fichero, se obtiene el fh_i del subfichero de metadatos (S_M) que reside en el EA_M y que incluirá los metadatos del fichero. Estos metadatos poseen los servidores y la política de reparto de bloques para el fichero. Los manejadores de los subficheros de datos se obtienen bajo demanda cuando se requiere el acceso a los datos de un subfichero. De esta forma se acelera el proceso de apertura de un fichero.

Cuando Expand necesita acceder a los datos de un subfichero, utiliza el manejador que emplea el espacio de almacenamiento donde reside el subfichero. Para mejorar la E/S, Expand realiza, siempre que sea posible, las operaciones en paralelo. Para ello, cuando una petición involucra a k espacios de almacenamiento, se realizan k operaciones en paralelo utilizando *threads*. Así, cuando se emplea NFS como servidor tiene lugar una operación paralela a k servidores que se divide en k operaciones individuales, cada una de las cuales utiliza RPC y el protocolo NFS para acceder al subfichero adecuado. Otro ejemplo de las operaciones en paralelo puede verse en la creación de un fichero en Expand, que implica la creación de varios subficheros de datos. Este proceso se muestra en la Figura 4.6 donde se realiza una operación de lectura que involucra varios bloques de datos y varios EA.

4.6. Reconfiguración dinámica de particiones

Expand permite reconfigurar de forma dinámica una partición añadiendo nuevos nodos servidores a la misma. De esta manera se puede incrementar de forma dinámica el tamaño de las particiones y agregar nuevos recursos existentes. El único problema que introduce la incorporación de un nuevo nodo a una partición existente es la localización del subfichero de metadatos de un fichero de Expand en la nueva partición. En efecto, debido a que la localización del nodo *master* de un fichero en el actual prototipo se basa en el nombre del fichero y el número de servidores de la partición, cuando este número cambia, también lo hace la localización de los metadatos. La única operación necesaria para mantener coherente la partición es reubicar los subficheros de metadatos a su nuevo nodo *master*. La Figura 4.7.a) muestra el efecto de añadir un nuevo nodo a una partición. El fichero en la partición antigua compuesta por tres nodos tenía su subfichero de metadatos en el servidor 2. Al añadir un nuevo nodo, el cálculo de la función de localización de los metadatos da como resultado el servidor 3, en el cual no reside el subfichero de metadatos. Para mantener coherente esta información el subfichero de metadatos se mueve del servidor 2 al servidor 3.

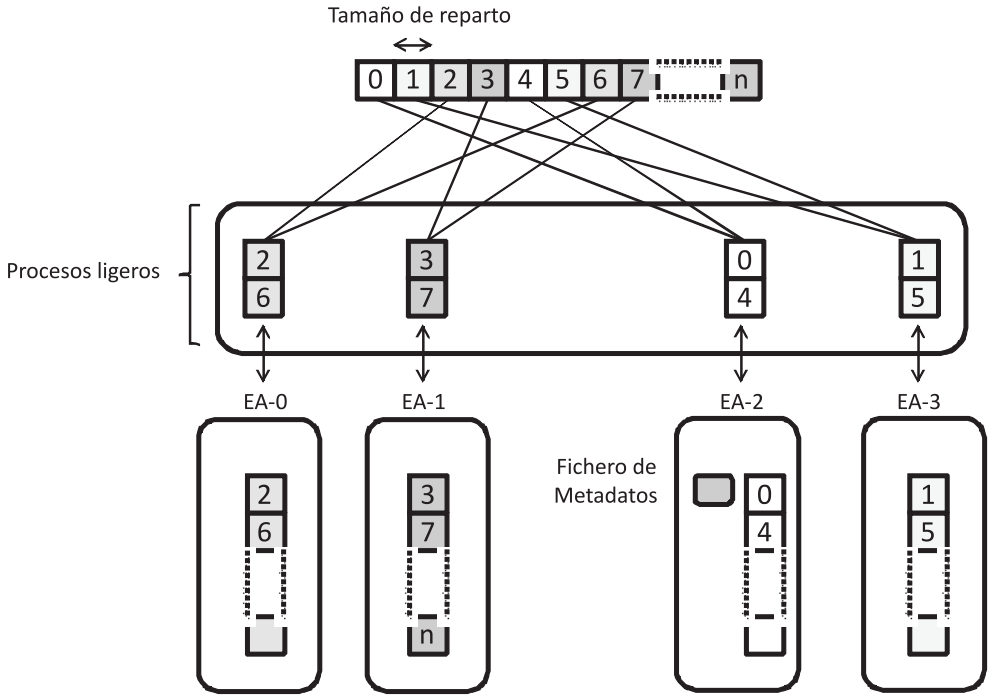

Figura 4.6: Acceso paralelo a los ficheros en Expand.

Esta operación se puede realizar en Expand de dos formas: de manera instantánea para todos los ficheros de la partición cuando se incorpora uno o más nodos servidores nuevos (como se verá en la sección de evaluación, el coste necesario para realizar esta operación no es elevado), o bien, de forma diferida bajo demanda cuando se abre un fichero. En este caso la operación se realiza en la apertura de un fichero.

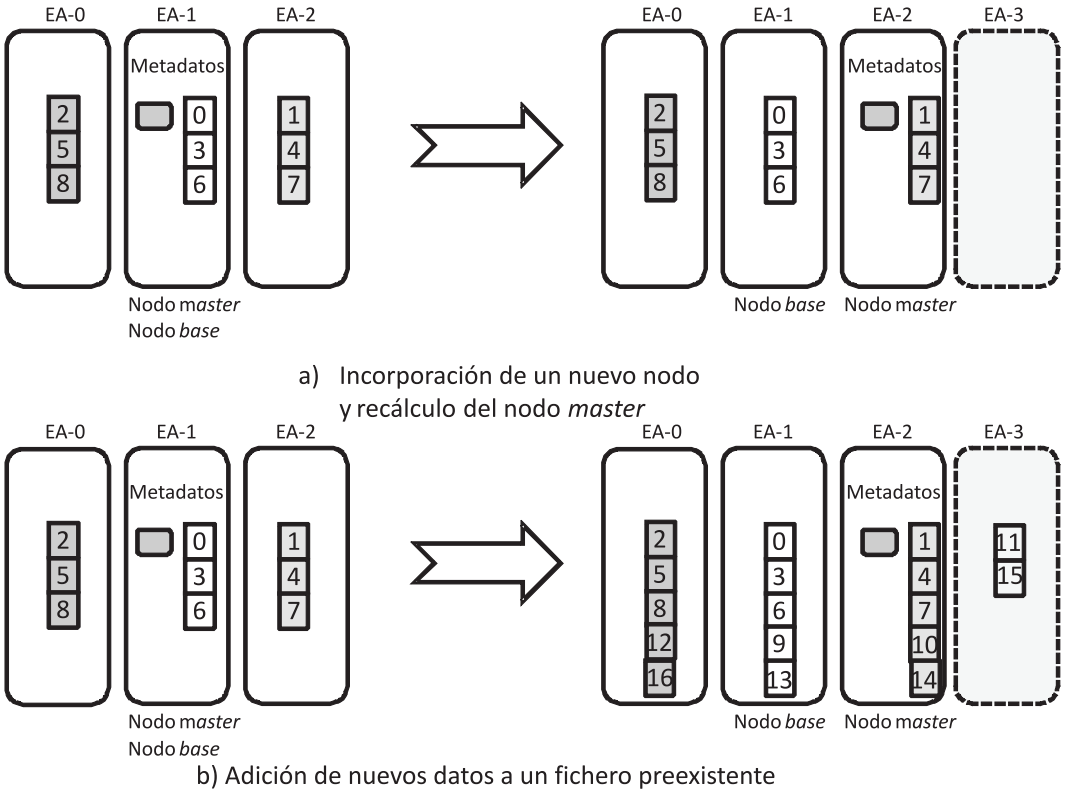

Figura 4.7: Incorporación de un nodo a una partición de Expand.

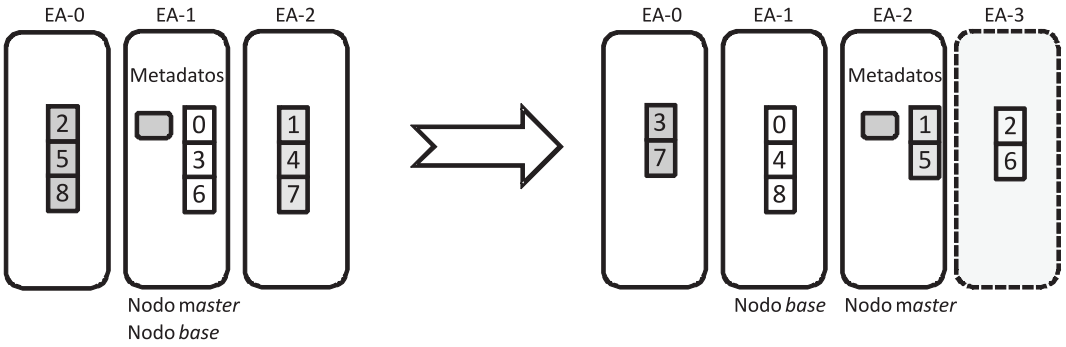

Figura 4.8: Reconstrucción de una partición al incorporar un nuevo nodo.

Si la búsqueda del subfichero de metadatos da un resultado erróneo, se realiza la reubicación del subfichero de metadatos a su nueva localización de forma transparente al usuario. Los nuevos datos que se añadan a un fichero existente utilizarán los nuevos nodos incorporados (véase la Figura 4.7.b). Para este tipo de reconfiguración, es necesaria una nueva función de distribución de datos que permita el uso de distintos patrones de reparto de datos dependiendo de si el bloque es anterior a la incorporación de los datos o posterior. En la Figura 4.7.b) se puede ver que los bloques anteriores a la incorporación del nuevo nodo de E/S, utilizan el patrón de distribución de datos anterior, mientras que los nuevos bloques de datos utilizan el patrón de distribución nuevo. Estos patrones se almacenan en los metadatos del

fichero lo que permite un rapido acceso a los datos, si bien es necesario un control en la modificción de los mismos.

Existe también la posibilidad de reconstruir un fichero o toda la partición cuando se incorporan uno o más nodos. Este proceso para un fichero se muestra en la Figura 4.8. El coste de este proceso en este caso es más elevado ya que conlleva la reubicación de todos los bloques de datos de todos los ficheros pertenecientes a la partición. Por otro lado, se establece una única función de localización de los datos para todos los ficheros de la partición, lo que evita un control sobre los metadatos.

4.7. Autenticación y control de acceso

Para la autenticación y el control de acceso, Expand utiliza los servicios que ofrece el servidor de datos para el control de acceso y la autenticación del ususario en el sistema. Así, por ejemplo, en el caso de NFS, que es un servidor sin estado y no mantiene información sobre los ficheros abiertos por los usuarios, se comprueba la identidad del usuario en cada acceso que se realiza a un fichero. El protocolo de las RPC de Sun requiere que los clientes envíen su información de autenticación en cada petición RPC que realizan al servidor. Esta información es utilizada por el servidor para verificar si se puede acceder al fichero. El actual prototipo de Expand para NFS, utiliza el método de autenticación `AUTH_SYS`. Con este método de autenticación el servidor accede al UID y GID del usuario que realiza la llamada para comprobar si tiene acceso al fichero. El empleo de UID y GID implica que el cliente y el servidor deben compartir la misma asignación de identificadores de usuario. Este requisito se puede resolver utilizando el servicio de nombres NIS.

En caso de otros protocolos de E/S distribuida, como FTP se requiere de una autenticación al inicio de la comunicación. Por ejemplo en FTP es necesario el uso de un identificador y una clave asociada, los cuales deben estar registrados con anterioridad en el servidor.

4.8. Arquitectura de Expand

Una vez establecidos los mecanismos de acceso a los datos, así como la gestión de los datos y metadatos en Expand, pasamos a mostrar una visión general del diseño seguido para la construcción de la arquitectura de Expand.

La arquitectura de Expand se compone de tres capas: *core*, *policy* y *NFI* (véase Figura 4.9), cada una de las cuales se ha diseñado para cumplir una función especifica para la consecución de las múltiples tareas definidas para la realización de las operaciones de E/S en Expand.

Los objetivos principales de cada una de las capas son:

- *core*: definir los algoritmos para la gestión del sistema de fichero paralelo Expand (datos, metadatos, directorios, etc.), con independencia del protocolo

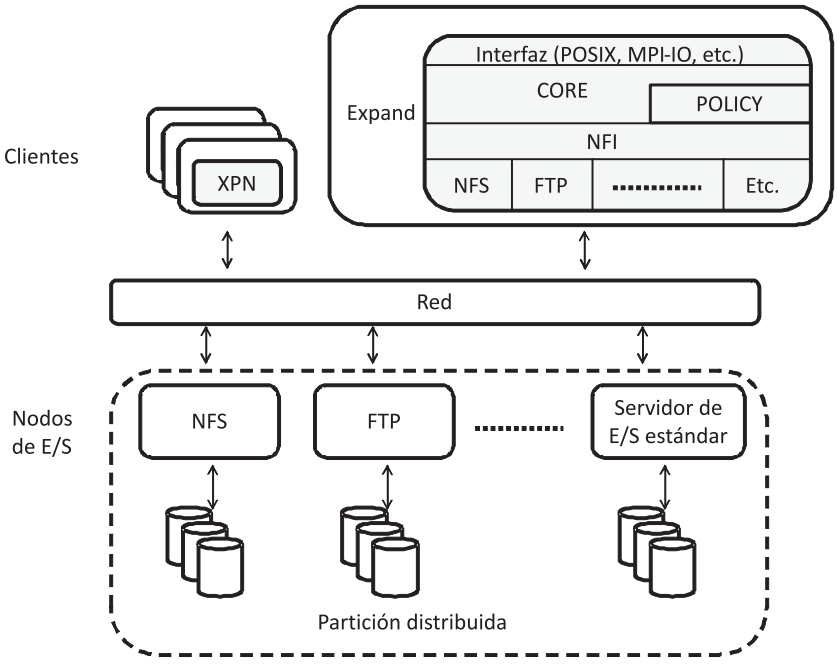

Figura 4.9: Arquitectura de Expand.

empleado en el servidor de datos.

- *policy*: determinar las políticas a seguir en distintos aspectos cruciales dentro de Expand, como son la distribución de los datos, la gestión de los metadatos, etc.

- *NFI*: abstraerlas comunicaciones con el servidor de E/S proporcionando una interfaz única para el acceso a los datos.

A continuación detallaremos con más profundidad cada una de estas capas del sistema de ficheros paralelo.

4.8.1. Capa *Core* o núcleo del sistema

En esta capa de la arquitectura de Expand se realizan las operaciones principales del sistema, se definen los algoritmos básicos para el acceso y la gestión de los datos, y se exporta la interfaz de usuario básica para el uso de Expand. Esta interfaz similar a la disponible en POSIX, abstrae al programador de toda la complejidad del sistema de ficheros paralelo.

Por otro lado, esta capa necesita de las capas inferiores para realizar las operaciones de acceso a los datos, o conocer la distribución de los datos.

Las operaciones que se realizan en este nivel de abstracción son genéricas e independientes de los protocolos y políticas utilizadas en niveles inferiores. Estas operaciones se pueden clasificar según el ámbito de trabajo que tengan:

- Operaciones para el manejo del sistema de ficheros Expand, como son las de inicio del sistema y de eliminación de los recursos utilizados (véase Algoritmos 4.3 y 4.4).

```
1  xpn_init() {
2          para cada P
3          n = ||EA_Pi||
4          P_i = {{N_Pi, T_Pi, S_Pi, ||E_Pi||, ⋃_{j=1}^{n} {EA_j}}
5          para cada EA_ij ∈ P_i
6                  EA_ij = {C_Ei, I_Ei, D_Ei}
7  }
```

Algoritmo 4.3: Inicialización del sistema de ficheros paralelo Expand.

```
1  xpn_destroy() {
2          para cada F_P
3                  {eliminar recursos de F_Pi}
4          para cada P
5                  {eliminar recursos de P_i}
```

Algoritmo 4.4: Operación de eliminación de recursos en Expand.

- Operaciones que sirven para gestionar los objetos del sistema de ficheros paralelo (ficheros y directorios), ya que permiten su creación (véase Algoritmo 4.5), apertura (véase Algoritmo 4.6) y eliminación (véase Algoritmo 4.7) del sistema.

```
1  xpn_creat(filename, mode) {
2          {obtener P del filename}
3          f_M(filename) → EA_M ∈ P
4          {crear S_M en P}
5          por cada EA ∈ P
6                  {crear S_i}
7          por cada S_i ∈ F_P
8                  {obtener fh_i ∈ S_i}
9          fd ← FH ∪ fh_i
10 }
```

Algoritmo 4.5: Operación de creación de fichero.

- Operaciones para el acceso a los datos de lectura (véase Algoritmo 4.8) o escritura (véase Algoritmo 4.9).

```
1  xpn_open(filename, flags [, mode]) {
2          {obtener P del filename}
3          f_M(filename) → EA_M ∈ P
4          por cada EA ∈ P
5                  {obtener fh_i ∈ S_i}
6          {obtener S_M ∈ EA_M}
7          fd ← FH
8  }
```

Algoritmo 4.6: Operación de apertura de fichero.

```
1  xpn_unlink(filename) {
2          {obtener P del filename}
3          f_M(filename) → EA_M ∈ P
4          por cada EA ∈ P
5                  {borrar S_i ∈ F_P}
6          {borrar S_M ∈ EA_M}
7  }
```

Algoritmo 4.7: Operación de borrado de fichero.

4.8.2. Capa *Policy* o de gestión de políticas

Esta capa se sitúa en los niveles intermedios del sistema, y es la encargada de proporcionar a la capa núcleo (denominada también capa *core*) las políticas a utilizar en los casos donde el algoritmo base definido anteriormente dependa de datos externos o de una política establecida por el usuario. Las operaciones que realiza son:

- Definir la distribución de datos en los nodos de E/S (*round-robin*).
- Localizar los nodos de E/S.
- Definir los nodos disponibles que se van a utilizar (por defecto todos los nodos disponibles).
- Realizar las operaciones de nombrado.
- Localizar la función de localización de los metadatos, etc.

También es la encargada de inicializar las capas inferiores según el protocolo establecido en los parámetros de configuración (url ,protocolo, directorios, etc.). Para poder realizar estas tareas, la capa proporciona una interfaz genérica que utiliza el núcleo del sistema de ficheros Expand.

```
1  xpn_read(fd, buffer, size) {
2          {obtener P del fd}
3          por cada EA ∈ P
4                  si fh_i ∩ EA_i = ∅
5                  {obtener fh_i}
6          {dividir buffer en {d,v} cuyo tamaño ≤ ||B_ij|| ∈ P }
7          ∀{d,v} ∈ buffer
8                  f_d(d_k, v_k) ⇐ B_ij ∈ S_i
9  }
```

Algoritmo 4.8: Operación de lectura en paralelo.

```
1  xpn_write(fd, buffer, size) {
2          {obtener P del fd}
3          por cada EA ∈ P
4                  si fh_i ∩ EA_i = ∅
5                  {obtener fh_i}
6          {dividir buffer en {d,v} cuyo tamaño ≤ ||B_i|| ∈ P}
7          ∀{d,v} ∈ buffer
8                  B_ij ∈ S_i ⇐ f_d(d_k, v_k)
9  }
```

Algoritmo 4.9: Operación de escritura en paralelo.

4.8.3. Capa *Network File Interface* o de acceso a los servidores de E/S

Esta capa encargada del acceso a los datos de los distintos servidores de E/S, se denomina *interfaz de ficheros en red* o *Network File Interface* (NFI). Proporciona principalmente dos elementos decisivos en la arquitectura, utilizados por las capas superiores:

- Establece una interfaz única para el acceso a los datos por parte de las capas superiores de la arquitectura abstrayendo las operaciones internas necesarias para realizar las operaciones y la gestión de los servidores de E/S.
- Define los mecanismos necesarios para permitir el paralelismo en las operaciones siempre que se haya establecido el paralelismo en los niveles superiores de la arquitectura.

La interfaz dispone de distintos tipos de operaciones, tanto para el manejo de elementos del sistema de ficheros del servidor de E/S (ficheros o directorios) como para el acceso a los datos almacenados en los mismos. Por otra parte, se han incluido

funciones para la gestión del servidor de E/S con el fin de poder realizar acciones como iniciar o desactivar la comunicación con los servidores de E/S, las cuales son necesarias para el correcto funcionamiento del sistema.

El mecanismo utilizado por NFI para manejar las operaciones realizadas sobre un servidor de E/S es similar al que utiliza Linux con su sistema virtual de ficheros (VFS) [113]. Previamente se ha de disponer de un módulo que implemente todas las operaciones necesarias para realizar la gestión del sistema de ficheros de un servidor de E/S. Este módulo sustituirá las llamadas de la interfaz NFI de tal modo que cuando, desde capas superiores, se realice una operación generica de NFI, esta se realizará en el módulo NFI correspondiente a la implementación deseada.

En Expand, se establece un modulo de NFI por cada servidor de E/S que se utilice, de tal manera que las operaciones entre servidores son independientes entre si. Toda comunicación entre módulos independientes NFI se realiza a través de los niveles superiores, reduciendo de esta manera la complejidad del sistema.

4.9. Interfaz de usuario

Expand ofrece distintas interfaces de E/S: interfaz propia, POSIX, `jExpand` y MPI-IO.

La interfaz propia, es similar a la que existe en POSIX, y permite acceder directamente a los datos disponibles en las particiones de Expand.

Las aplicaciones pueden acceder a las particiones de Expand a través de la integración de la biblioteca Expand mediante el desarrollo de un modulo FUSE (*Filesystem in Userspace*) [108]. Como se muestra en la Figura 4.10, el módulo se encuentra en el espacio de usuario, lo que reduce el rendimiento de las aplicaciones.

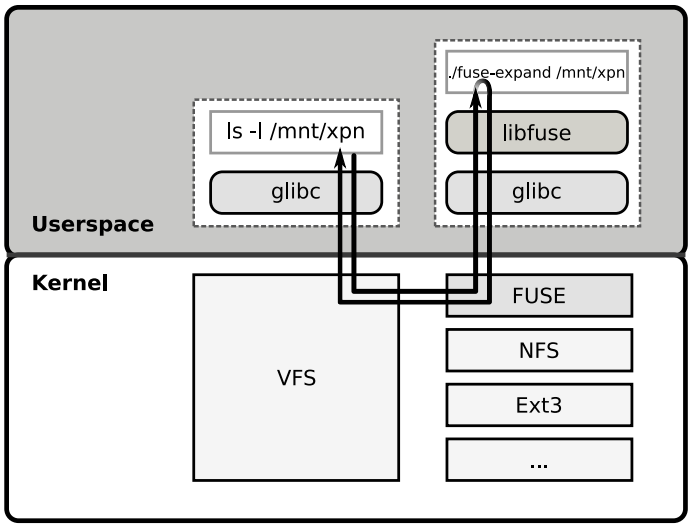

Figura 4.10: Integración de Expand en FUSE.

Támbien se proporciona una interfaz para aplicaciones Java similar a la pro-

porcionada por el estándar, mediante una clase denominada `jExpand` [114] (véase la Figura 4.4). Ésta ofrece grandes ventajas en entornos heterogéneos puesto que se puede utilizar en cualquier computador que disponga de la máquina virtual Java. Las aplicaciones Java que utilizan Expand usan el *Extended Filesystem API* desarrollado por Sun [25]. Esta API ofrece una interfaz común para múltiples tipos de sistemas de ficheros. Es muy adecuada para definir de forma dinámica nuevos sistemas de ficheros e incluye un conjunto de clases similares a las que se encuentran en las clases `java.io` de Java.

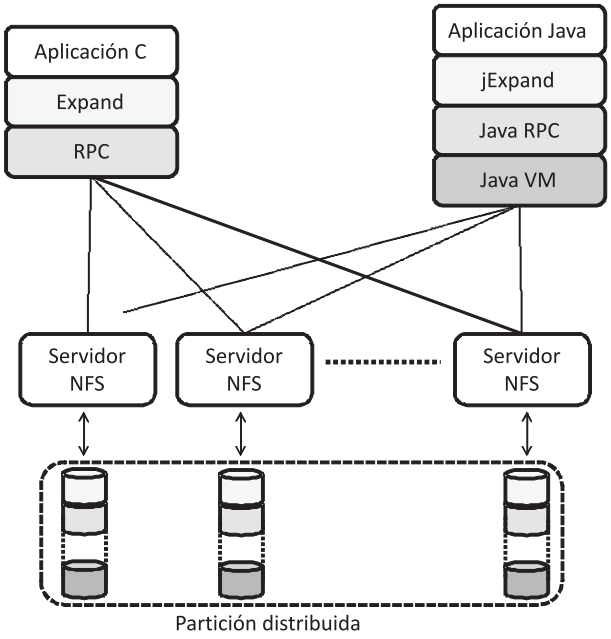

Figura 4.11: Implementación de Expand usando C y Java para entornos Cluster.

Otra interfaz soportada es MPI-IO, la cual es muy utilizada por parte aplicaciones paralelas, que suelen realizar accesos entrelazados y de pequeño tamaño [43].

MPI-IO permite a las aplicaciones MPI definir vistas sobre un fichero. Para ello, se define un *etype* como la unidad de acceso y posicionamiento; un *filetype* que es la base para la división de un fichero entre los procesos y una plantilla para acceder al fichero. Un *filetype* puede estar formado por un tipo de datos sencillo o por un tipo de datos MPI derivado construido a partir de múltiples instancias del mismo *etype*. Una vista (*view*) establece el conjunto de datos real visible y accesible de un fichero como un conjunto ordenado de *etypes*. Cada proceso tiene su propia vista, constituida por tres valores: un desplazamiento, un *etype* y un *filetype*. El patrón descrito por el *filetype* se repite, comenzando en el desplazamiento indicado, para definir la vista.

Esta interfaz se ha incluido en Expand [109] mediante su integración en ROMIO [115] y se puede utilizar con la distribución MPICH (véase la Figura 4.12). La portabilidad se consigue en ROMIO mediante una interfaz abstracta de E/S paralela denominada ADIO [116].

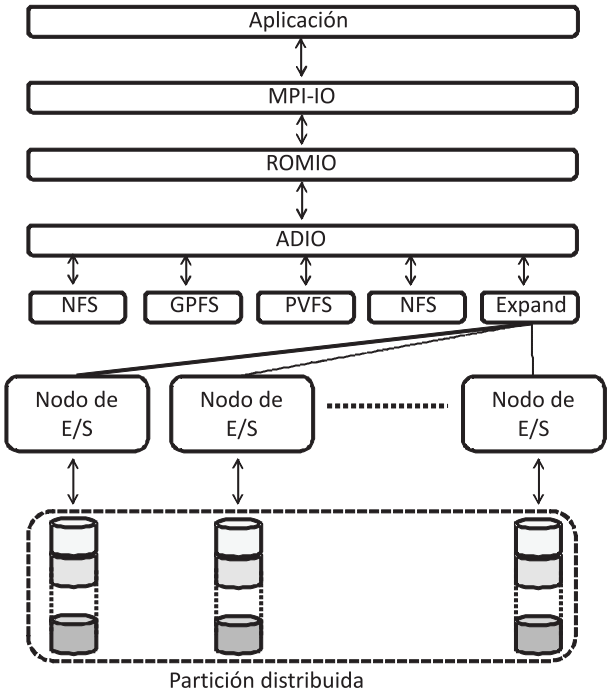

Figura 4.12: Integración de Expand en ROMIO.

La Figura 4.12 muestra la integración de Expand en ROMIO, que es una implementación de MPI-IO que utiliza la interfaz de E/S abstracta ADIO [115]. Esta interfaz es un mecanismo diseñado para implementar interfaces de E/S paralelas portables sobre múltiples sistemas de ficheros. La implementación de ADIO para Expand incluye:

- *Apertura de un fichero.* Todas las operaciones de apertura se consideran colectivas.
- *Cierre de un fichero.* La operación de cierre también es una colectiva.
- *Lecturas y escrituras contiguas.* ADIO ofrece funciones diferentes para acceso a datos contiguos o no dentro del fichero.
- *Lecturas y escrituras no contiguas.* Las aplicaciones paralelas normalmente necesitan leer y escribir datos que no están contiguos en un fichero. ADIO ofrece funciones para acceder a estas regiones utilizando una única llamada.
- *Lecturas y escrituras no bloqueantes.* ADIO ofrece versiones no bloqueantes de todas las llamadas de lectura y escritura.
- *Lecturas y escrituras colectivas.* Una operación colectiva es aquella que debe ser invocada por todos los procesos del grupo que abrieron el fichero.
- *Posicionamiento.* Esta función se emplea para cambiar el puntero de la posición sobre un fichero.

- *Test y wait.* Estas operaciones se utilizan para comprobar la finalización de las no bloqueantes.
- *File Control.* Esta operación se usa para fijar u obtener información sobre un fichero abierto.
- *Otras.* ADIO también ofrece otras operaciones para borrar, cambiar el tamaño de un fichero, etc.

4.10. Implementación utilizando NFS

Para entornos de tipo *cluster* se dispone de una implementación integrada en Expand, que utiliza servidores de datos NFS (*Network File System*) [101]. Aunque fue originalmente diseñado para su uso en sistemas UNIX, hoy en día se encuentra ampliamente disponible para muchos entornos, como Linux e incluso plataformas Windows. La Tabla 4.2 muestra una lista simplificada de las operaciones que exporta el servidor NFS. En estas, los ficheros se identifican mediante una manejador o *file handle*, que es una estructura opaca para el cliente que contiene toda la información que el servidor necesita para distinguir un fichero individual.

Operación NFS	Descripción
lookup(fh, name) → (fh2, attr2)	Busca un fichero en un directorio
create(fh, name, attr) → (fh2, attr2)	Crea un fichero
remove(fh, name) → status	Elimina un fichero
rename(fh, name, fh2, name2) → status	Renombra un fichero
getattr(fh) → attr	Devuelve los atributos asociados a un fichero
setattr(fh, attr) → attr2	Fija los atributos de un fichero
read(fh, offset, count) → (attr, data)	Lee datos de un fichero
write(fh, offset, count, data) → attr	Escribe datos en un fichero
link(fh2, name2, fh, name) → status	Crea un nuevo enlace
symlink(fh2, name2, string) → status	Crea un enlace simbólico
readlink(fh) → string	Obtiene el nombre asociado a un enlace simbólico
mkdir(fh, name, attr) → (fh2, attr2)	Crea un nuevo directorio
rmdir(fh, name) → status	Elimina un directorio
readdir(fh, cookie, count) → entries	Devuelve entradas de un directorio
statfs(fh) → status	Obtiene información sobre el sistema de ficheros

Tabla 4.2: Principales operaciones del protocolo NFS

4.11. Resumen

En este capítulo se ha presentado la arquitectura, el diseño y la implementación del sistema de ficheros paralelo *Expand*.

La motivación principal es ofrecer un sistema de ficheros paralelo para *clusters* heterogéneos que utilice servidores estándar que no precisen de modificación. *Expand* está construido usando como base servidores de datos estándar. Esta solución es muy flexible puesto que no es necesario instalar nuevos servidores para que funcione *Expand*. Además es independiente del sistema operativo utilizado en los clientes debido a que usa el protocolo que ofrezca el propio servidor de datos.

Las razones para utilizar *Expand* como modelo es que es lo suficientemente genérico, ofrece una arquitectura similar al resto de sistemas de ficheros paralelos para *clusters*. Además, el hecho de que no tenga un gestor de metadatos centralizado y de que disponga del árbol de directorios replicado le permite adaptarse mejor a distintos tipos de entornos de computación.

Se han definido la arquitectura del sistema de ficheros paralelos *Expand*, el modelo de partición distribuida y el modelo de fichero paralelo. También se ha descrito el sistema de nombrado, así como el de acceso a los datos o el de gestión de los metadatos. Por otra parte, se han detallado los métodos de control de acceso y de autenticación utilizados en el acceso a los datos dispuestos en las particiones. Por último, se han detallado las interfaces utilizadas por la arquitectura (MPI-IO, POSIX, etc).

Capítulo 5

Arquitectura de sistemas intermedios de almacenamiento

En este capítulo se detallará la propuesta de arquitectura de E/S para grandes *clusters*, que permite mejorar sustancialmente el rendimiento de las aplicaciones y del sistema de almacenamiento, mediante la ampliación de los esquemas de memoria tradicionales a este tipo de entornos usando sistemas intermedios de almacenamiento.

5.1. Descripción de la arquitectura

La Figura 5.1 representa el diseño estándar de la arquitectura de un *cluster*. Se compone de un gran número de nodos de cómputo, de un sistema de almacenamiento formado por un pequeño conjunto de nodos de E/S y al menos una red que intercomunica los nodos del *cluster*. El gran número de nodos de cómputo accediendo al sistema de almacenamiento da lugar a un sistema no equilibrado, convirtiéndose el sistema de almacenamiento en un cuello de botella para las aplicaciones, como ya se indicó en el capítulo 2.

La Figura 5.2 representa una visión lógica de la arquitectura propuesta, denominada *arquitectura de sistemas intermedios de almacenamiento*. La arquitectura emplea el concepto de nodos de almacenamiento intermedio denominados *I/O proxies* (similar al concepto de *proxies* para entornos web) para el almacenamiento y la gestión de los datos manejados por las aplicaciones de un *cluster*.

La arquitectura genera la existencia de dos ámbitos diferentes de funcionamiento, cuyas políticas de gestión se encuentran definidas para la interacción de los distintos elementos que la componen: el primero contiene a las operaciones de E/S realizadas entre los nodos de cómputo y la capa intermedia de E/S, mientras que el

segundo ámbito incluye a las operaciones de E/S producidas entre la capa intermedia y el sistema de almacenamiento.

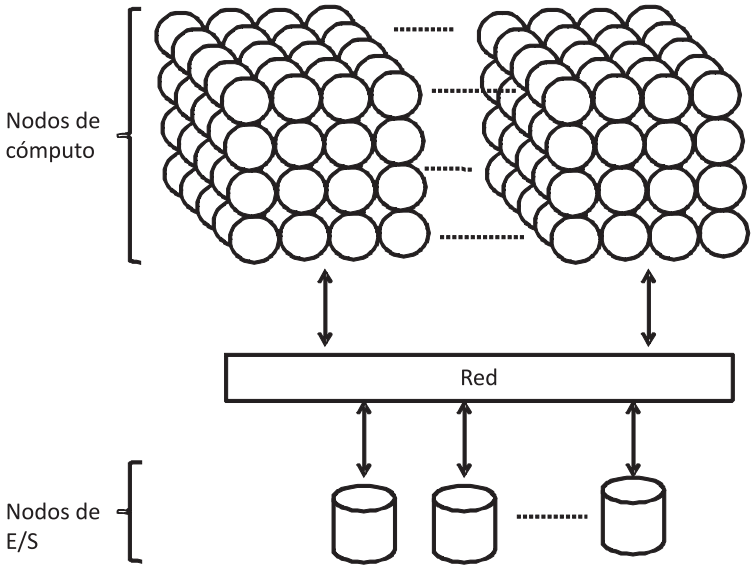

Figura 5.1: Estructura de la arquitectura estándar de un *cluster*.

La arquitectura de almacenamiento intermedio se compone de los siguientes elementos:

- Nodos de cómputo (CN): estos nodos se encargan de la ejecución de las aplicaciones. Disponen de uno o más procesadores, gran capacidad de memoria RAM y algún dispositivo de almacenamiento, como discos duros.
- Nodos de E/S (ION): componen el sistema de almacenamiento del *cluster*. Cada nodo de E/S gestiona uno o más dispositivos de almacenamiento.
- Nodos intermedios de almacenamiento (IOP): un nodo intermedio de almacenamiento o *I/O proxy* es un servidor de datos que ejecuta en un nodo de cómputo del *cluster*. Utiliza algunos recursos como la memoria y los dispositivos de almacenamiento para gestionar las peticiones de E/S de las aplicaciones. Estos nodos intermedios de almacenamiento se agrupan para formar uno o varios espacios de almacenamiento, denominados *espacios virtuales de almacenamiento* (EVA), equivalentes al concepto de partición de Expand. Estos EVA permiten el almacenamiento temporal de los datos manejados por parte de las aplicaciones, distribuyéndolos entre los distintos *I/O proxies* y gestionando las transferencias de datos realizadas sobre el sistema de almacenamiento de un *cluster*.

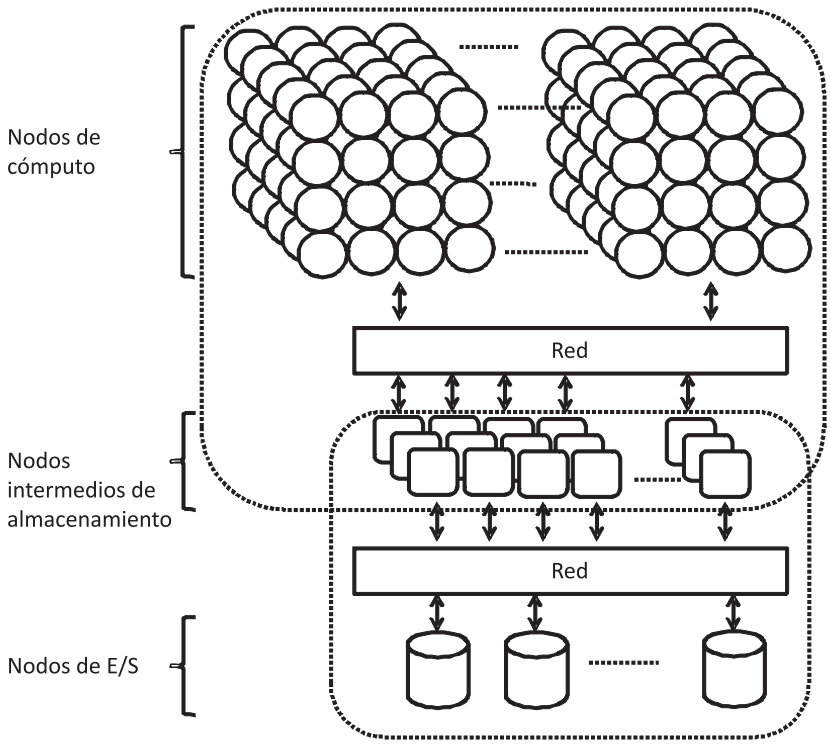

Figura 5.2: Visión lógica de la arquitectura de nodos intermedios de almacenamiento.

La arquitectura de nodos intermedios de almacenamiento funciona de forma equivalente a los sistemas tradicionales de memoria virtual, donde los EVA se comportan como la memoria principal, estableciendo un sistema de ficheros donde las aplicaciones trabajan, mientras que el sistema de almacenamiento del *cluster* representa el sistema de almacenamiento permanente. Al disponer de un espacio limitado de almacenamiento en los IOP, se establecen mecanismos de transferencia de los datos almacenados en estos. Estos mecanismos se encuentran establecidos en los IOP, los cuales gestionan tanto los accesos de E/S producidos por las aplicaciones como las transferencias de datos realizadas entre el sistema de almacenamiento y los IOP. El objetivo principal es que los datos manejados por las aplicaciones se encuentren disponibles en los IOP.

En la arquitectura de nodos intermedios de almacenamiento, se establecen dos ámbitos para la gestión de los datos: el primero comprende a las aplicaciones y los IOP donde se utiliza el sistema de ficheros paralelo Expand para la gestión de los datos almacenados en los EVA. En este caso, para el cliente de Expand, los IOP son servidores de datos, mientras que los EVA son las distintas particiones establecidas en la configuración del SFP. Por otro lado, se establece otro ámbito entre los IOP y el sistema de almacenamiento del *cluster*. En este caso, los IOP hacen uso del sistema de ficheros nativo proporcionado por el sistema de almacenamiento del *cluster* para realizar las transferencias de datos entre ambos. Los IOP pertenecientes a un mismo EVA interactúan entre si usando la interfaz disponible en Expand para la gestión de los datos.

Los EVA, al igual que las particiones en Expand, se identifican mediante etiquetas y se definen por medio de varios parámetros que determinan su configuración: tamaño de reparto usado, número de IOP utilizados, etc., incluyendo los métodos para la gestión de datos en los IOP como políticas de reemplazo, de lectura, etc. Es posible definir varios EVA, estableciendo una restricción para su uso: un fichero sólo puede encontrarse en un EVA a la vez, evitando la posible inconsistencia de los ficheros utilizados.

5.2. Definiciones

Una arquitectura *cluster* tradicional (C) se puede definir como:

$$C \equiv CN \cup ION \tag{5.1}$$

Donde:

- CN representa a los nodos de cómputo de un *cluster*.
- ION compone el conjunto de nodos de E/S de un *cluster*.

$$CN = \{cn_1, cn_2, \ldots, cn_n\} \tag{5.2}$$

$$ION = \{ion_1, ion_2, \ldots, ion_m\} \tag{5.3}$$

Generalmente, el número de nodos de cómputo suele ser mayor que el de nodos de E/S.

$$\|CN\| \gg \|ION\| \tag{5.4}$$

En esta arquitectura, se introduce un elemento más denominado *I/O proxy* (IOP), los cuales se encuentran localizados en los nodos de cómputo. El número de estos IOP es menor o igual al número de CN disponibles en el sistema pero es mayor que el de ION.

$$IOP = \{iop_0, iop_1, \ldots, iop_p\} \tag{5.5}$$

$$IOP \subseteq CN \tag{5.6}$$

$$\|CN\| \geq \|IOP\| > \|ION\| \tag{5.7}$$

Los IOP se agrupan formando espacios virtuales de almacenamiento (EVA), existiendo al menos un IOP por cada EVA:

$$EVA = \{eva_0, eva_1, \ldots, eva_n, \} \tag{5.8}$$

$$\forall iop, iop \in IOP, \exists eva_y, eva_y \in EVA : iop_x \in eva_y \tag{5.9}$$

$$eva_y = \{iop_1, \ldots, iop_n\} \tag{5.10}$$

$$\forall eva_y, eva_y \in EVA, \|eva_y\| > 0 \tag{5.11}$$

$$\forall eva \in EVA, \exists eva_y, eva_z : eva_y \cap eva_z = \emptyset \tag{5.12}$$

Sea SFN el sistema de ficheros nativo que gestiona las peticiones a ION por parte de las aplicaciones disponibles en CN, y sea F_s el conjunto de ficheros disponibles en el sistema de almacenamiento.

$$F_S = \{f_{S1}, f_{S2}, \ldots, f_{Sn}\} \tag{5.13}$$

$$F_S \in ION \tag{5.14}$$

Sea F_I el conjunto de ficheros almacenados en los IOP. Entonces el conjunto total de los ficheros F se puede representar como:

$$F_I = \{f_{I1}, f_{I2}, \ldots, f_{In}\} \tag{5.15}$$

$$F_I \in EVA \tag{5.16}$$

$$F = F_S \cup F_I \tag{5.17}$$

Sobre la arquitectura intermedia se utilizará un sistema de ficheros intermedio SFI que gestionará los datos almacenados en los EVA; en esta tesis se usará Expand como SFI.

Los ficheros F_I tienen una función de correspondencia g con los ficheros F_S. A su vez existe una función g' inversa a la anterior.

$$g : F_I \rightarrow F_S \tag{5.18}$$

$$\forall f_i, f_i \in F_I, f_s \in F_S, \exists f_s : g(f_i) \Rightarrow f_s, g'(f_s) \Rightarrow f_i \tag{5.19}$$

Cuando una aplicación a_i solicita un fichero f_s a S mediante SFN, los datos del fichero son trasladados al EVA correspondiente usando la función g' que determina la distribución de los mismos f_i. Por último, la aplicación a_i hace uso del SFI para acceder a los datos f_i.

Todos los ficheros F_I gestionados por las aplicaciones se encuentran almacenados en los distintos EVA del sistema. Para evitar incoherencias en los datos, se ha establecido como principal restricción que un fichero f_i sólo puede encontrarse en un EVA.

$$\forall eva_x, eva_y \in EVA : eva_x \cap eva_y = \emptyset \tag{5.20}$$

5.2.1. Configuraciones de los sistemas intermedios de almacenamiento

En un *cluster* es posible disponer de uno o varios EVA configurados en el sistema. Estos EVA disponen cada uno de varios IOP, de manera que se pueden configurar distintos esquemas de distribución de datos o de tolerancia a fallos tanto por EVA como por fichero. Las distintas configuraciones de los EVA dependiendo del uso que se realice de los nodos de cómputo donde se encuentran localizados los IOP son:

- Modo exclusivo: en este caso los nodos usados por los IOP únicamente están destinados para la gestión de los datos manejados por las distintas aplicaciones del *cluster*, como se ve reflejado en la Figura 5.3. Esto permite una mejor gestión de los recursos locales por parte de los IOP, si bien reduce los recursos globales del sistema, al disminuir número de nodos de cómputo disponibles por parte de las aplicaciones.

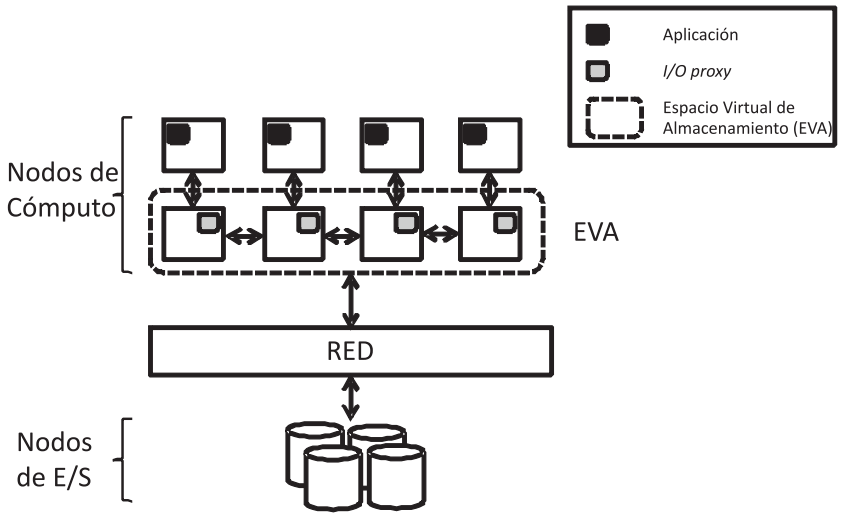

Figura 5.3: Nodos intermedios de almacenamiento usados en modo exclusivo.

- Modo compartido: los nodos de cómputo donde se encuentran localizados los IOP son usados para ejecutar aplicaciones del *cluster*. Esta estrategia, reflejada en la Figura 5.4, permite un mayor aprovechamiento de los recursos del sistema. Además al encontrarse las aplicaciones en el mismo lugar que los datos manejados por éstas, el tiempo de acceso a los mismos es menor al no necesitar operaciones de transferencia a través de la red. Sin embargo plantea problemas, como una mayor limitación en la utilización de los recursos locales de los nodos de cómputo (como la memoria principal o el procesador), aliviados en parte en caso de uso de arquitecturas multicore.

- Modo híbrido: donde se utilizan ambas estrategias presentadas anteriormente, estableciendo una u otra dependiendo de las necesidades del entorno.

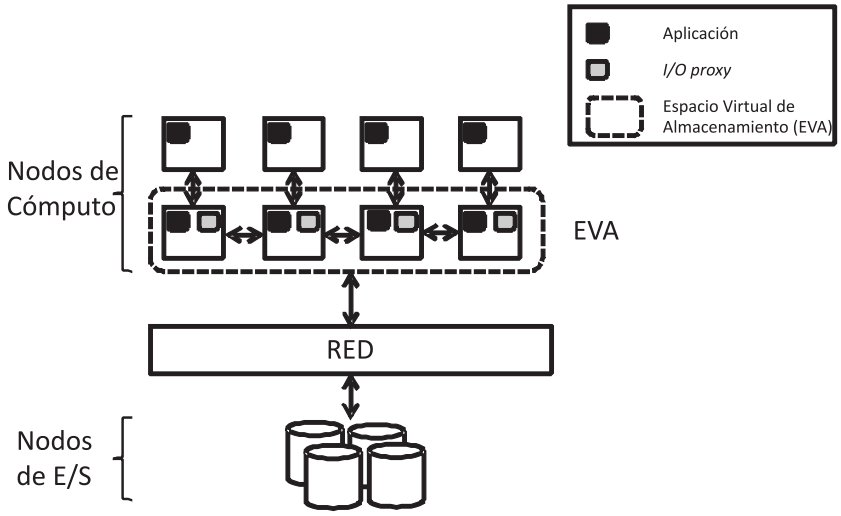

Figura 5.4: Nodos intermedios de almacenamiento usados en modo compartido.

En el capítulo de evaluación se procederá a presentar un estudio realizado usando ambas configuraciones para corroborar las bondades e inconvenientes anteriormente descritos.

Para todos los modos indicados anteriormente, se considera que todas las aplicaciones acceden a todos los EVA establecidos en el sistema. Si bien es posible definir políticas de seguridad para evitar el acceso a algún EVA.

Por otra parte, también es posible definir EVA para aplicaciones concretas donde éstas y los IOP se encuentren localizados en los mismo nodos de cómputo, de forma similar a cómo se definía en el modo compartido, con la diferencia que en este caso los EVA tienen un ciclo de vida asociado a la aplicaciones a las que dan soporte. Este tipo de configuraciones se pueden utilizar para aplicaciones que generen datos que no sean necesarios al finalizar, como ficheros de *checkpointing* o temporales. Estos datos se almacenarían sólo en los IOP de forma temporal, eliminándose al terminar la aplicación. En caso de error, se podría disponer de ellos accediendo a los IOP.

5.3. Arquitectura software

La Figura 5.5 representa a la arquitectura software del sistema de almacenamiento intermedio. Los elementos de la arquitectura son:

- Aplicación: secuencial o paralela que realiza operaciones de E/S para el manejo de datos.

- Biblioteca Expand: se encarga de gestionar los datos almacenados en los distintos *IOP*, mediante un protocolo de comunicaciones disponible en la capa NFI del sistema de ficheros paralelo. Dispone de toda la lógica de gestión del sistema. La biblioteca Expand se encuentra integrada con las aplicaciones, permitiendo el acceso transparente y la gestión de los datos almacenados en los distintos *IOP*.

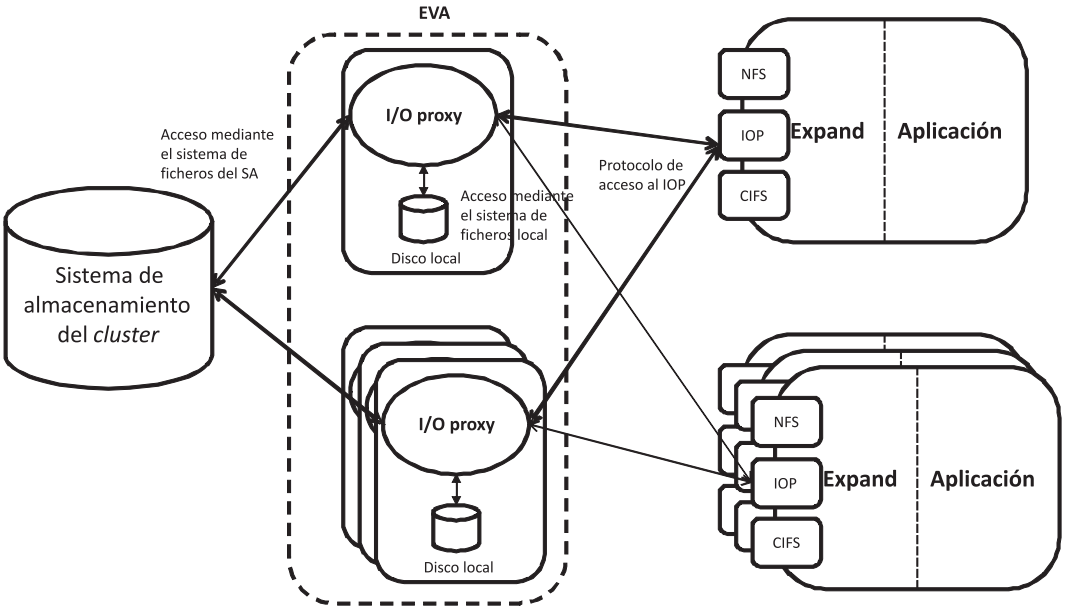

Figura 5.5: Arquitectura software del sistema de almacenamiento intermedio.

- Sistema de almacenamiento: compuesto por los distintos nodos de almacenamiento de los que dispone el sistema. Es accedido de forma transparente mediante el sistema de ficheros nativo del *cluster*.
- *I/O proxy* (*IOP*): servidor de datos localizado en los nodos de cómputo que utiliza el sistema de ficheros local como sistema de almacenamiento temporal de datos. Este servidor creado para la arquitectura propuesta, dispone de tres tipos de comunicaciones dependiendo del tipo de ente con el que se realiza:
  - Con los clientes: a través de un protocolo integrado en la capa NFI del sistema de ficheros Expand.
  - Con el sistema de almacenamiento del *cluster*: los *IOP* realizan la gestión de los datos disponibles en el sistema de almacenamiento del *cluster* mediante el sistema de ficheros nativo, disponible en el sistema operativo de los nodos de cómputo.
  - Con el resto de *IOP*: en la arquitectura los *IOP* realizan operaciones de gestión de datos de forma conjunta a través de la interfaz Expand integrada en los *IOP*, permitiendo el acceso transparente a los datos.

En las siguientes secciones se detallarán los siguientes aspectos de la arquitectura:

- Gestión del espacio de nombres
- Gestión de los metadatos
- Gestión de los datos
- Tolerancia a fallos

5.4. Gestión del espacio de nombres

Las aplicaciones utilizan una ruta de directorios para acceder a los datos del sistema de almacenamiento. La arquitectura de sistemas intermedios de almacenamiento realiza una réplica de estos árboles de directorios en todos los IOP, creando una correspondencia entre el espacio de nombres de un EVA y del sistema de almacenamiento. Esta correspondencia se realiza bajo demanda cuando un fichero es creado o abierto en el EVA. La correspondencia entre el espacio de directorios del EVA permite la ocultación de esta capa a las aplicaciones.

La Figura 5.6 muestra un ejemplo de la replicación del espacio de directorios de los ficheros *f0* y *f3*.

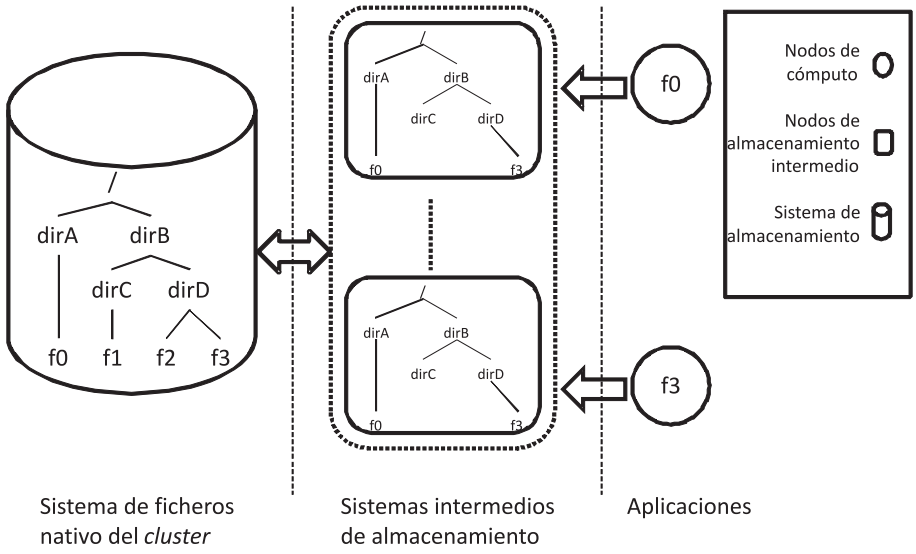

Figura 5.6: Espacio de nombre en la arquitectura de sistemas intermedios de almacenamiento.

5.5. Gestión de metadatos

Los metadatos son gestionados de la misma forma que en el *SFP* Expand, repartiendo estos entre los distintos *IOP*, lo que permite evitar posibles cuellos de botella en el acceso a los mismos. Estos metadatos se generan cuando se crea o se accede a un fichero del sistema de almacenamiento. Contienen la información de un fichero perteneciente a un *EVA*: tamaño de reparto, nodo base (nodo con los datos iniciales del fichero), y el patrón de distribución utilizado para el reparto.

Actualmente la política para el reparto de los datos es cíclica (*round-robin*). Otros metadatos, como el tiempo de modificación/creación o el tamaño de los ficheros, se obtienen mediante operaciones coordinadas entre los distintos *IOP*.

Los metadatos se pueden almacenar en varios *IOP* pertenecientes al mismo *EVA* para tener un mayor soporte de tolerancia a fallos.

5.6. Gestión de datos

En la arquitectura presente, todos los datos manejados por las aplicaciones, que se encuentran ejecutando en un *cluster*, son almacenados en algún momento en los distintos *IOP*. Se disponen de políticas de gestión de los datos para el almacenamiento de estos, de tal manera que siempre se encuentren disponibles en los *IOP*. Sin embargo, estos *IOP* disponen de un espacio limitado de almacenamiento, por lo que son necesarias políticas de reemplazo. En esta sección se detallarán las políticas diseñadas para la arquitectura presentada.

5.6.1. Políticas de gestión de datos en los *EVA*

Tienen como misión:

- mantener la coherencia de los datos entre el sistema de almacenamiento y los datos disponibles en los *IOP*,
- y decidir cuándo se realiza la transferencia de los datos entre el sistema de almacenamiento y los *IOP*.

Se pueden clasificar por el tipo de operación empleada en un fichero:

- Lectura de datos: definen los tiempos y el modo de actuación en las operaciones.
- Sincronización de datos, donde se determinan las estrategias utilizadas para evitar incoherencias entre los datos almacenados en los *IOP* y los disponibles en el sistema de almacenamiento del *cluster*.

Para coordinar estas políticas (de lectura y sincronización), se establece por cada fichero un *iop* del conjunto de *IOP* de un *EVA*, denominado *I/O proxy maestro*

(IOP_M). Este IOP_M se escoge utilizando una función de distribución a partir del nombre (al igual que ocurre en el SFP Expand). La aplicación puede indicarle a este IOP_M que políticas utilizar mediante el uso de *hints* (o pistas). En caso de no indicarse ninguna, se dispone de políticas globales por defecto.

5.6.1.1. Políticas de lectura de datos

La Figura 5.7 representa un esquema del funcionamiento de una operación de lectura sobre un EVA compuesto por varios IOP. En la Figura 5.7(a), una aplicación realiza una llamada de apertura de un fichero sobre el IOP_M usando el SFP Expand. Si el fichero no se encuentra almacenado en el EVA, como se ve reflejado en la Figura 5.7(b), se transfiere el fichero del sistema de almacenamiento del *cluster* al EVA.

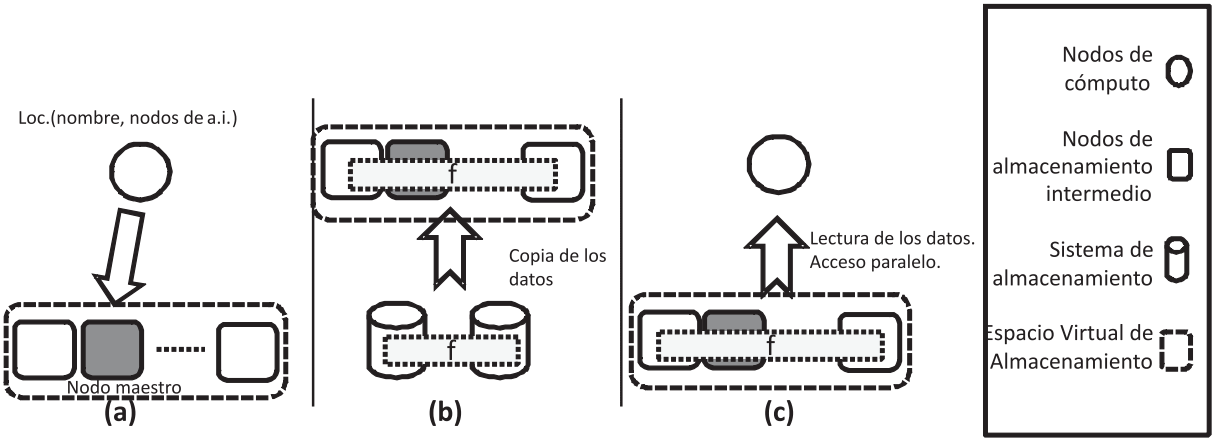

Figura 5.7: Políticas de lectura de datos cuando el fichero no se encuentra en los I/O *proxy*.

La distribución de los datos es posible mediante dos políticas distintas de transferencia:

- Política coordinada de transferencia, donde el IOP_M gestiona la transferencia del fichero. Se encuentra reflejada en la Figura 5.8. En este caso, el IOP_M realiza operaciones de lectura de los datos disponibles en sistema de almacenamiento del *cluster* a través del sistema de ficheros nativo, almacenando los datos en memoria para después distribuidos en los IOP pertenecientes al EVA mediante la interfaz proporcionada por Expand.
- Política de transferencia distribuida, donde cada uno de los IOP realiza las operaciones necesarias de E/S de forma independiente, permitiendo el paralelismo en las operaciones de transferencia, como muestra la Figura 5.9. El IOP_M distribuye a cada uno de los IOP los patrones de acceso a los datos que deben almacenar en cada uno de ellos. Esta estrategia permite una transferencia más rápida de los datos, pero aumenta la carga en el sistema de almacenamiento.

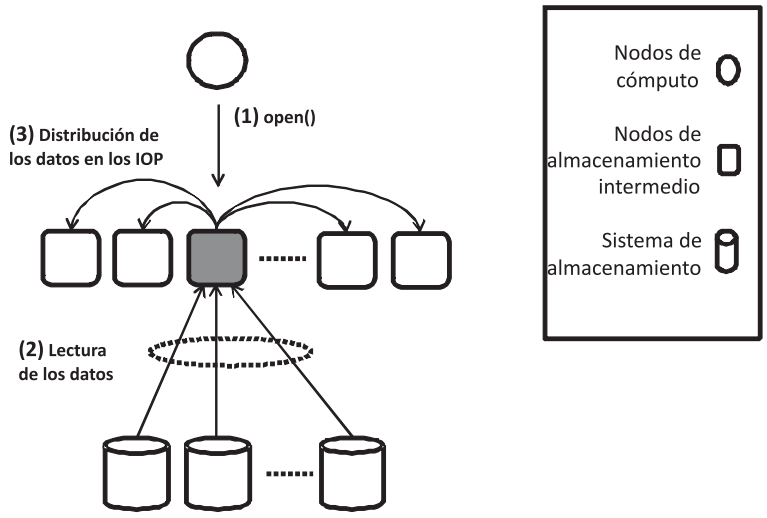

Figura 5.8: Política coordinada de transferencia de datos.

Una vez que el fichero se encuentra en el EVA, la aplicación realiza las operaciones de lectura sobre el EVA, como muestra la Figura 5.7(c). Otras políticas de transferencia que deciden el momento y la cantidad de datos que se pueden utilizar son: leer datos bajo demanda, al inicio de la aplicación o en la apertura del fichero (política por defecto).

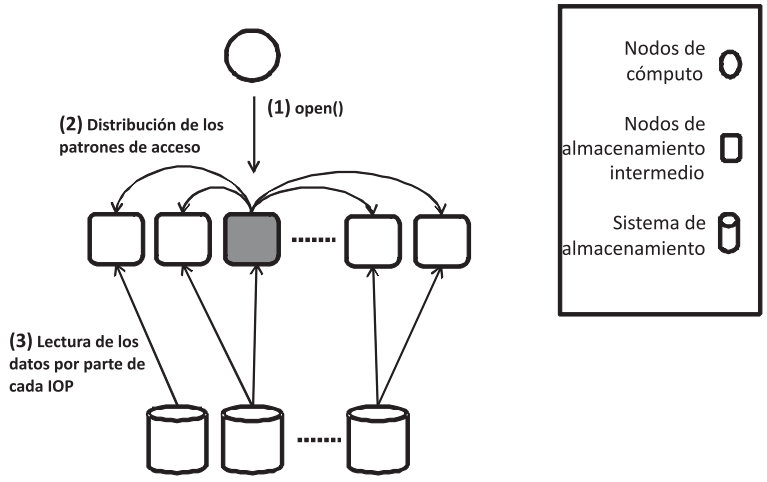

Figura 5.9: Política de transferencia distribuida de datos.

5.6.1.2. Políticas de sincronización de datos

Los IOP garantizan que todos los datos creados o modificados almacenados en un EVA se encuentran en algún momento en el sistema de almacenamiento.

Existen distintas políticas para realizar la sincronización de ficheros entre los IOP y el sistema de almacenamiento con el fin asegurar esta premisa. Los datos modificados o creado en los EVA se transfieren dependiendo de *hints* establecidos por parte de las aplicaciones o por políticas globales definidas por los administradores del sistema. Los ficheros transferidos pasan a ser no modificados, a efectos de políticas de reemplazo.

Las políticas de sincronización de datos definidas son las siguientes:

- Escritura al cierre, en la que un fichero es transferido en el momento en que la aplicación o aplicaciones realicen la operación de cierre del fichero. Existen dos posibles estrategias a la hora de realizar la transferencia en base a esta política, dependiendo del comportamiento de la aplicación:
  - Síncrona, la aplicación espera la transferencia completa del fichero en la operación de cierre del fichero. Esto asegura la sincronización del fichero, pero tiene un coste en el rendimiento de la aplicación.
  - Asíncrona, la aplicación indica a los IOP que se realice la transferencia, pero no espera a que esta operación termine. Esta estrategia permite independizar a la aplicación de los datos, mejorando el rendimiento de la misma, sin embargo no puede asegurar la sincronización de los datos.
- Escritura retardada, realiza una transferencia de los datos cada vez que se cumple un tiempo determinado definido en la configuración del EVA o cuando no existe espacio libre en el EVA. Esta transferencia se lleva a cabo de forma autónoma por parte de los IOP, sin ningún tipo de inferencia por parte de las aplicaciones.
- Sincronización bajo demanda, en la que se transfieren los datos al sistema de almacenamiento según llegan al EVA. Esto permite sincronizar los datos en el EVA con los del sistema de almacenamiento, lo que aumenta el tiempo por operación, y la convierte en la política menos recomendable para realizar la sincronización.

Ambas políticas reducen el número de operaciones de E/S sobre el sistema de almacenamiento al realizar un agrupamiento de los datos en los IOP, incrementando el rendimiento del sistema. Además, al delegar la carga de E/S a los IOP, disminuye el tiempo necesario para realizar una operación de E/S por parte de las aplicaciones.

Las políticas de transferencia son indicadas a las indicadas anteriormente para la lectura de datos.

5.6.2. Políticas de reemplazo

La capacidad de un EVA viene dada por la suma de las distintas capacidades de almacenamiento de los distintos IOP que la componen, siendo esta limitada. Cuando se requiere de espacio libre en el EVA para almacenar un nuevo fichero,

ya sea por la apertura de un fichero no disponible en el EVA o por la creación de un nuevo fichero, es necesaria una política de reemplazo que evite movimientos innecesarios entre el sistema de almacenamiento y el EVA.

Para establecer el orden de eliminación de ficheros del EVA se han establecido diversos atributos de los ficheros: modo de acceso a los datos (escritura/lectura), fecha de creación, fecha de modificación, fecha de acceso, etc. Esto permite diseñar distintos algoritmos para la eliminación de los ficheros de un EVA.

El algoritmo 5.1, utiliza una política LRU pura para la selección de los ficheros del EVA. Este algoritmo no tiene en cuenta el coste a la hora de sincronizar los datos modificados en el EVA.

```
1  eliminar_fichero_por_fecha() {
2          {ordenar por fecha de acceso F_i ∈ eva}
3          {mientras el espacio de eva no sea suficiente}
4                  {sincronizar g(f_i) ⇒ f_s}
5                  {eliminar  f_i ∈ eva  donde  fecha(f_i) > fecha(f'_i)}
6  }
```

Algoritmo 5.1: Operación de eliminación de los ficheros discriminando por fecha de acceso.

El algoritmo 5.2, utiliza una política LRU modificada para la eliminación de los ficheros del EVA que tiene en cuenta el modo de operación utilizado para acceder a los datos. Esta estrategia evita la sincronización de los datos, lo que reduce los tiempos en la eliminación de los mismos.

```
1  eliminar_fichero_por_modo() {
2          {ordenar F_i ∈ eva}
3                  {f_i no modificados por fecha}
4                  {f_i modificados por fecha}
5          {mientras el espacio de eva no sea suficiente}
6                  {sincronizar g(f_i) ⇒ f_s}
7                  {eliminar  f_i ∈ eva}
8  }
```

Algoritmo 5.2: Operación de la eliminación de los ficheros discriminando por modo de acceso.

.

En caso de no existir espacio después de la ejecución se procede a indicar a la aplicación que funcione en el denominado *modo degradado*. Este modo consiste en el acceso a los datos de forma directa al sistema de almacenamiento del sistema. Si bien para establecer un control del mismo, cada aplicación que funcione en el

modo degradado ha de indicar el acceso a los datos al EVA correspondiente en cada operación de creación, apertura y cierre. Esto permite controlar los ficheros accedidos en *modo degradado*, evitando problemas de coherencia entre los EVA y el sistema de almacenamiento en caso de accesos por parte de múltiples aplicaciones.

5.7. Sistemas de replicación

La arquitectura de IOP, basa su tolerancia a fallos en la disponibilidad de los nodos de cómputo del *cluster*. Los nodos de cómputo tienen una disponibilidad limitada lo que puede provocar perdidas de los datos manejados por parte de las aplicaciones. Para evitar esta posible perdida de datos se han definido varios modelos de tolerancia a fallos basados en la replicación de los datos.

5.7.1. Esquemas de replicación de datos

Los ficheros que se encuentran modificados en los EVA, pueden provocar una incoherencia de los datos en caso de uno o varios fallos de distintos nodos de cómputo donde se encuentren localizados los IOP. Para evitar este tipo de problemas se han establecido distintos esquemas de replicación de datos:

- Replicación interna: un bloque de datos (b) de un fichero f_i se asigna a distintos *iop* pertenecientes a un mismo *eva*, mediante una función de distribución f_B.

$$\exists iop_i, iop_j \in eva : f_B(b) \rightarrow \{iop_i, iop_j\} \tag{5.21}$$

  Mediante esta estrategia de replicación, representada en la Figura 5.10, se pueden situar datos en distintos IOP de un mismo *eva*, permitiendo un mayor aprovechamiento de los recursos del *eva*. La coherencia de los datos es fácil de gestionar al encontrarse todos los datos (originales y réplicas) en un mismo *eva*. El mayor inconveniente es el uso de una gran cantidad de recursos de almacenamiento (mayor espacio ocupado en cada uno de los *iop*).

- Replicación distribuida: un bloque de datos (b) de un fichero f_i se asigna a distintos *iop* pertenecientes a distintos *eva*, mediante una función de distribución f_B.

$$\exists iop_i \in eva_i, iop_j \in eva_j : f_B(b) \rightarrow \{iop_i, iop_j\} \tag{5.22}$$

Mediante esta estrategia, representada en la Figura 5.11, podemos situar datos en distintos EVA, los cuales pueden disponer de distintas características (tamaño de reparto, número de IOP, etc) lo que permite una gran flexibilidad. Además, este sistema de replicación permite realizar operaciones de balanceo de carga. Como grandes inconvenientes tenemos el uso de un mayor número de recursos (IOP) y

una mayor dificultad en la gestión de la coherencia de los datos, en caso de verse modificados.

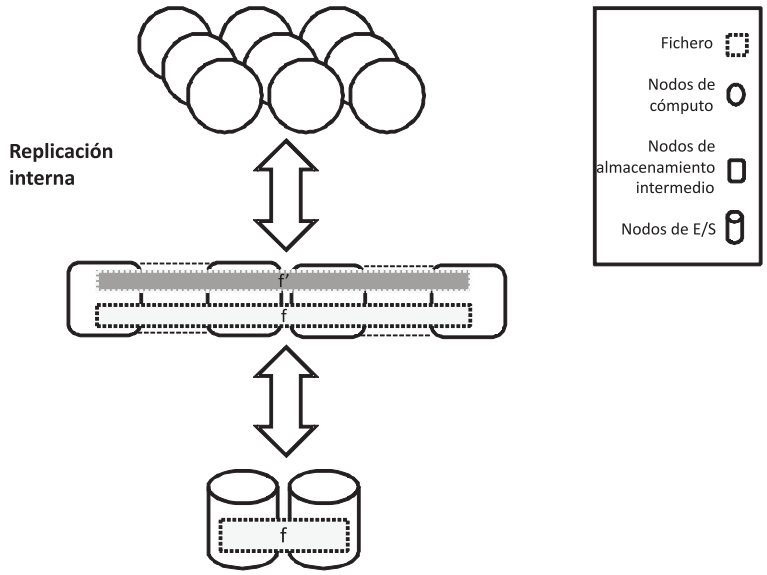

Figura 5.10: Esquema de replicación interna donde f y f' tienen distinto patrón de distribución.

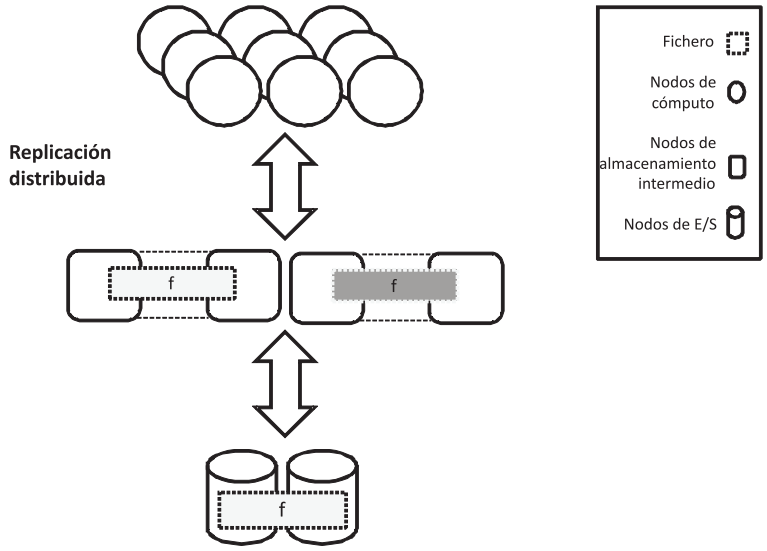

Figura 5.11: Esquema de replicación distribuida usando IOP independientes.

Los IOP gestionan la replicación de los datos, realizando las operaciones de actualización y modificación de forma autónoma, desacoplando la complejidad de

estas operaciones a las aplicaciones. Los IOP establecen mecanismos de control en el acceso a los datos modificados, realizando las réplicas atómicamente para evitar incoherencias entre aplicaciones que acceden a estos datos. La Figura 5.12 representa el proceso de modificación de un dato usando replicación interna.

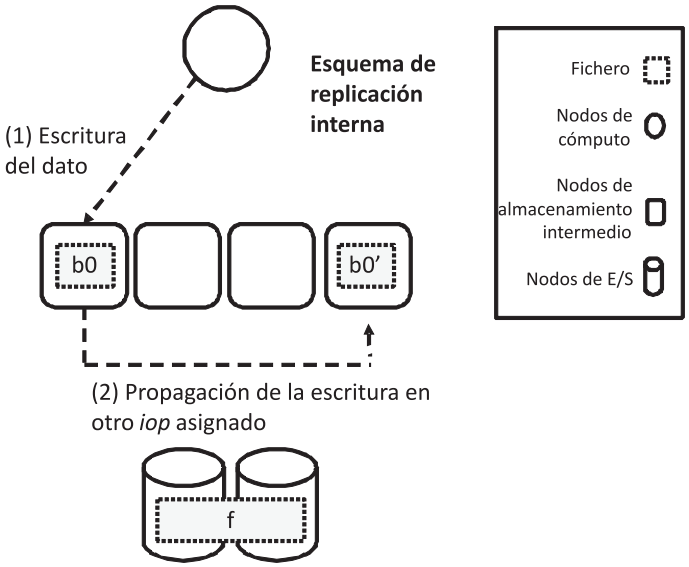

Figura 5.12: Sistema de propagación de las réplicas.

En caso de fallo de algún *iop*, este esquema permite que las aplicaciones que accedan a estos datos dispongan de, al menos, una replica de los datos del fichero repartidos a los largo de los distintos IOP. En el momento que un cliente detecta el fallo en un *iop*, se puede realizar una operación de localización del dato solicitado usando un nuevo patrón de distribución, que sustituirá al anterior.

El empleo de estos esquemas de replicación de datos reduce el rendimiento de las aplicaciones en las operaciones de escritura de datos, al necesitar de al menos una transferencia de datos y en algún caso, la lectura y modificación del bloque transferido. Sin embargo, estos esquemas no afectan al rendimiento de las aplicaciones que realizan operaciones de lectura sobre los mismos datos.

Por último, para los metadatos del SFI, al gestionarse como datos, es posible usar estos mismos esquemas de replicación para evitar fallos en el acceso a los metadatos, simplificando y unificando los sistemas de replicación de la arquitectura.

5.7.2. Esquema de acceso degradado a los datos

Además del sistema de réplicas, se dispone de otro mecanismo de tolerancia a fallos que se denomina *modo degradado*. Este sistema se basa en el uso de los datos disponibles en el sistema de almacenamiento cuando se detecte un fallo en

un *iop*, sin necesidad de replicar los datos o limitar el acceso al resto de IOP. Este esquema únicamente se encuentra disponible para aquellos datos que no se han visto modificados por parte de las aplicaciones (es decir, sobre aquellos ficheros sobre los que se ha utilizado un modo de acceso de *sólo lectura*).

La Figura 5.13 representa el funcionamiento de este esquema de acceso. En la Figura 5.13(a), el cliente intenta acceder a los datos de un *iop* que no funciona. Al detectar el fallo, el cliente procede a realizar la operación de E/S sobre el sistema de almacenamiento del sistema, como se muestra en la Figura 5.13(b).

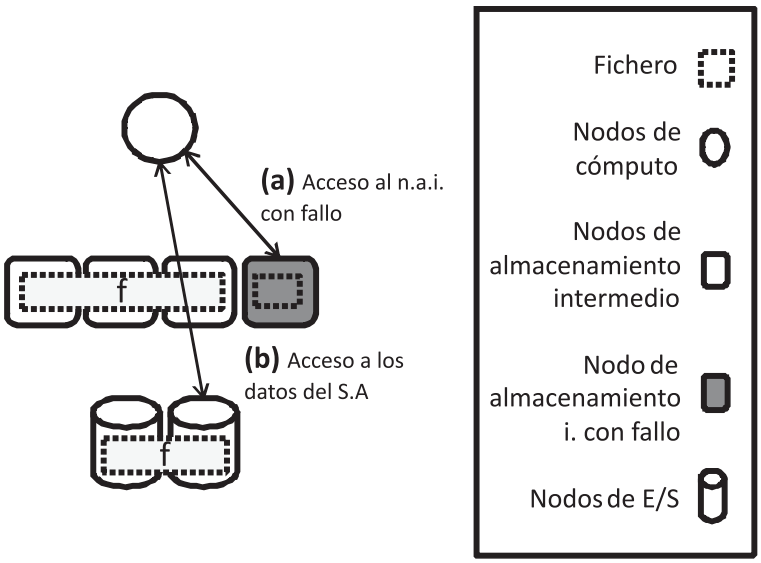

Figura 5.13: Modo degradado de funcionamiento.

Este método de acceso a los datos permite la escalabilidad de la arquitectura de sistemas intermedios de almacenamiento, ya que el proceso de acceso a los datos se puede utilizar con múltiples fallos en los IOP, siguiendo el algoritmo mostrado en la Figura 5.13 por cada uno de los accesos realizados sobre un *iop* con fallo. Sin embargo, este modo de funcionamiento en el acceso a los datos, produce una disminución del rendimiento de los clientes al necesitarse dos accesos para poder realizar una operación sobre datos. En caso de incrementarse el número de fallos de IOP, sobrepasando un limite indicado en la configuración del sistema, es posible pasar al modo nativo de acceso a los datos disponibles en el sistema de almacenamiento del *cluster*. De igual manera, si los IOP se recuperan de los fallos, es posible pasar al modo original de acceso a los datos gestionados por los IOP.

5.7.3. Esquemas de reconstrucción de datos

En el caso de los ficheros no modificados, no es necesaria la implantación de esquemas de replicación de datos, ya que existe una replicación implícita de los datos

en el sistema al disponerse de una copia siempre en el sistema de almacenamiento del sistema. Además, al no emplear esquemas de replicación de datos para este tipo de ficheros, las capacidades de almacenamiento de los IOP no se ven afectadas.

Para este tipo de ficheros se han considerado otras estrategias para la recuperación de los datos en caso de algún fallo en uno o varios *iop*, como se muestra en la Figura 5.14(a):

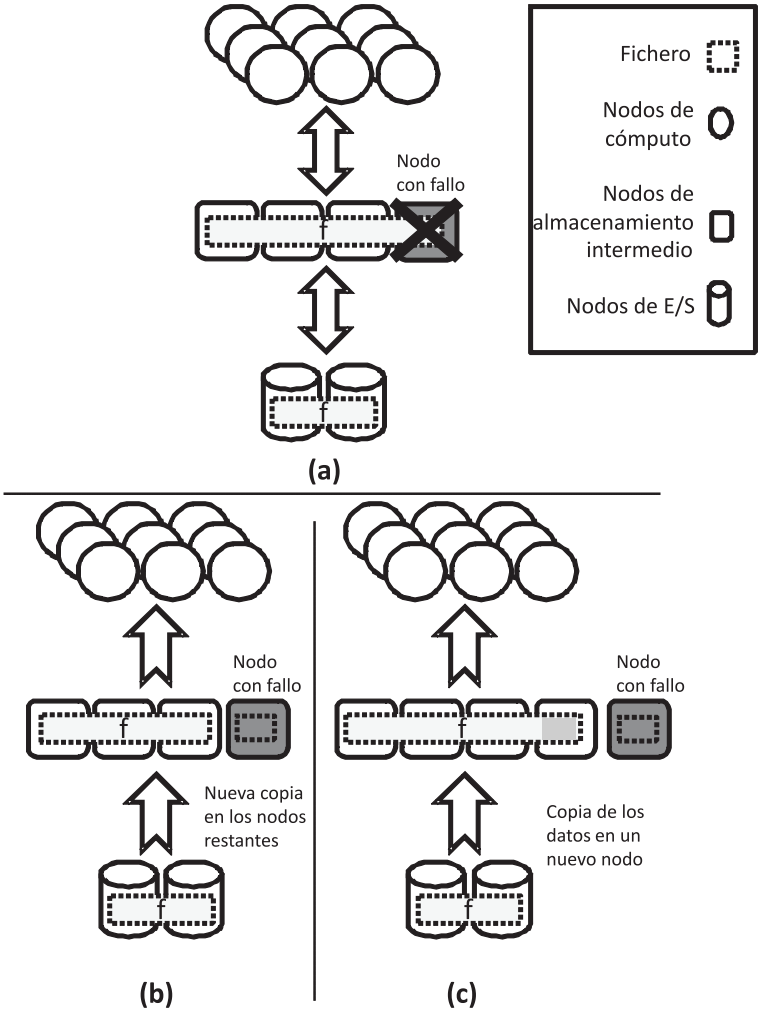

Figura 5.14: Esquemas de tolerancia a fallos para ficheros usados en modo lectura.

- Nueva copia de los datos, como muestra la Figura 5.14(b). En este caso, se procede al movimiento de los datos sobre los nodos restantes, eliminando o moviendo los datos anteriormente almacenados en cada uno de los IOP. La principal ventaja de esta estrategia es la simplicidad de la solución. Sin embargo, este tipo de reconstrucción de los datos tiene un gran coste en tiempo y recursos, siendo el tiempo proporcional al tamaño de los datos en movimiento.

- Sustitución del nodo con fallos, como aparece en la Figura 5.14(c). En este caso, se procede a una reconstrucción en un *iop* de forma autónoma de los datos perdidos anteriormente. Esta estrategia mejora sustancialmente el rendimiento del sistema al mantener los datos disponibles en los otros IOP. Como mayor inconveniente, el *iop* precisa de un conocimiento completo de la composición del EVA y de los metadatos del fichero, como el tamaño y la política de reparto de los datos, etc.

5.8. Implementación

Para la creación de la arquitectura propuesta, se ha realizado un prototipo basado en el sistema de ficheros paralelo Expand, puesto que permite incorporar nuevos servidores de datos sin que la aplicación cliente se vea afectada.

Para el desarrollo del prototipo se ha diseñado e implementado un servidor de datos (*iop*) que gestionará las peticiones de los clientes, estableciendo las políticas y estrategias definidas anteriormente. Además se han establecido mecanismos para la gestión de los datos almacenados localmente en los dispositivos de almacenamiento del nodo de cómputo donde se encuentre localizado dicho servidor de datos. Posteriormente se han definido los mensajes y las estructuras de comunicación usadas entre el cliente del SFP Expand y el *iop*. Esta comunicación se realizará mediante mecanismos fiables para la transmisión de datos utilizando el protocolo TCP para su desarrollo.

Los IOP replican el espacio de nombre bajo demanda, usando el sistema de ficheros local para ello. Por otra parte, los metadatos del sistema son tratados como datos por parte de los IOP, siendo gestionados por los clientes de Expand integrados en las aplicaciones.

En la biblioteca cliente del SFP Expand se ha incorporado un nuevo protocolo para poder acceder a los IOP. Esta implementación define las operaciones básicas de la capa NFI (indicadas en el capítulo anterior). E incorpora dos más: *preload* o precarga de datos solicitados por parte de las aplicaciones, y *flush* o volcado de datos disponibles en los IOP. Mediante estas sencillas operaciones, el cliente puede definir el comportamiento en la transferencia de datos entre los IOP y el sistema de almacenamiento del *cluster*.

La gestión de los datos distribuidos en los distintos IOP pertenecientes a un mismo EVA es posible mediante la integración de un cliente de Expand en los IOP. Esto facilita el acceso a los datos y el desarrollo de la arquitectura. Para la interacción con el sistema de ficheros local, el *iop* hace uso del sistema de ficheros nativo proporcionado por el sistema de almacenamiento. La Figura 5.15 representa los distintos niveles o capas de gestión integrados en un *iop*.

Además se ha incorporado una operación más en la capa NFI de Expand, cuyo objetivo es indicar los patrones de acceso necesarios por parte de los IOP para realizar una transferencia de datos distribuida. Estos patrones disponen de los siguientes elementos:

```
precarga_de_datos() {
        {el cliente c solicita a iop_M ∈ eva un dato de f_i}
        {si f_i ∉ eva}
                {reservar espacio en eva para f_i}
                {mover f_s → f_i ∈ eva}
        {iop_M ∈ eva indica al cliente c la disponibilidad
           de los datos}
}
```

Algoritmo 5.3: Operación de precarga de datos.

```
volcado_de_datos() {
        {el cliente c solicita a iop_M ∈ eva sincronizar f_i}
        {mover f_i → f_s ∈ eva}
        {iop_M ∈ eva indica al cliente c el fin de la
           operación}
}
```

Algoritmo 5.4: Operación de volcado de los datos.

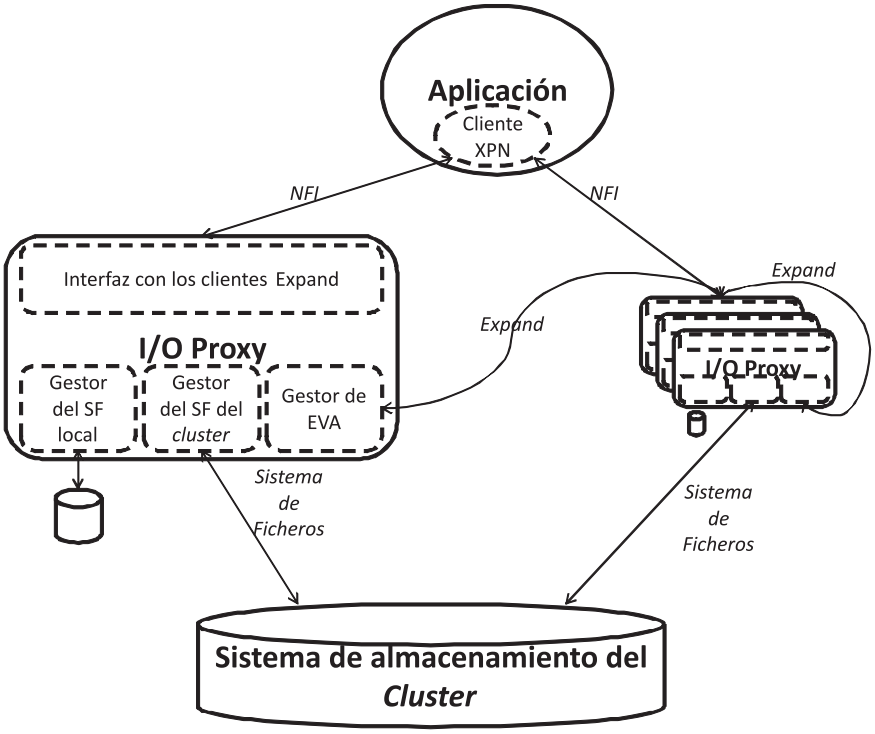

Figura 5.15: Niveles de administración de un *iop*

- Nombre del fichero donde se encuentran los datos.
- Desplazamiento dentro del fichero local almacenado en el *iop*.
- Desplazamiento dentro del fichero perteneciente al sistema de almacenamiento del sistema.
- Tamaño del dato a leer o a escribir dentro del sistema de almacenamiento.

Es posible la integración de la arquitectura en entornos *cluster* mediante la incorporación de la arquitectura de sistemas intermedios de almacenamiento en distintos niveles de E/S. La Figura 5.16 representa un ejemplo de integración de la arquitectura de sistemas intermedios de almacenamiento a través de la interfaz estándar MPI-IO.

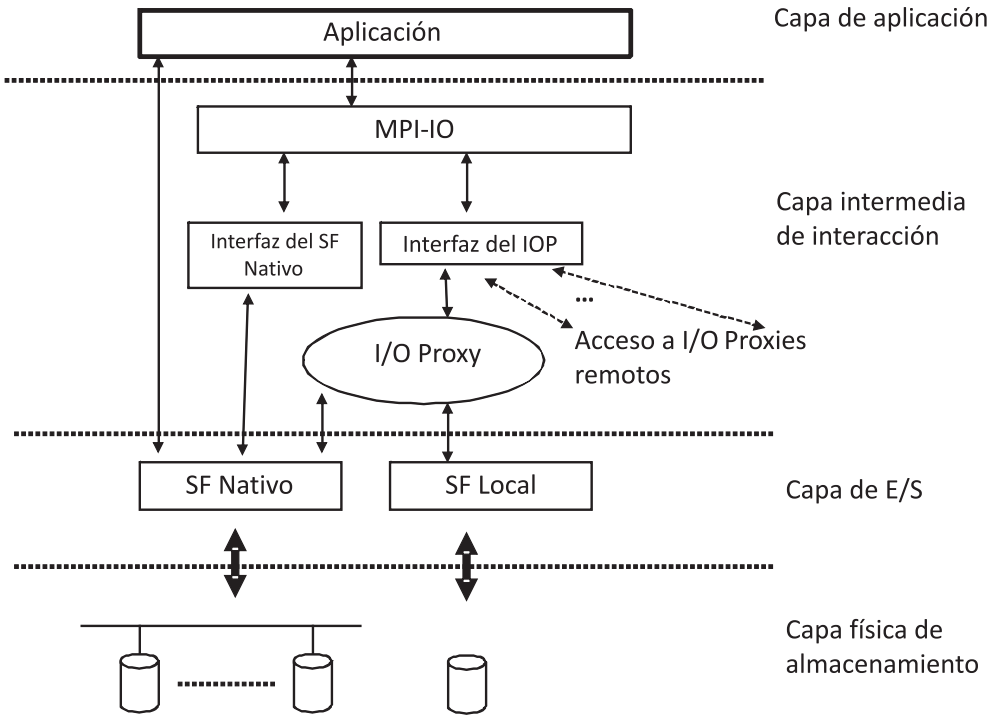

Figura 5.16: Integración de la arquitectura de *IOP* en una aplicación.

5.9. Resumen

En este capítulo se han definido los principales problemas de las arquitecturas de E/S en grandes *clusters*:

- Escaso número de nodos de E/S comparado con el gran número de nodos de cómputo disponibles en estos sistemas [6], lo que repercute en el rendimiento de las aplicaciones y en la escalabilidad del sistema.

- Poca flexibilidad de la arquitectura de E/S de los *cluster*, para adaptarse a las necesidades de las aplicaciones.

Para solucionar estos problemas, se propone una nueva arquitectura de E/S para grandes *clusters*, basada en el uso de servidores intermedios de almacenamiento, denominados *I/O proxies* (*IOP*). Estos se agrupan proporcionando un nivel de E/S intermedio entre las aplicaciones y el sistema de almacenamiento, gestionando los datos manejados entre ambos.

Estos *IOP* utilizan los dispositivos de almacenamiento disponibles en los nodos de cómputo, lo que evita la adquisición de hardware adicional. Al disponer de tantos *IOP* como nodos de cómputo dispuestos en un *cluster*, se puede incrementar virtualmente el número de nodos de E/S en el *cluster* aumentando el grado de paralelismo. Por otra parte, al ser una arquitectura software, es posible incorporarla fácilmente entre los niveles de E/S disponibles en un *cluster*.

Los *IOP* se agrupan formando espacios virtuales de almacenamiento (*EVA*), donde se almacenan los datos manejados por las aplicaciones. Los datos se almacenan en los *EVA* usando distintas políticas de distribución, y un sistema de replicación para proporcionar tolerancia a fallos. Estas agrupaciones pueden tener distintas configuraciones (públicas, privadas, etc.), para ajustarse de una forma más adecuada a las necesidades de las aplicaciones. Para el intercambio de datos entre los *EVA* y el sistema de almacenamiento, se han establecido políticas de sincronización y de lectura de datos.

Por último, se ha definido una implementación de la arquitectura de E/S así como la integración de la misma en un entorno *cluster*.

Capítulo 6

Evaluación

En este capítulo se presentan los principales resultados de la evaluación de los temas propuestos en la tesis, el sistema de ficheros paralelo Expand y la arquitectura de servidores intermedios de almacenamiento. En primer lugar se muestran los resultados obtenidos de Expand junto a distintos sistemas de ficheros paralelos en un entorno *cluster*. A continuación, las evaluaciones de la arquitectura de servidores intermedios para *clusters* y finalmente, se detalla de forma resumida los resultados obtenidos tanto de Expand como de la arquitectura de servidores intermedios.

6.1. Evaluación del sistema de ficheros paralelo Expand

En esta sección se detalla el estudio realizado del sistema de ficheros paralelo Expand en un entorno *cluster* y se especifican los distintos escenarios y *benchmarks* utilizados.

6.1.1. Definición de las pruebas

Para analizar el comportamiento del sistema de ficheros paralelo Expand en un entorno de tipo *cluster* se han realizado evaluaciones en diferentes escenarios utilizando *benchmarks* de E/S estándar.

- Acceso paralelo a un fichero mediante el *benchmark* IOR.
- *Benchmark effective File-I/O bandwidth* (b_eff_io).
- *Benchmark* FLASH I/O.
- Operaciones sobre metadatos.

- Aplicación de procesamiento de imágenes.

Las tres primeras pruebas se han realizado empleando la interfaz MPI-IO y pretenden probar el rendimiento de Expand en entornos de *cluster*. Para las dos últimas se ha utilizado la interfaz POSIX. Las operaciones sobre metadatos intentan medir el comportamiento del sistema frente a operaciones sobre metadatos de ficheros. La última persigue evaluar el rendimiento del sistema en un entorno de alta productividad. También se ha estudiado el efecto de añadir nuevos nodos a particiones existentes.

A continuación se describen la plataforma utilizando en las pruebas y cada una de las pruebas junto con sus resultados.

6.1.2. Plataforma de pruebas

La plataforma utilizada en la evaluación de las pruebas descritas anteriormente ha sido un *cluster* de 8 biprocesadores Pentium IV a 3.2 GHz con 2 GB de memoria RAM conectados a través de una Gigabit Ethernet, ejecutando el sistema operativo Linux (kernel 2.6.9) y la distribución para Clusters Rocks. En todos los experimentos se ha comparado Expand con los sistemas de ficheros paralelos PVFS (versión 2) [11] y GPFS [52]. PVFS2 está desarrollado por investigadores de los laboratorios Argonne, la Universidad de Clemson y el centro de supercomputación de Ohio siendo un proyecto de código abierto. GPFS está desarrollado y mantenido por IBM.

El tamaño del bloque utilizado en PVFS y GPFS ha sido el tamaño de bloque por defecto: 64 KB en PVFS y 256 KB en GPFS. Para Expand se ha utilizado la versión 2 de NFS con protocolo TCP y un tamaño de bloque de 8 KB. Sobre este cluster se ha configurado una partición de 8 servidores para cada uno de los sistemas de ficheros paralelos a evaluar (Expand, PVFS y GPFS), es decir, todas las máquinas del *cluster* se han utilizado como clientes y servidores. Para las pruebas que utilizan MPI-IO se ha empleado MPICH2 [117]. Ni Expand ni PVFS disponen de caché en los nodos clientes, lo que evita posibles problemas de coherencia. GPFS sí incluye una caché en los clientes del sistema de ficheros paralelo junto con un mecanismo de coherencia basado en cerrojos que aseguran la coherencia en el acceso a los datos. Todas las pruebas se han realizado 10 veces y en los resultados se muestran los tiempos medios.

6.1.3. Acceso paralelo a un fichero

La prueba *acceso paralelo a un fichero* se ha realizado utilizando el *benchmark* IOR [118] desarrollado en el *Lawrence Livermore National Laboratory* de EE UU. Este test emula el acceso paralelo a un fichero por parte de una aplicación paralela. Los procesos (véase la Figura 6.1) de la aplicación acceden al fichero de forma entrelazada y no contigua, utilizando diferentes tamaños de acceso. Para la prueba se ha empleado un fichero de 500 MB y diferentes tamaños de acceso (desde 256 bytes hasta 1 MB).

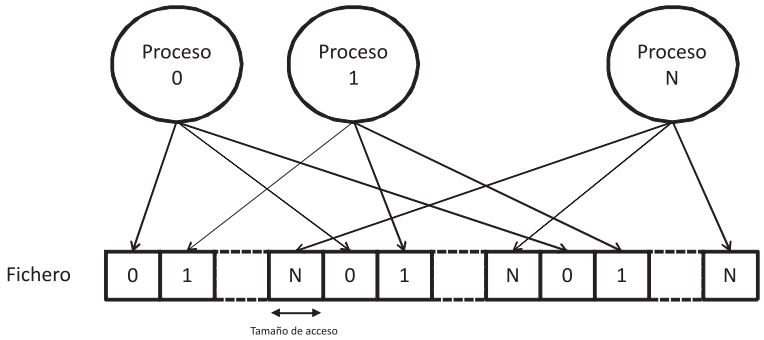

Figura 6.1: Patrón de acceso paralelo a un fichero

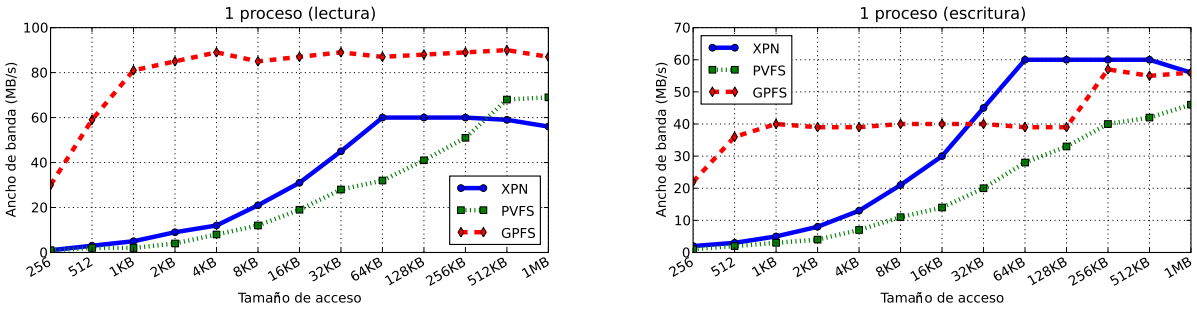

Figura 6.2: Resultados para la aplicación con 1 proceso (lectura y escritura)

Los resultados para la prueba de acceso paralelo a un fichero se muestran en las Figuras 6.2 a 6.6. Estas reflejan el ancho de banda agregado obtenido en las operaciones de lectura y escritura para 1, 2, 4, 8 y 16 procesos y diferentes tamaños de acceso (desde 256 bytes hasta 1MB).

Como se puede comprobar en las Figuras 6.2 a 6.6, en el caso de escritura concurrente los mejores resultados se obtienen con Expand y los peores con GPFS debido al coste necesario para asegurar la coherencia. Sólo en el caso de utilizar un único proceso, lo que implica que no hay acceso paralelo, GPFS ofrece mejores resultados debido a la caché de bloques utilizada en el cliente. En cuanto a los resultados de lectura, GPFS ofrece mejores resultados para tamaños de acceso pequeños debido a la caché de bloques del cliente. A medida que el tamaño de acceso aumenta el efecto de la caché es menor.

Las Figuras 6.7 y 6.8 comparan los resultados obtenidos en esta prueba para diferentes números de procesos accediendo con tamaños de acceso de 8 KB y 256 KB. Los datos señalados en estas figuras muestran que la mejor escalabilidad se consigue con Expand y PVFS. Los tiempos de escritura en GPFS para tamaños de acceso de 256 KB mejoran debido a que este tamaño de acceso coincide con el tamaño de bloque empleado por este sistema de ficheros.

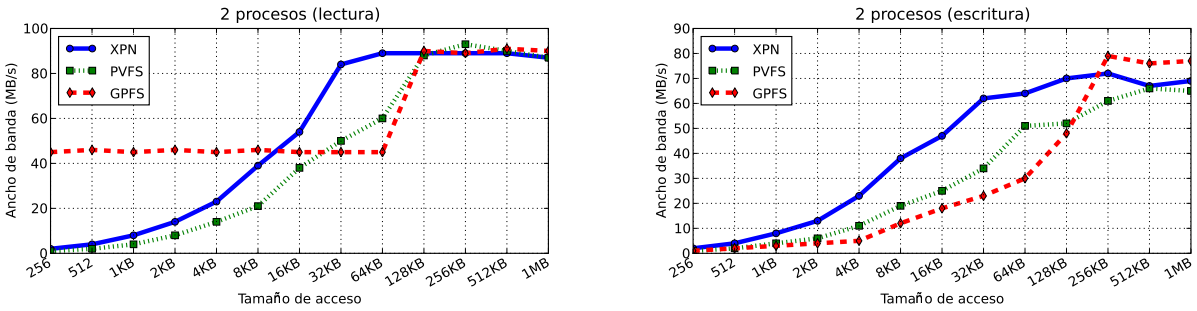

Figura 6.3: Resultados para la aplicación paralela con 2 procesos (lectura y escritura)

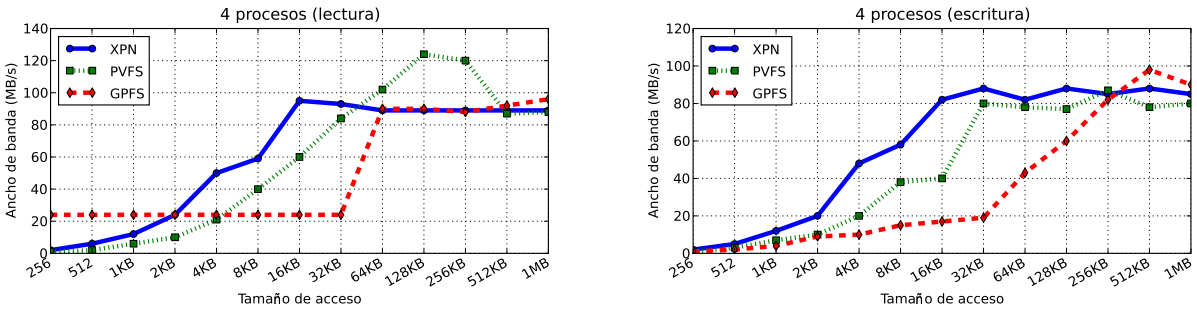

Figura 6.4: Resultados para la aplicación paralela con 4 procesos (lectura y escritura)

6.1.4. *Benchmark effective File-I/O bandwidth*

El test *effective File-I/O bandwidth benchmark* (*b_eff_io*) [119] tiene dos objetivos: obtener un número medio característico del ancho de banda que se puede conseguir con una aplicación paralela que utiliza la interfaz MPI-IO, y los detalles sobre diferentes patrones y tamaños de acceso. El *benchmark* examina los siguientes aspectos:

- Un conjunto de particiones.
- Diferentes métodos de acceso, escritura inicial (*initial write*), reescritura (*rewrite*) y lectura (*read*).
- Diferentes patrones de acceso (véase la Figura 6.9):(1) Acceso colectivo entrelazado, distribuyendo grandes porciones de memoria a disco y viceversa; (2)Acceso colectivo entrelazado, pero con una lectura o escritura por bloque; (3)Acceso no colectivo a un fichero por diferentes procesos MPI, es decir, acceso a ficheros independientes; (4) Igual que en el caso 2, pero los ficheros individuales se combinan en un fichero segmentado; (5) Igual que el caso 3, pero al fichero segmentado se accede con operaciones colectivas.
- El tamaño del bloque contiguo se elige de dos formas, como potencia de dos (*wellformed*) y de forma irregular sumando 8 bytes a los tamaños anteriores.

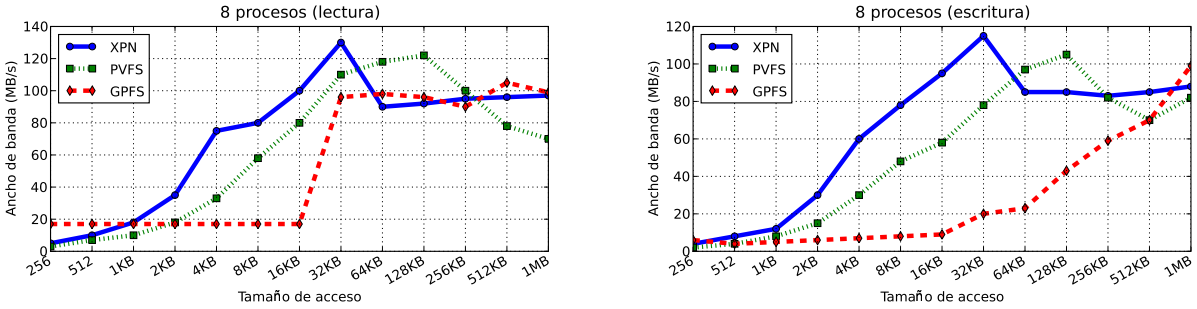

Figura 6.5: Resultados para la aplicación paralela con 8 procesos (lectura y escritura)

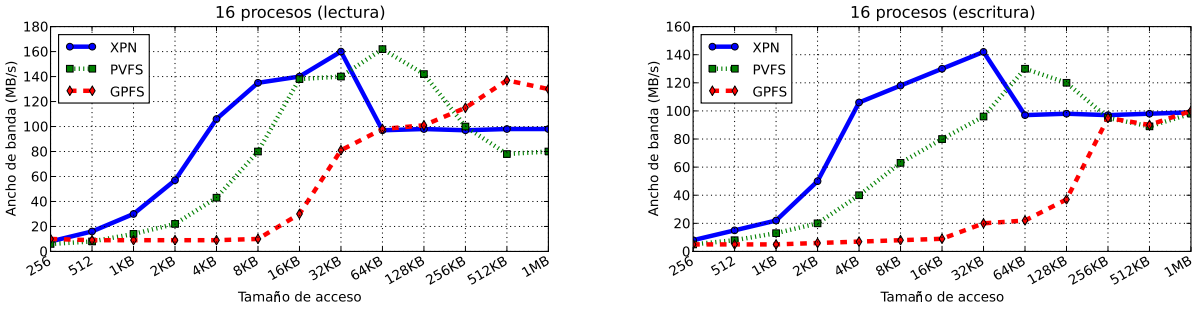

Figura 6.6: Resultados para la aplicación paralela con 16 procesos (lectura y escritura)

Se utilizan diferentes tamaños de acceso, fundamentalmente de 1 KB, 32 KB, 1 MB y 8 MB.

Las figuras 6.10, 6.11 y 6.12 presentan los resultados obtenidos para el *benchmark* *b_eff_io* en Expand, PVFS y GPFS.

La prueba se ha ejecutado durante 30 minutos, tal y como recomienda el *benchmark*. Estas Gráficas muestran el ancho de banda obtenido para tres métodos de acceso diferentes: escritura por primera vez en el fichero, reescritura del mismo fichero y lectura. El ancho de banda se muestra en escala logarítmica, para cada patrón y tamaño de acceso. Las pruebas se han realizado utilizando punteros a fichero individuales. El rendimiento obtenido es muy similar en los sistemas de ficheros evaluados. Los resultados globales del *benchmark* se pueden ver también en la Figura 6.13. Los mejores resultados se obtienen en lectura para GPFS debido al efecto de la caché del cliente y en escritura para Expand.

6.1.5. *Benchmark* FLASH-IO

El *benchmark* FLASH-IO [105] utiliza la interfaz paralela HDF5 que a su vez utiliza la interfaz MPI-IO. FLASH es una aplicación paralela de la Universidad de Chicago desarrollada para estudiar destellos termonucleares, tales como supernovas y ráfagas de rayos X, tanto en dos como en tres dimensiones. El *benchmark* FLASH-

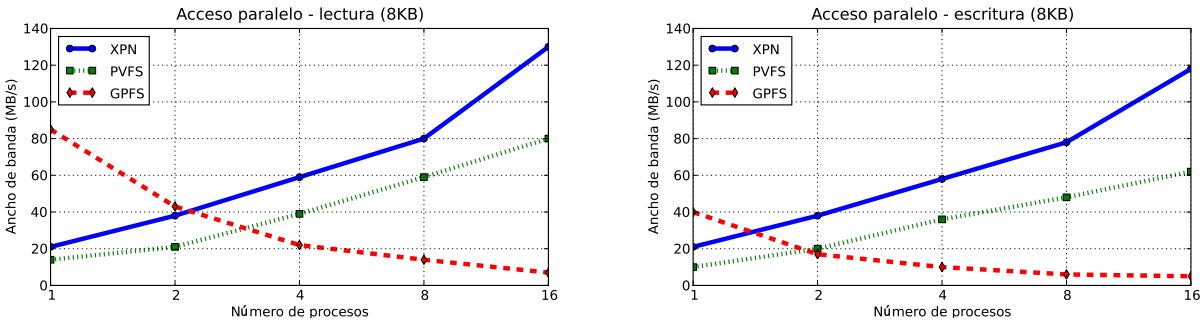

Figura 6.7: Resultados para la aplicación paralela para diferentes procesos y 8 KB de tamaño de acceso (lectura y escritura)

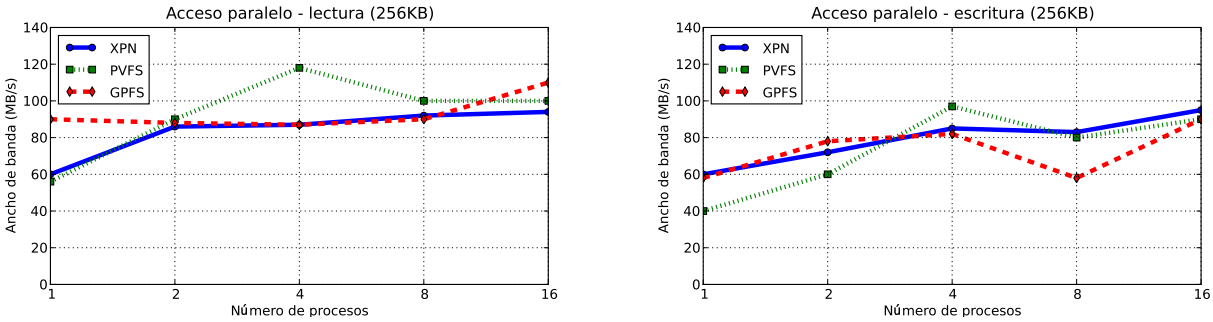

Figura 6.8: Resultados para la aplicación paralela para diferentes procesos y 256 KB de tamaño de acceso (lectura y escritura)

IO prueba la parte de E/S de la aplicación FLASH. Para ello define las mismas estructuras que el programa FLASH, las rellena con datos ficticios y realiza las operaciones de E/S a través de la interfaz HDF5. Este *benchmark* realiza tres pruebas por separado para tres clases de ficheros:

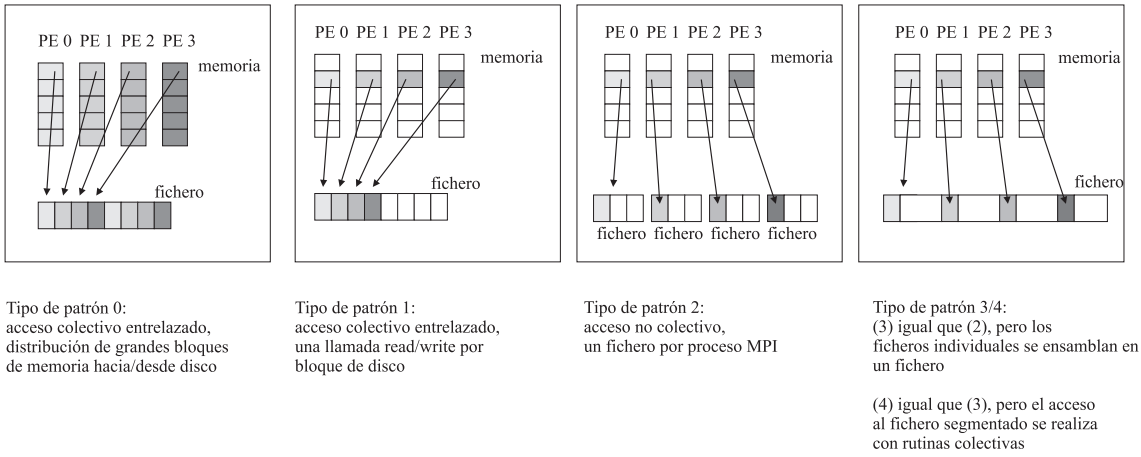

Figura 6.9: Patrones de acceso utilizados en el *benchmark b_ eff_ io*. Cada diagrama muestra los datos transferidos por un llamada MPI-IO de escritura

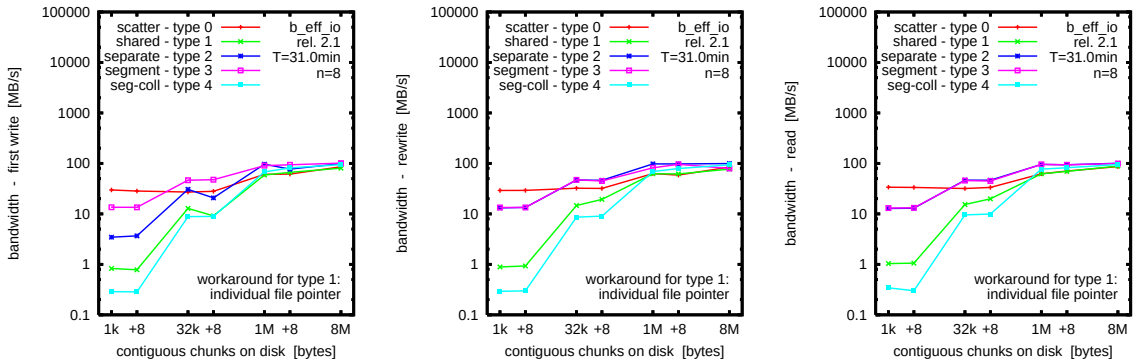

Figura 6.10: Resultados para el *benchmark b_ eff_ io*. 8 nodos en Expand

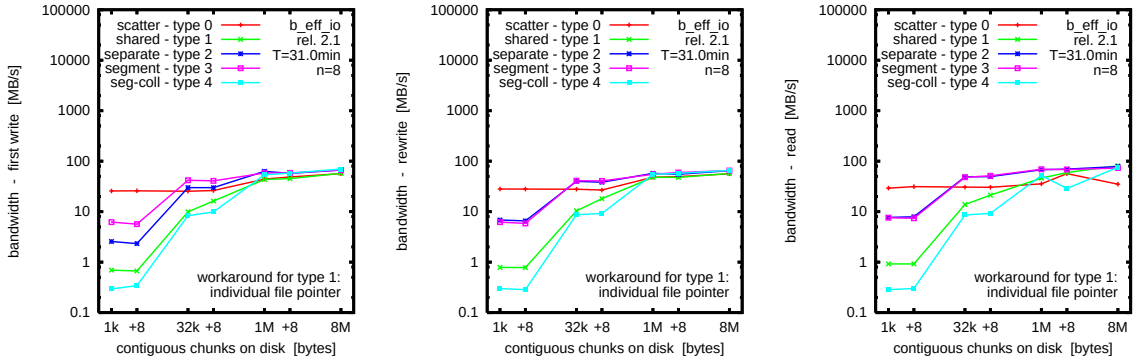

Figura 6.11: Resultados para el *benchmark b_ eff_ io*. 8 nodos en PVFS

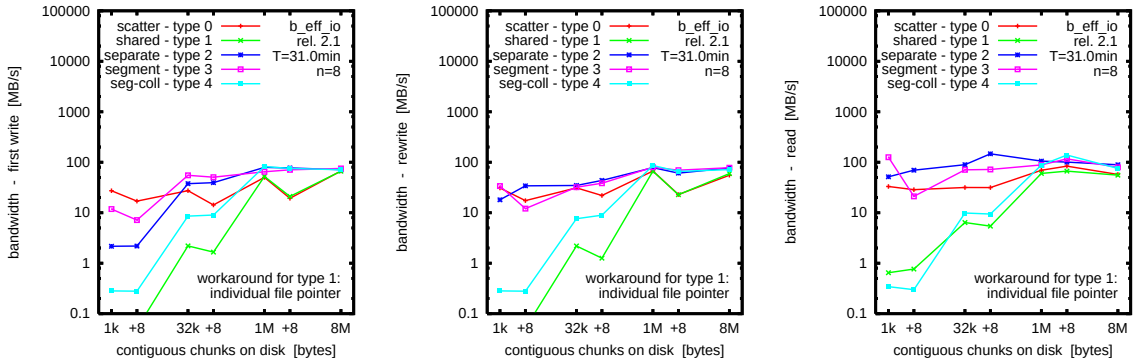

Figura 6.12: Resultados para el *benchmark b_ eff_ io*. 8 nodos en GPFS

- Ficheros de *checkpoint* necesarios para reiniciar las pruebas después de una ejecución fallida. Estos almacenan todas las variables, la estructura en árbol, el paso de simulación en el momento y el número de pasos totales de la simulación.

- Ficheros usados para la visualización de la ejecución.

- Ficheros que se utilizan de forma similar a los anteriores pero con un paso extra para generar bloques interpolados de 9 por 9 en lugar de 8 por 8, de forma que se facilite la visualización.

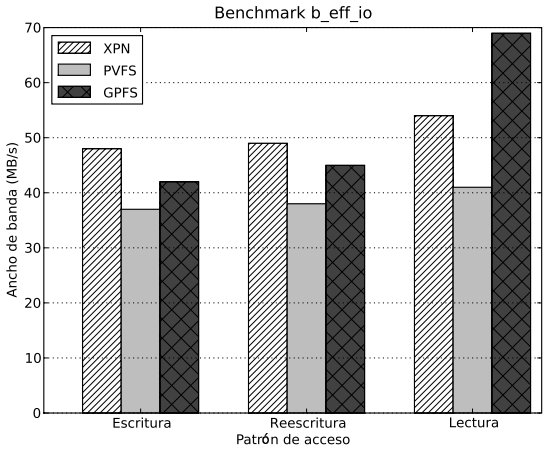

Figura 6.13: Resultados globales del *benchmark b_ eff_ io*

Las estructuras de datos en memoria son bloques de 3 dimensiones de tamaño $8 \times 8 \times 8$. En la simulación cada proceso de la aplicación almacena 640 de estos bloques. Cada uno de estos elementos de datos tiene 24 variables asociadas. Dentro de cada fichero, los datos para las mismas variables deben almacenarse de forma contigua. Este *benchmark* es una aplicación intensiva en escrituras con un patrón de acceso no contiguo, tanto en memoria como en el fichero, y con tamaños de acceso muy dispares, lo que representa uno de los peores escenarios en el acceso a un fichero por parte de una aplicación paralela. La cantidad de datos que maneja el *benchmark* es proporcional al número de procesos de la aplicación paralela y varía desde 73,53 MB en el caso de 1 proceso hasta 1.16 GB en el caso de 16 procesos.

La Figura 6.14 muestra el ancho de banda, que se obtiene en la ejecución del *benchmark* FLASH-IO. Cuando se emplea un único proceso el mejor resultado se obtiene para GPFS debido al efecto de la caché del cliente y a que el protocolo de coherencia no tiene efecto. A medida que aumenta el número de procesos de la aplicación el rendimiento para GPFS se degrada como consecuencia del protocolo de coherencia. Este efecto no aparece en Expand y en PVFS cuyo comportamiento escala con el número de procesos de la aplicación.

6.1.6. Operaciones sobre metadatos

El objetivo de este test es analizar el comportamiento de los diferentes sistemas de ficheros comparados en este trabajo en operaciones realizadas sobre metadatos. Para ello se realizan distintas pruebas para su evaluación:

- Creación de 1000 ficheros vacíos.
- Creación de 1000 ficheros pequeños (100 bytes).
- Creación de 1000 ficheros grandes (200 MB).

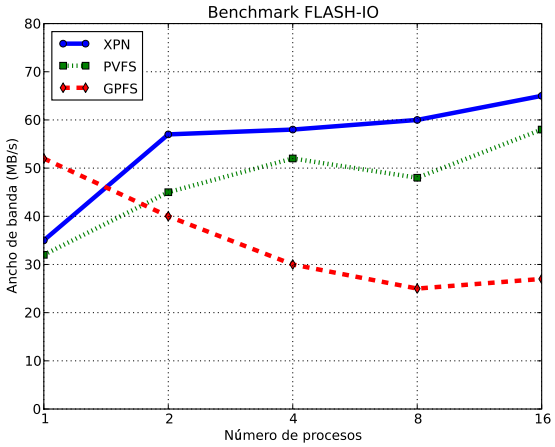

Figura 6.14: Resultados del *benchmark* FLASH-IO

- Renombrado de 1000 ficheros vacíos.
- Borrado de 1000 ficheros vacíos.

En el caso de PVFS, se han evaluado dos configuraciones distintas: una con un servidor de metadatos y otra usando 8 (denominadas en las Gráficas como *1 sm* y *8 sm* respectivamente).

Las pruebas se desarrollan en dos escenarios, en el primero un proceso efectua la operación y en el segundo 8 procesos realizan las mismas operaciones en paralelo, cada uno sobre un directorio diferente. Las Figuras 6.15 y 6.16 muestran el número de ficheros por segundo que se pueden crear en cada uno de los sistemas de ficheros paralelos evaluados para ficheros vacíos y pequeños. Los peores resultados se obtienen para PVFS usando un único servidor de metadatos, debido a que concentra todos los accesos de metadatos en un único servidor creando un cuello de botella y también al coste de almacenar los metadatos en una base de datos. Expand ofrece un buen comportamiento habida cuenta de que un fichero en Expand consta de varios subficheros de datos y uno de metadatos distribuido entre todos los servidores de datos.

La Figura 6.17 muestra el ancho de banda que se obtiene en la creación de ficheros grandes. Para un único proceso los mejores resultados se obtienen para GPFS debido, como en pruebas anteriores, al efecto de la caché del cliente. Para ficheros grandes y 8 procesos los resultados que se obtienen en Expand, PVFS (con 8 servidores de metadatos) y en GPFS son muy similares.

Por último, se ha evaluado el renombrado (Figura 6.18) y borrado de ficheros (Figura 6.19). Se puede apreciar que el rendimiento de Expand es similar al obtenido con PVFS usando 8 servidores de metadatos. Como casos extremos, encontramos el uso de PVFS con un único servidor de metadatos como peor sistema de ficheros y GPFS como el mejor. Todos los resultados demuestran el buen comportamiento de Expand en la gestión de operaciones sobre metadatos.

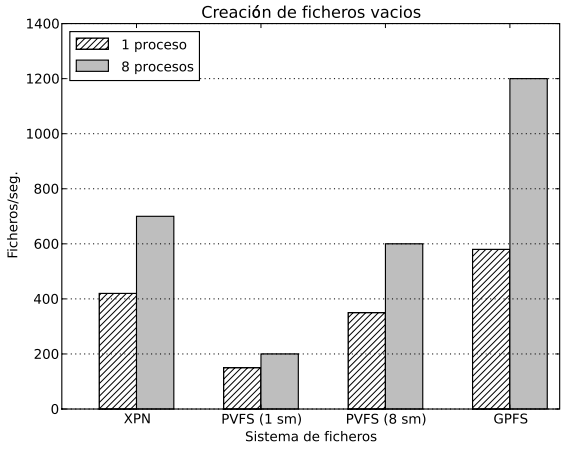

Figura 6.15: Resultados de la creación de ficheros (ficheros vacíos)

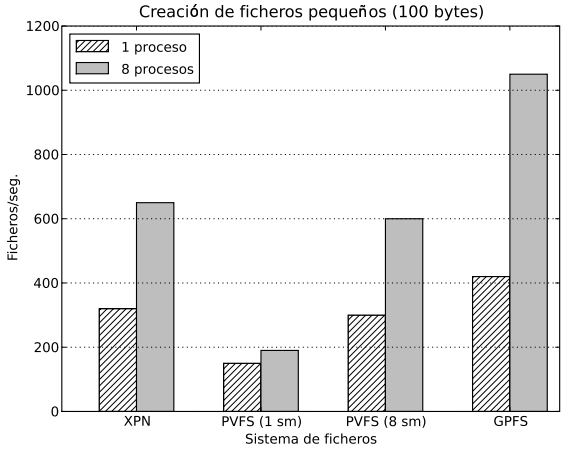

Figura 6.16: Resultados de la creación de ficheros (ficheros pequeños)

6.1.7. Aplicación de procesamiento de imágenes

En la aplicación de procesamiento de imágenes se dispone de un programa que procesa un conjunto de 256 imágenes, todas ellas de 5 MB de tamaño. Sobre cada imagen, la aplicación devuelve una nueva imagen que se obtiene de aplicar una máscara fija de 64 pixels. El volumen de datos manejado en esta aplicación es de 2,5 GB. En esta prueba cada proceso de la aplicación procesa de forma independiente un subconjunto de todas las imágenes y no hay acceso paralelo a un fichero.

Para esta prueba se han evaluado dos escenarios diferentes:

- Versión del programa escrito en Java utilizando la implementación de Expand en Java, una partición de PVFS y una partición de GPFS.
- Versión escrita en C utilizando Expand, una partición de PVFS y una partición de GPFS.

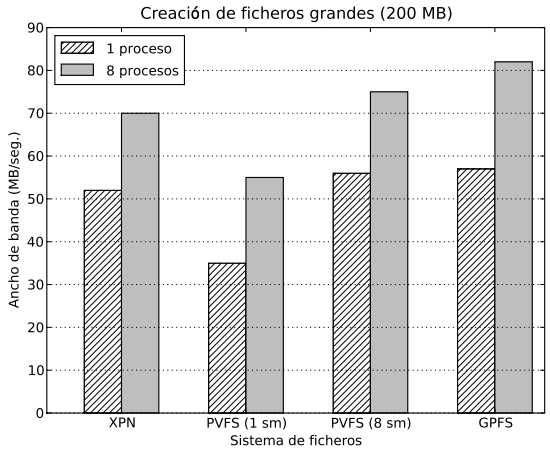

Figura 6.17: Resultados de la creación de ficheros (ficheros grandes)

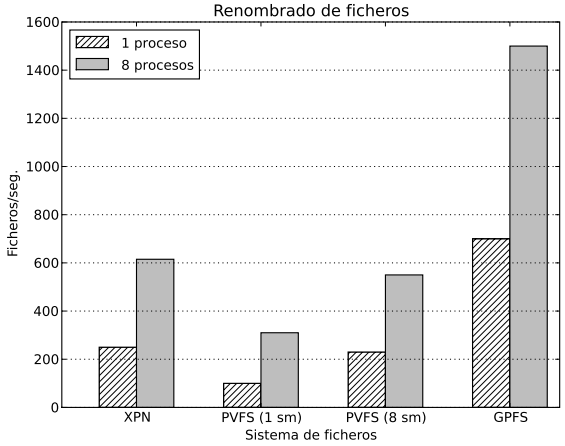

Figura 6.18: Resultados del renombrado de ficheros

Los resultados para esta aplicación se muestran en la Figura 6.20. En el caso de la versión programada en C, el comportamiento de los tres sistemas de ficheros es muy similar. Para 8 procesos el rendimiento que se obtiene es peor que para 4 debido a que para 8 el sistema de ficheros paralelo se satura. La mejoría en el rendimiento que se aprecia para 16 clientes se debe a que los computadores utilizados son biprocesadores. En cuanto a la versión programada en Java, el mejor resultado se obtiene para PVFS y GPFS debido a que estos sistemas de ficheros no se encuentran programados en Java. Para estos dos sistemas de ficheros no se obtiene una mejora en el rendimiento cuando se incrementa el número de procesos de la aplicación debido a que el cuello de botella se encuentra en el propio sistema de ficheros. En el caso de la versión de Expand programada en Java, se pueden observar que se reduce el tiempo a medida que se aumenta el número de procesos. Esto se debe a que para pocos procesos el cuello de botella se encuentra en la máquina virtual de Java y a medida que aumenta el número de procesos el cuello de botella deja de ser la

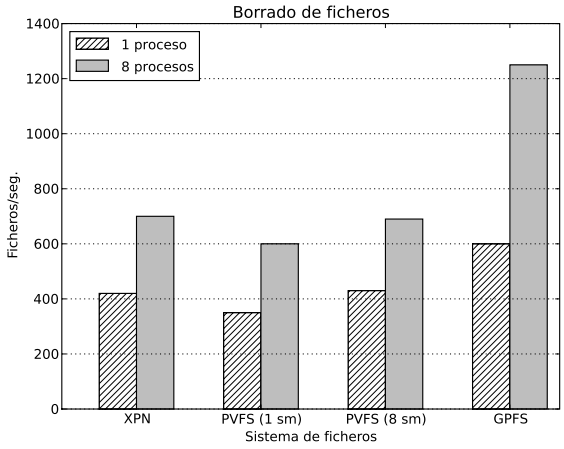

Figura 6.19: Resultados del borrado de ficheros

máquina virtual para pasar a ser el sistema de ficheros. También se pueden observar los buenos resultados que se obtienen para 8 y 16 procesos. En estos dos casos el cuello de botella deja de ser la máquina virtual para pasar a serlo como en el resto de sistemas de ficheros evaluados, los servidores de E/S.

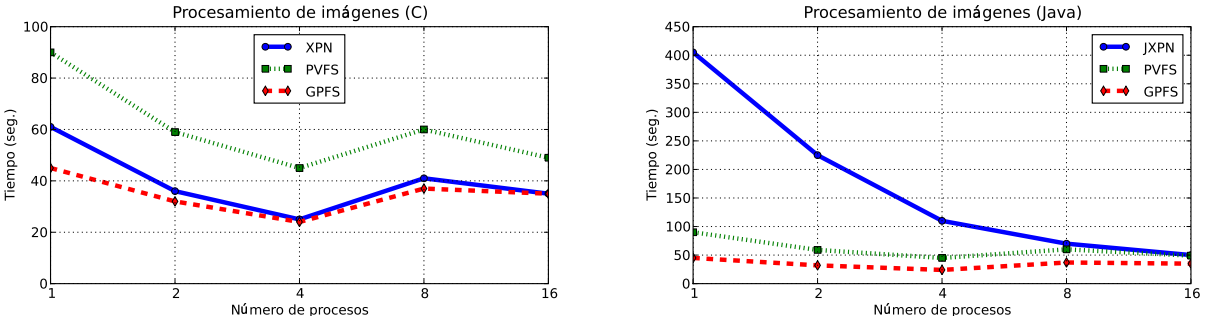

Figura 6.20: Resultados de la aplicación de procesamiento de imágenes.

Como conclusión de todas estas pruebas se puede decir que los resultados que se obtienen utilizando Expand son muy satisfactorios y demuestran que es factible utilizar servidores de datos estándar para construir un sistema de ficheros paralelo.

6.1.8. Reconfiguración dinámica de particiones

En esta sección se evalúa el impacto de la incorporación de nuevos nodos a una partición. Para ello se han utilizado los ficheros de un sistema de ficheros de un servidor real que se han copiado a diferentes particiones de Expand. Este sistema de ficheros está formado por 256.400 ficheros y ocupa un tamaño de 30 GB. La parte izquierda de la Figura 6.21 muestra el ancho de banda que se obtiene en la reconstrucción de los metadatos del sistema de ficheros y la parte derecha el ancho de banda que se obtiene en la reconstrucción total de todos los ficheros de la

partición. Las barras incluyen, en su parte superior, el tiempo en minutos necesario para realizar la reconstrucción de la partición. El eje de abscisas de las dos figuras muestra diferentes modelos de reconstrucción, donde I-J representa el efecto de pasar de una partición compuesta por I nodos a una partición compuesta por J nodos. Como se puede observar la reconstrucción de los metadatos permitiría incorporar nuevos nodos de datos con una operación que se podría efectuar en unos 8 minutos.

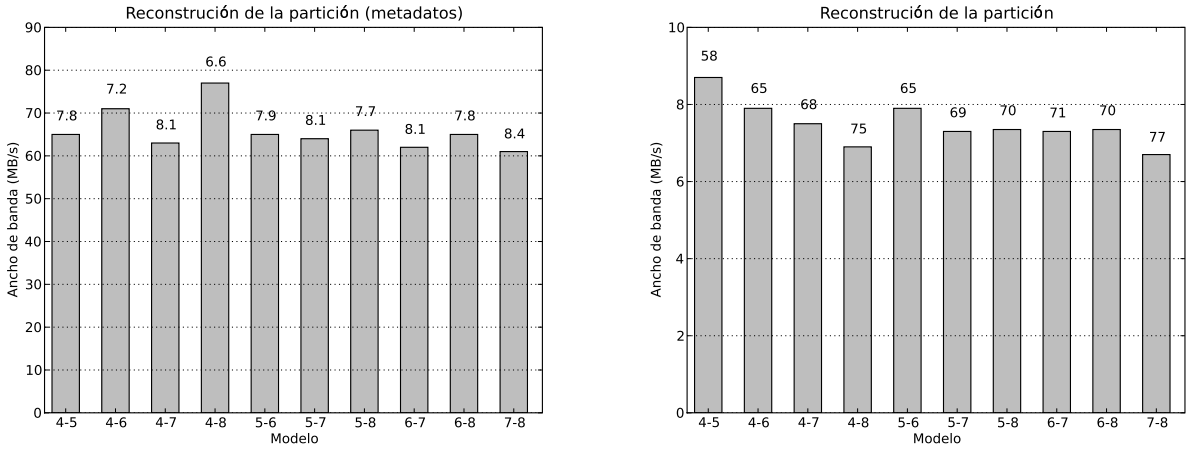

Figura 6.21: Resultados de la incorporación de nuevos nodos a una partición (reconstrucción sólo de los metadatos, reconstrucción de toda la partición)

6.2. Evaluación de la arquitectura de nodos de almacenamiento intermedios

En esta sección se mostrará la evaluación de la arquitectura basada en sistemas de almacenamiento intermedio para entornos *cluster*. En ella se detallan las pruebas y test, así como los entornos utilizados.

6.2.1. Definición de las pruebas

Para analizar el comportamiento del sistema de almacenamiento intermedio en un *cluster* se han tenido en cuenta diferentes escenarios utilizando *benchmarks* de E/S estándar. En todos los casos se compara la arquitectura basada en sistemas intermedios de almacenamiento con tradicional de E/S para este tipo de entornos. Se han estudiado varios aspectos:

- Acceso paralelo a un fichero mediante el *benchmark* IOR usando un prototipo de la arquitectura.
- Evaluación del prototipo en un entorno de alta productividad .

- Emulación de un entorno de alta productividad en grandes entornos *cluster*.

Las dos primeras pruebas estudian el comportamiento de la arquitectura propuesta con respecto a las arquitecturas de almacenamiento tradicionales en distintos entornos:

- Computación paralela (HPC): mediante una aplicación paralela efectuando operaciones de E/S (lectura y escritura) sobre un mismo fichero.
- Computación de alta productividad (HTC): mediante la ejecución de una carga de trabajo que emula operaciones de minería de datos.

Para la última prueba se han evaluado arquitecturas de E/S de grandes *cluster* mediante el diseño de un modelo analítico de este tipo de entornos. Este modelo permite definir y comparar las aquitecturas de E/S tradicionales y la arquitectura de E/S propuesta.

6.2.2. Plataforma de pruebas

Para los dos primeros test se ha utilizado la misma la plataforma que la descrita en la sección 6.1.2. Para todas las pruebas se va a disponer de un servidor NFS (versión 3) para la transferencia de datos, y de 8 nodos de cómputo como servidores intermedios de datos, empleando Expand como sistema de fichero paralelo intermedio. Además en todas las pruebas se han manejado dos redes de datos GigaEthernet: una conexión para la transmisión de los datos entre los nodos de cómputo y el servidor NFS, y otra para el intercámbio de información entre las aplicaciones y los IOP.

Todas las pruebas se ejecutarán con el benchmark IOR, detallado anteriormente en la sección 6.1.3. En todos los casos se utilizará MPI-IO como interfaz estándar para el acceso a los datos. Las cargas de trabajo se lanzarán en el sistema con el sistema de colas Torque [120].

El modelo analítico, se ha implementado mediante la plataforma de eventos discretos **OMNET++** [121]. Esta plataforma permite simular/emular múltiples arquitecturas hardware y software, según referencian numerosos artículos científicos y proyectos de evaluación de sistemas.

6.2.3. Acceso paralelo a un fichero

En esta ocasión se quiere conocer el rendimiento de una aplicación paralela usando los dos sistemas de almacenamiento disponibles: un sistema centralizado con NFS como servidor estándar de datos y otro con sistemas de almacenamiento intermedio. Así analizaremos el rendimiento percibido por el usuario utilizando únicamente los sistemas intermedios de almacenamiento, y el coste temporal de efectuar las operaciones de *escritura adelantada* o de *escritura al cierre* sobre el sistema centralizado de datos.

En la prueba, un programa paralelo compuesto por 4 procesos, generará múltiples ficheros de 100 MB cada uno. Posteriormente se leerán usando diversos tamaños de acceso a los datos (8KB, 64KB y 256KB). El número de los servidores intermedios de datos en las pruebas es de 2 y 4 respectivamente y emplean distintos tamaños de reparto (64KB y 256KB). Se han considerado dos configuraciones:

- Primero se ha evaluado una arquitectura donde los procesos de la aplicación y los nodos intermedios de almacenamiento se encuentran localizados en el mismo conjunto de nodos de cómputo.
- Por último, se ha determinado situar las aplicaciones y los nodos de almacenamiento intermedio en distintos conjuntos de nodos de cómputo.

En todos los casos los datos son eliminados de la caché de los nodos clientes, de los servidores intermedios y del servidor NFS antes de comenzar un test.

6.2.3.1. Resultados

En los resultados de las pruebas se muestra el *speedup* obtenido entre los nodos intermedios y la arquitectura tradicional. Este *speedup* se ha calculado mediante la siguiente fórmula:

$$Speedup = \frac{Time_S(W)}{Time_{IOP}(W)} \tag{6.1}$$

Donde:

- $Time_S(W)$: Tiempo de la aplicación usando el sistema de almacenamiento del cluster.
- $Time_{IOP}(W)$: Tiempo de la aplicación utilizando los servidores intermedios de almacenamiento.

Se han estudiado distintas situaciones usando los servidores intermedios, dependiendo del tipo de operación realizada:

- Lectura:
  - Carga de los datos bajo demanda.
  - Lectura adelantada de los datos.
- Escritura:
  - Escritura en los servidores intermedios.
  - Uso de escritura al cierre de los ficheros.

Las Gráficas 6.22 y 6.23 indican el *speedup* conseguido por parte de una aplicación utilizando 2 y 4 nodos de almacenamiento intermedio (IOP) en el eje de ordenadas y las distintas configuraciones utilizadas, en el de abscisas. Así, los primeros valores representan el *speedup* conseguido usando únicamente los IOP para realizar las operaciones de E/S, mientras que los segundos valores representan el *speedup* si tiene lugar la lectura adelantada de los datos como es el caso de la Figura 6.22. Por otro lado, los primeros valores de la Figura 6.23 representa el *speedup* con la creación de un fichero nuevo, mientras que los siguientes valores representan el *speedup* obtenido al sobreescribir los datos de un fichero ya existente en el servidor NFS.

Las Gráficas 6.24 y 6.25 muestran las situaciones en las que los IOP y las aplicaciones se encuentran en distintos nodos. Se puede apreciar que los datos obtenidos mejoran ostensiblemente el sistema de almacenamiento, como en el caso de los accesos de pequeño tamaño (8KB), especialmente cuando la aplicación se encuentra en nodos de cómputo distintos. Usar únicamente los proxies incrementa el rendimiento percibido por las aplicaciones de forma sustancial.

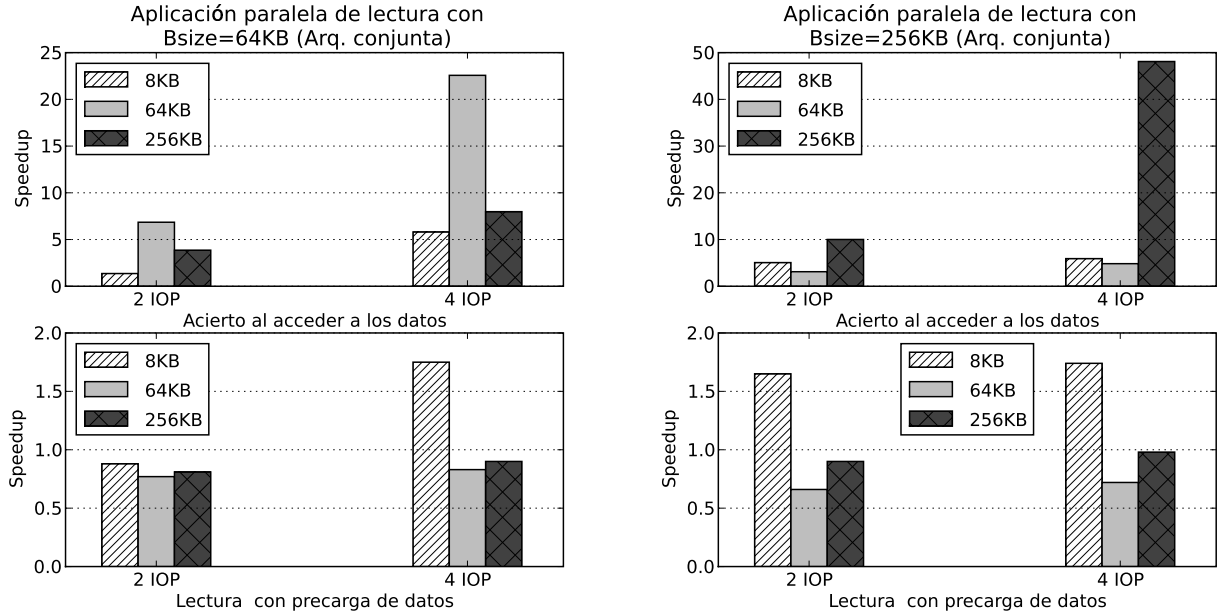

Figura 6.22: Resultados de la aplicación paralela en modo lectura usando los IOP en los mismos nodos que las aplicaciones

Por otra parte, los mejores resultados se obtienen cuando el tamaño de acceso de la aplicación y el tamaño de reparto usado por parte de los IOP son similares. Además, emplear los IOP en los mismos nodos de cómputo donde se encuentra la aplicación es beneficioso para obtener mejores rendimientos del sistema. Esto mejora sobretodo el sistema en los casos en que los *I/O proxies* se encuentran en los mismos nodos que las aplicaciones, debido a que el número de accesos a datos locales es muy alto.

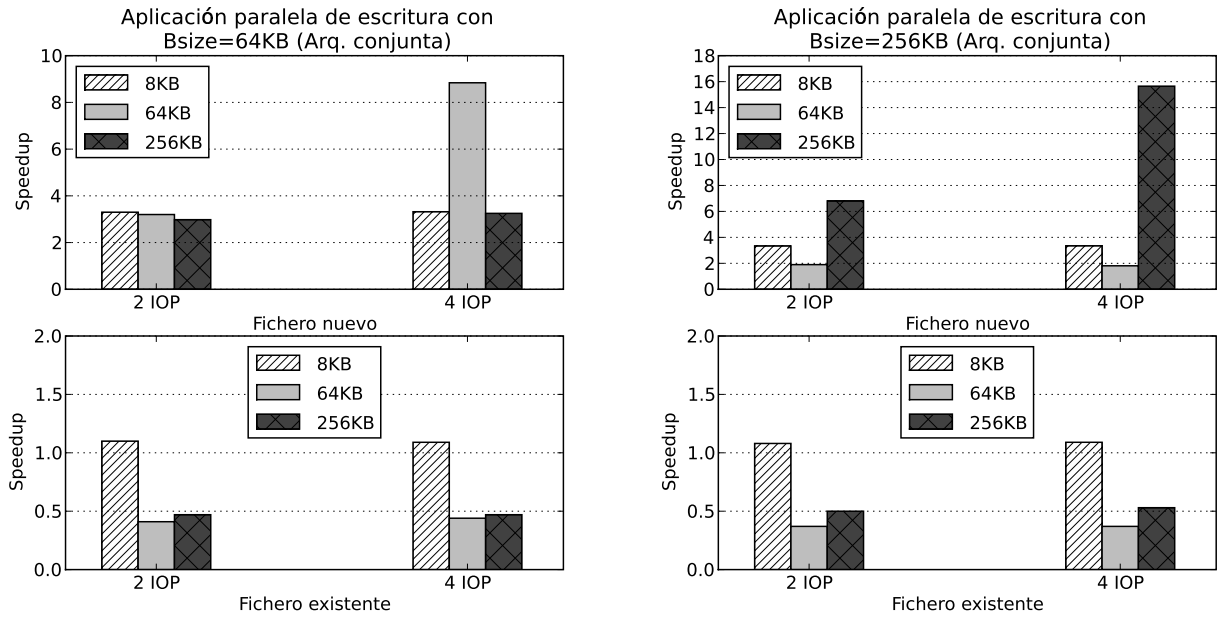

Figura 6.23: Resultados de la aplicación paralela en modo escritura usando los IOP en los mismos nodos que las aplicaciones

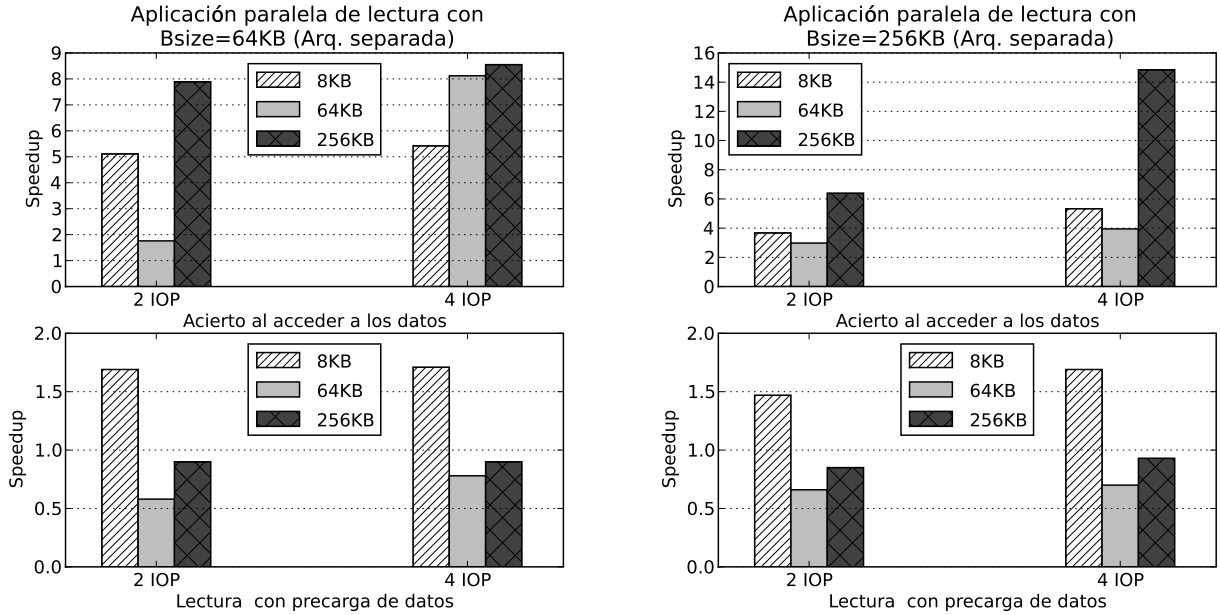

Figura 6.24: Resultados de la aplicación paralela en modo lectura ejecutando los IOP en nodos distintos a los de las aplicaciones

Otro aspecto importante es que en múltiples de casos se incrementa el rendimiento de la aplicación, aunque los ficheros no se encuentren en los proxies y haya que precargarlos (como es el caso de los accesos de pequeño tamaño).

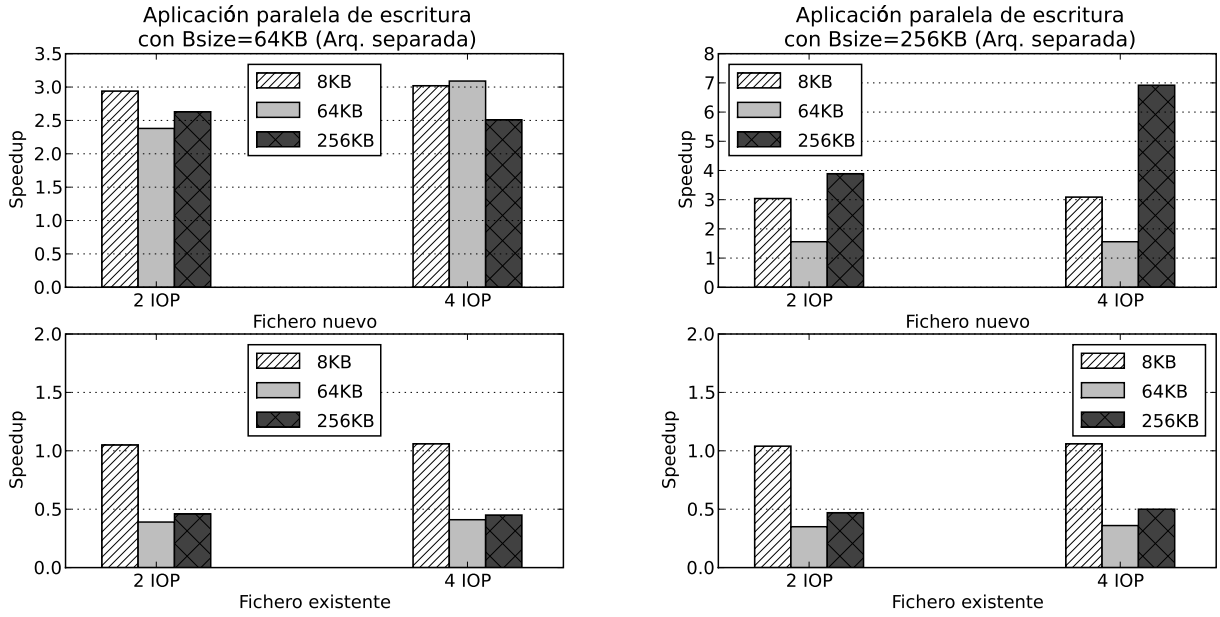

Figura 6.25: Resultados de la aplicación paralela en modo escritura ejecutando los IOP en nodos distintos a los de las aplicaciones

6.2.4. Evaluación de cargas de trabajos

Se pretende evaluar cargas de trabajos que emulen el comportamiento de programas para el análisis de datos, realizando múltiples lecturas sobre un conjunto determinado de datos. En este caso se evalua el rendimiento de cargas de trabajo suponiendo que una parte de los datos se encuentre directamente precargada en la arquitectura propuesta. Se analizarán distintas tasas de acierto: desde 0 % que indicará que los datos no están precargados, hasta 100 % que indicará que todos los datos solicitados se encuentran disponibles en los IOP.

Para el estudio se han establecido los siguientes parámetros en la arquitectura de IOP:

- Número de IOP: 2, 4 y 8
- Tamaño de bloque de los IOP: 64 KB
- Tamaño de transferencia de datos entre el servidor central y los IOP: 64 KB

La carga de trabajo consiste en 256 programas de tipo *batch* que leen individualmente un fichero de 100 MB. Se han utilizado distintos tamaños de acceso a los datos para las pruebas:

- Pequeños: 8 KB
- Medianos: 64 KB

- Grandes: 256 KB

Igual que en el caso anterior, se comparara ambas arquitecturas mediante el *speedup* para conocer la mejora del rendimiento usando la propuesta con respecto a la tradicional (compuesta por un servidor NFS).

6.2.4.1. Resultados

Los resultados obtenidos se encuentran reflejados en las Figuras 6.26, 6.27 y 6.28. Estos indican que la arquitectura propuesta es propicia para mejorar el rendimiento de cargas de trabajo de este tipo.

Los accesos de pequeño tamaño (8 KB) son los más beneficiados al usar un número escaso de IOP, mientras que los de gran tamaño (256 KB) son lo que obtienen mejores resultados al aumentar el número de IOP.

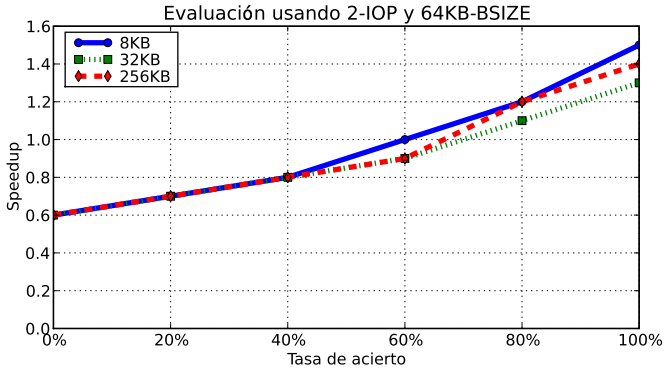

Figura 6.26: Resultados obtenidos usando 2 IOP

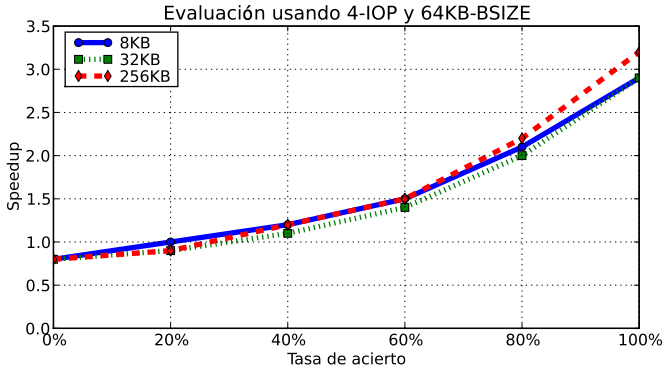

Figura 6.27: Resultados obtenidos usando 4 IOP

Por otra parte, el incremento en el número de IOP permite una mejora sustancial del *speedup*, aumentando el *speedup* máximo (de un 5,9 cuando contamos con 8 IOP en la arquitectura intermedia). Esto se debe a que este incremento el grado de paralelismo en la E/S del sistema, lo que reduce los cuellos de botella de la E/S. De

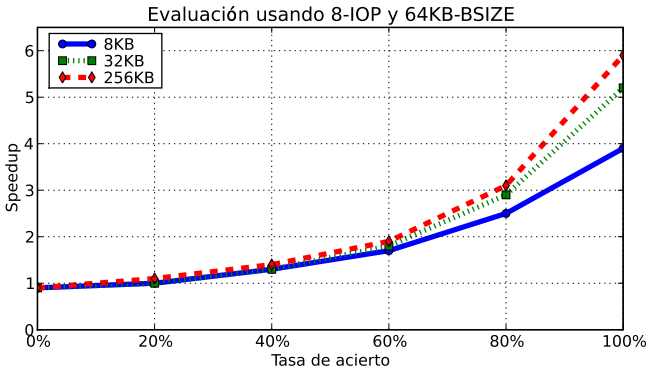

Figura 6.28: Resultados obtenidos usando 8 IOP

esta manera la tasa de acierto necesaria para obtener una mejora en el sistema es menor según aumenta el grado de paralelismo, pasando de una tasa de acierto del 60 % usando 2 IOP a un 20 % con 8 IOP.

Para tasas de acierto bajas, el uso de un número bajo de IOP, logra una mejora en los resultados al disminuir el número de transferencias de datos entre los IOP, lo que permite concentrar las peticiones de E/S y hacer un uso más eficiente de los recursos locales de los IOP (disco, caché y red).

6.2.5. Evaluación de grandes entornos *cluster*

Para la evaluación de grandes entornos *cluster* se ha realizado un modelo analítico implementado en computador, de los distintos elementos de la arquitectura de sistemas intermedios de almacenamiento. Los sistemas de almacenamiento de los *clusters* se caracterizan por su complejidad y por el gran conjunto de elementos que lo componen: red, discos, planificadores, etc. Esta complejidad puede convertir la evaluación de grandes entornos en un proyecto inabarcable. Así, podemos encontrar múltiples trabajos que modelizan sólo partes concretas de los sistemas de almacenamiento o de los sistemas de discos múltiples [122, 123, 124, 125].

Sin embargo, otros trabajos han analizado en conjunto los distintos elementos de estos sistemas [126, 127, 128], permitiendo la generalización y la simplificación de los modelos para la caracterización de los sistemas de almacenamiento, lo que permite la evaluación de estos sistemas mediante herramientas de simulación/emulación.

A continuación se detallará la construcción del modelo analítico utilizado para el análisis de distintas arquitecturas de E/S en grandes *clusters*.

6.2.5.1. Diseño del modelo analítico

Para la construcción del modelo analítico, se ha utilizado como referencia el modelo presentado en [126] donde se evalúan sistemas de almacenamiento mediante modelos de colas. En este caso, el sistema de almacenamiento se define mediante un conjunto de módulos (red, caché y disco) agrupados en servidores de datos, pudiendo

caracterizarse esquemas de almacenamiento RAID 0, modelizando el comportamiento de un sistema de ficheros paralelo, como es PVFS.

En el desarrollo de la arquitectura analizada, se han definido los siguientes módulos:

- Aplicaciones: realizan peticiones de E/S, tanto a los sistema de almacenamiento intermedio como al sistema de almacenamiento central.
- *I/O proxies*: situados entre las aplicaciones y los nodos de E/S, que almacenan y gestionan los datos manejados por las aplicaciones.
- Nodos de E/S: componen el sistema de almacenamiento.

Para la evaluación del comportamiento de los distintos módulos de la arquitectura, es necesario disponer de submódulos que agrupan los distintos elementos ya mencionados. Aunque en los sistemas nos podemos encontrar numerosos elementos como son los planificadores de disco, gestores de caché, etc. para la realización de la arquitectura se ha procedido a una simplificación de los sistemas de almacenamiento mediante una generalización y unificación los distintos elementos presentes, para favorecer el entendimiento y la evaluación de grandes entornos.

La Figura 6.29 representa el diagrama de estados usado para definir el comportamiento de una aplicación, que dispone de cuatro estados diferenciados: `Inicio`, `Fin`, `Operación` y `Red`. Cabe destacar que el submódulo `Operación` implementa el comportamiento de un cliente de un SFP. La Tabla 6.1 define cada una de las transiciones indicadas en la Figura 6.29. En la Figura 6.29 un IOS representa un servidor de E/S genérico, que puede representarse tanto por un ION como por un IOP.

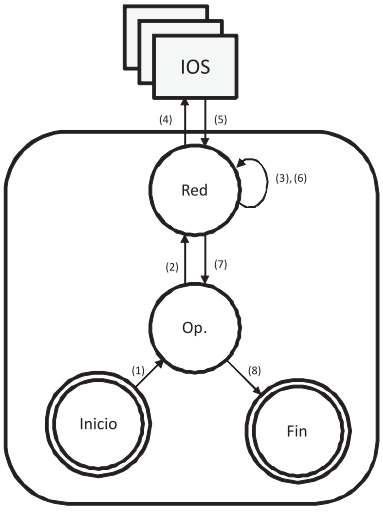

Figura 6.29: Diagrama de estados de una aplicación.

En el caso de los servidores de E/S (tanto ION como IOP) se han establecido tres submódulos diferenciados, que conforman el comportamiento de cada uno de

los módulos definidos anteriormente: red, caché y dispositivos de almacenamiento. En la Figura 6.30 se define el comportamiento de un servidor de datos E/S (ION). En el caso de un ION, el `emisor` puede ser tanto una aplicación como un IOP. La Tabla 6.2 incluye las descripciones de las transiciones reflejadas en la Figura 6.30.

Transición	Descripción
1	Comienzo de la aplicación.
2	Una vez establecida la operación a realizar (abrir fichero, cerrar fichero, escribir o leer de fichero), se envía la petición al módulo `Red`.
3	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`.
4	Se envía la petición al servidor de E/S.
5	Se recibe la respuesta del servidor.
6	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`.
7	Se envía la respuesta al módulo `Operaciones`.
8	En caso de que se cumpla la condición de finalización, se termina la ejecución de la aplicación.

Tabla 6.1: Transiciones del módulo `aplicación`

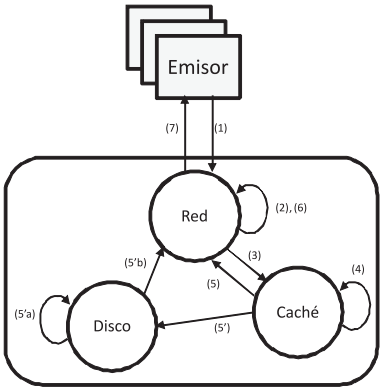

Figura 6.30: Diagrama de estados de un servidor de E/S estándar.

El diagrama de estados (Figura 6.31) repesenta el comportamiento de un IOP. Éste es similar al de un IOS pero incorpora un nuevo submódulo denominado `Gestor`, que realiza las labores de control y tranferencia de datos entre un IOP y los ION (y también entre los propios IOP), en caso de que un fichero no se encuentre en el IOP o se tenga que realizar un volcado de los datos almacenados a los ION.

Transición	Descripción
1	El emisor envía la petición al servidor de E/S.
2	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`. Si la operación es de apertura o cierre de un fichero, se genera un mensaje de respuesta.
3	Si la petición es un operación de E/S (escritura o lectura), se envía la petición al módulo `Caché`.
3'	Se envía respuesta a la operación de apertura o cierre de fichero.
4	Se procede al cálculo del tiempo de espera de la petición en el módulo `Caché`.
5	En caso de encontrase el dato en caché, se envía la respuesta al módulo `Red`.
5'	En caso de fallo de caché, se envía la petición al módulo `Disco`.
5á	Se procede al cálculo del tiempo de espera de la petición en el módulo `Disco`.
5'b	Se envía la respuesta al módulo `Red`.
6	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`.
7	Se envía la respuesta final al emisor.

Tabla 6.2: Transiciones del módulo ION

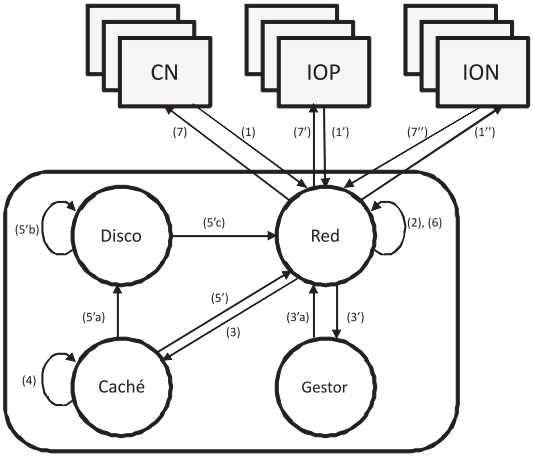

Figura 6.31: Diagrama de estados de un servidor intermedio de datos.

Además, este submódulo introduce los algoritmos de localización y distribución de datos tanto en los ION como en los IOP. Los mensajes que se establecen entre los IOP e ION y el IOP, al tratarse de una operación de E/S, son siempre tratados

como una petición más (pasando por los submódulos de `Caché` y de `Disco` si es necesario), para posteriormente establecerse una respuesta en el submódulo `Gestor`.

En la construcción del modelo, se han definido las siguientes restricciones:

- En el comportamiento de los submódulos (red, caché y disco) no existe paralelismo, lo que implica que cuando una petición se encuentre ejecutando en un submódulo, la siguiente no podrá ser atendida hasta que se complete la que se encuentre en el submódulo (tiempo de servicio).

Transición	Descripción
1	Una aplicación envía una petición al IOP.
1'	Un IOP envía una petición o respuesta al IOP.
1"	Un ION envía una respuesta al IOP.
2	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`. Si la operación es de apertura o cierre de un fichero, se genera un mensaje de respuesta.
3	Si la petición es un operación de E/S (escritura o lectura), se envía la petición al módulo `Caché`.
3'	Se envía respuesta a la operación de apertura o cierre de fichero.
3"	En caso de no encontrarse el fichero en el IOP o de se encuentre en curso una petición de E/S realizada por el `Gestor`, se envía la petición al módulo `Gestor`.
3"a	El módulo `Gestor` envía una petición a un ION.
4	Se procede al cálculo del tiempo de espera de la petición en el módulo `Caché`.
5	En caso de encontrase el dato en caché, se envía la respuesta al módulo `Red`.
5'	En caso de fallo de caché, se envía la petición al módulo `Disco`.
5á	Se procede al cálculo del tiempo de espera de la petición en el módulo `Disco`.
5'b	Se envía la respuesta al módulo `Red`.
6	Se procede al cálculo del tiempo de espera de la petición en el módulo `Red`.
7	Se envía la respuesta final al CN.
7'	Se envía la petición o respuesta a otro IOP.
7"	Se envía una nueva petición a un ION.

Tabla 6.3: Transiciones del módulo IOP

- Todos los nodos del sistema, tanto de los servidores de E/S del sistema de almacenamiento como los intermedios de almacenamiento poseen características similares en cuanto a rendimiento de cada uno de sus submódulos (red, caché y disco).
- Las capacidades tanto de lectura como de escritura del sistema son equivalentes.
- En el sistema de almacenamiento, los datos se encuentran repartidos entre los distintos servidores de datos formando un RAID 0.
- Las colas de peticiones presentes en los submódulos siguen una política FIFO para la atención de estas peticiones.
- La red es *full-duplex*.
- Los nodos de cómputo disponen de dos tipos de redes distintas: una para la comunicación entre distintos nodos de cómputo y otra para la transmisión de datos entre los nodos de cómputo y el sistema de almacenamiento.
- Las operaciones básicas que se pueden realizar sobre un fichero son: abrir, cerrar, leer datos y escribir datos.
- Tanto los clientes como los servidores intermedios de datos no realizan transmisiones de datos a varios servidores de datos al mismo tiempo.
- El tamaño máximo de lectura o escritura de datos por parte de un cliente sobre un servidor de datos o un servidor intermedio será el determinado por el tamaño de bloque del servidor.

La Tabla 6.4 contiene los parámetros y variables usados para el diseño del modelo analítico.

A continuación se definen las formulas usadas para el cálculo interno de los valores del modelo:

$$Tnet(size) = (\lceil \frac{size}{PSnet} \rceil) Lnet + \frac{size}{BWnet} \tag{6.2}$$

$$Tcache(size) = \frac{(\lceil \frac{size}{Bcache} \rceil) Bdisk}{BWcache} \tag{6.3}$$

$$Nblocks(size) = \lceil \frac{size}{Bcache} \rceil \tag{6.4}$$

$$Ndiskblocks = (\lceil \frac{size}{Bdisk} \rceil)(1 - Pcache) \tag{6.5}$$

$$Tdisk(size) = TSdisk + \frac{Ndiskblocks \cdot Bdisk}{BWdisk} \tag{6.6}$$

Parámetro	Descripción
Nodos de cómputo	Número de nodos de cómputo usados en el *cluster* evaluado
Nodos de E/S	Número de nodos de E/S usados en el sistema de almacenamiento del *cluster* evaluado
Nodos IOP	Número de nodos de almacenamiento intermedio
Red	Tipo de red utilizada para intercomunicar los nodos del *cluster*
BWnet	Ancho de banda de la red (MB/s)
Lnet	Latencia de la red (μ s)
PSnet	Tamaño de paquete (Bytes)
BWcache	Ancho de banda de la caché (MB/s)
Pcache	Probabilidad de que el dato se encuentre en caché (%)
BWdisk	Ancho de banda de la red (MB/s)
TSdisk	Tiempo de búsqueda de disco (ms)
Bdisk	Tamaño de bloque de disco (Bytes)
Piop	Probabilidad de que los datos se encuentren en los servidores intermedios de datos (%)
BSiop	Tamaño de bloque usado por los servidores intermedios de datos
BSios	Tamaño de bloque usado por los servidores de datos (KBytes)
$Tnet(size)$	Tiempo de servicio de la red (s)
$Tcache(size)$	Tiempo de servicio de la caché (s)
$Tdisk(size)$	Tiempo de servicio del disco (s)

Tabla 6.4: Parámetros del modelo

A continuación se formalizan los algoritmos utilizados para la construcción del modelo analítico. Primero, se detallan los algoritmos usados para el cálculo de cada uno de los submódulos, y posteriormente se describen los modelos diseñados:

- Modelo estándar de almacenamiento, donde se disponde de un sistema de fichero paralelo tradicional (Figura 6.32).
- Modelo basado en el empleo de servidores intermedios (Figura 6.33).

6.2.5.2. Resultados

En esta sección se muestran los resultados obtenidos al comprarar un sistema de almacenamiento tradicional basado en un sistema de ficheros paralelo similar a

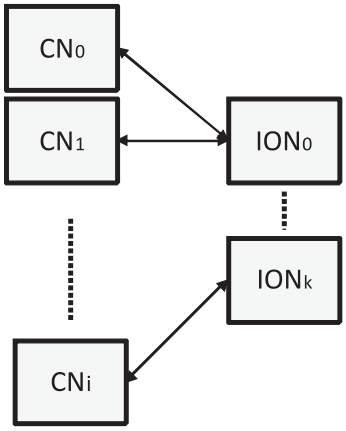

Figura 6.32: Esquema del modelo de almacenamiento estándar.

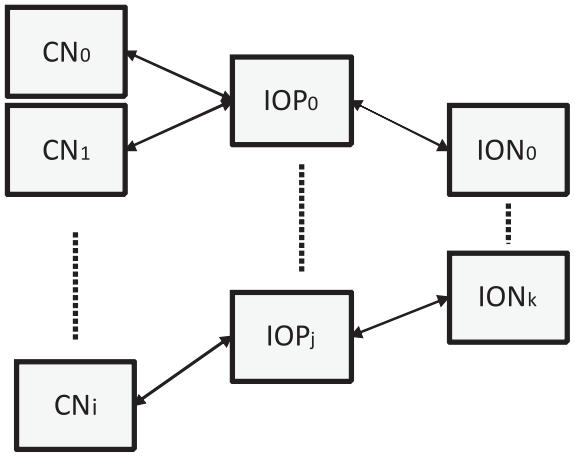

Figura 6.33: Esquema del modelo de servidores intermedios.

PVFS como sistema de almacenamiento del *cluster*, con respecto a los obtenidos usando los nodos de cómputo como sistemas de almacenamiento intermedio.

Para la evaluación, se ha establecido una plataforma que corresponda con los estándares actuales en la construcción de los grandes *clusters*. Esta arquitectura está basada en los datos disponibles del proyecto TOP500 [3]. Uno de los factores mas importantes a decidir ha sido el ratio entre el número de nodos de cómputo y el número de nodos de E/S. Para tomar esa determinación se ha tomado como referencia el supercomputador MareNostrum [6], el cuál se estima como un estándar en estos entornos de computación. A partir de los datos de la Tabla 1.1, se ha optado por construir un entorno con un ratio por nodos de cómputo de 32 (ver Tabla 6.5).

Para el análisis se ha dispuesto una carga de trabajo compuesta por 10000 aplicaciones de análisis de datos, que utilizan ficheros con tamaño de 100 MB. Estas aplicaciones emplearán distintos tamaños de acceso a los datos. En la Tabla 6.6 se han establecido los siguientes valores y para cada uno de los parámetros del entorno, así como de las variables utilizadas en cada una de las pruebas. Algunos parámetros y variables disponen de varios valores que serán estudiados en cada una de las pruebas

a realizar.

Nombre	Ratio de nodos de cómputo por nodo de E/S
Arquitectura evaluada	32
Marenostrum[6]	57,05
Magerit[13]	100
IBM ASCI Purple[7]	11
JUGENE[129]	120

Tabla 6.5: Ratio entre nodos de cómputo y nodos de E/S.

Parámetro o variable	Valores
Nodos de cómputo	1024
Nodos de E/S	32
Nodos IOP	64, 128, 256, 512
Red	Gigabit (1000Mbits/s)
Tamaño de acceso de las aplicaciones	8 KB, 32 KB, 256 KB, 4 MB
Número de trabajos	10000
Tamaño de los datos	100 MB
BWnet	1000 Mbits/s
Lnet	150 μs
PSnet	1500 Bytes
BWcache	800 MB/s
Pcache	70 %
BWdisk	40MB/s
TSdisk	5 ms
Bdisk	4096 Bytes
Piop	0 %, 20 %, 40 %, 60 %, 80 %,100 %
BSiop	512KB, 256 KB, 128KB y 64 KB
BSios	256 KB y 64 KB

Tabla 6.6: Valores de los parámetros y variables de las evaluaciones

Se han establecido 3 arquitecturas de almacenamiento evaluadas:

- Arquitectura estándar de almacenamiento: formada por la unión de distintos nodos de E/S.
- Arquitectura basada en el uso de nodos intermedios de almacenamiento, donde encontramos dos posibilidades:
  - Utilizar los nodos de cómputo como nodos de almacenamiento intermedio y de ejecución de aplicaciones. Esta estrategia permite un mejor aprovechamiento de los nodos de cómputo. Sin embargo, puede interferir en el rendimiento de las aplicaciones al utilizar recursos compartidos (CPU, memoria, etc.).
  - Disponer de un conjunto de nodos de cómputo sólo dedicados a funcionar como nodos intermedios de datos. Esto reduce el número de nodos pero permite disponer de nodos de almacenamiento intermedio de dedicación exclusiva.

A continuación se mostrarán los resultados obtenidos a partir de las arquitecturas citadas. Las Gráficas se encuentran agrupadas por tamaño de reparto utilizado, variando el número de IOP disponibles en cada caso. El eje de las abscisas representa la tasa de aciertos en los IOP (un acierto se corresponde con una apertura de un fichero que se encuentra en los proxies), mientras que el eje de las ordenadas representa el *speedup* conseguido por parte de la carga de trabajo. El *speedup* se ha medido como el resultado de la división entre el tiempo obtenido usando el sistema de almacenamiento tradicional (T_{IO}) y el tiempo usando la arquitectura de nodos intermedios de almacenamiento (T_{IOP}), tal y como se refleja en la formula 6.7. En cada Gráfica se muestran los resultados variando el tamaño de acceso.

$$Speedup = (\frac{T_{IO}}{T_{IOP}}) \tag{6.7}$$

Las Figuras 6.34, 6.35, 6.36 y 6.37 representan el *speedup* conseguido por las aplicaciones compartiendo los nodos de cómputo con los IOP (arquitectura conjunta de datos). Mientras que las figuras 6.38, 6.39, 6.40 y 6.41 representan un modelo separado entre los IOP y las aplicaciones (arquitectura separada de datos), donde las aplicaciones y los IOP se encuentran localizados en nodos de cómputo distintos.

Como muestran los resultados, el *speedup* conseguido usando la arquitectura de nodos intermedios de almacenamiento es superior a 1 en numerosas situaciones, lo que implica un aumento en el rendimiento de las aplicaciones y en general del sistema de E/S del *cluster*. A continuación detallaremos con más profundidad los resultados obtenidos:

- Según aumenta la tasa de aciertos de la arquitectura, el *speedup* aumenta, obteniendo resultados por encima del 1 con tasas de aciertos comprendidos entre 30 % y 40 % en media.
- Se puede apreciar que la arquitectura de nodos intermedios de almacenamiento es beneficiosa cuando el tamaño de acceso es pequeño (8KB), ya que el sistema

de nodos intermedios aumenta el paralelismo (al disponer de un mayor número de nodos de E/S) y agrupa las peticiones de los clientes cuando acceden a datos no presentes en los IOP.

- El uso de arquitecturas separadas de IOP y CN mejora el rendimiento del sistema ya que se produce una menor sobrecarga en los nodos (tanto IOP como CN), lo que compensa un mayor número de comunicaciones entre estos. Esto se aprecia especialmente en el caso de accesos de gran tamaño (256KB), como se refleja en las Figuras 6.36 y 6.40. Por el contrario, los accesos de pequeño tamaño (8KB) obtienen mejoras sustanciales con bajas tasas de acierto.

- El tamaño de reparto utilizado en la distribución de los datos supone un factor importante a tener en cuenta a la hora de configurar el sistema. Así en las arquitecturas separadas, el uso de un mayor tamaño de reparto mejora el sistema al agrupar las peticiones, como se puede observar en las Gráficas 6.38 y 6.41, que representan dos casos extremos de una misma configuración.

- En el caso de arquitecturas conjuntas, un menor tamaño de reparto permite aumentar la probabilidad de acceder a los datos locales, lo que mejora el rendimiento en caso de tasas de acierto muy elevadas. Mientras que con menores tasas de acierto el rendimiento es peor que con mayores tamaños de reparto, debido a una menor agregación de las peticiones.

- Un incremento del número de IOP mejora el rendimiento de los accesos medio (32KB) y gran (256KB) tamaño. Por otro lado, un escaso número de IOP sólo beneficia a los accesos de pequeño tamaño (8KB), pudiéndose obtener *speedup* mayores que 1 con bajas tasas de acierto debido a la agregación de las peticiones.

Este estudio permite augurar buenos resultados en entornos similares a los presentados en una infraestructura real. Esto representaria un aumento de la productividad en grandes entornos *cluster*.

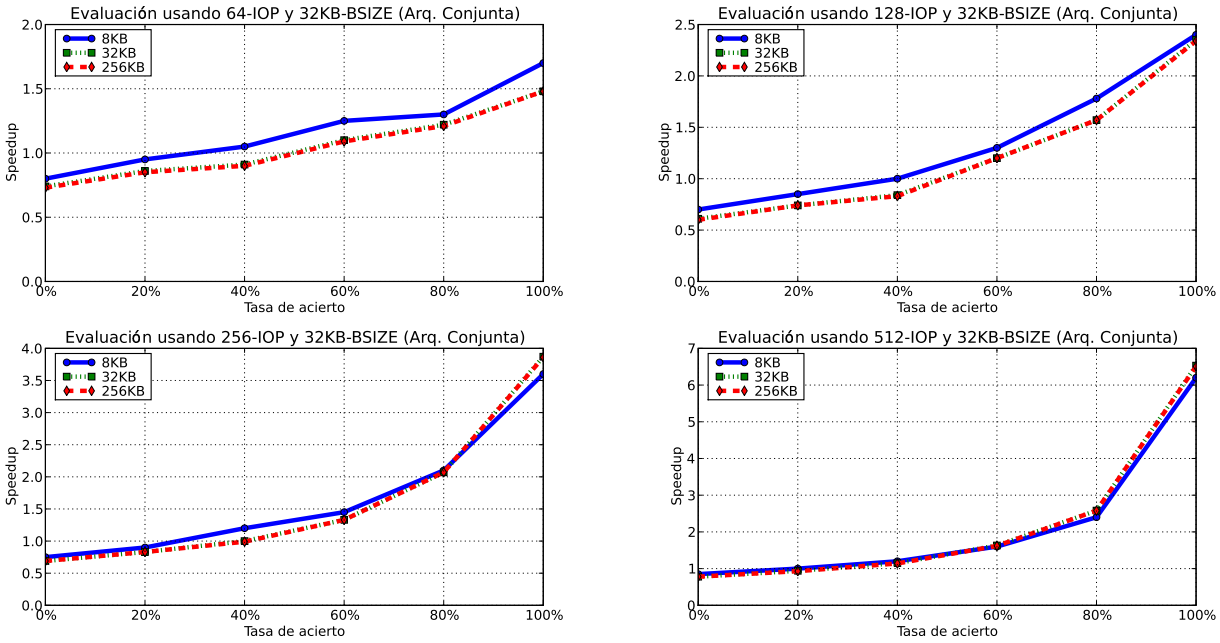

Figura 6.34: Resultados de las arquitecturas conjuntas, tamaño de reparto de 32KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

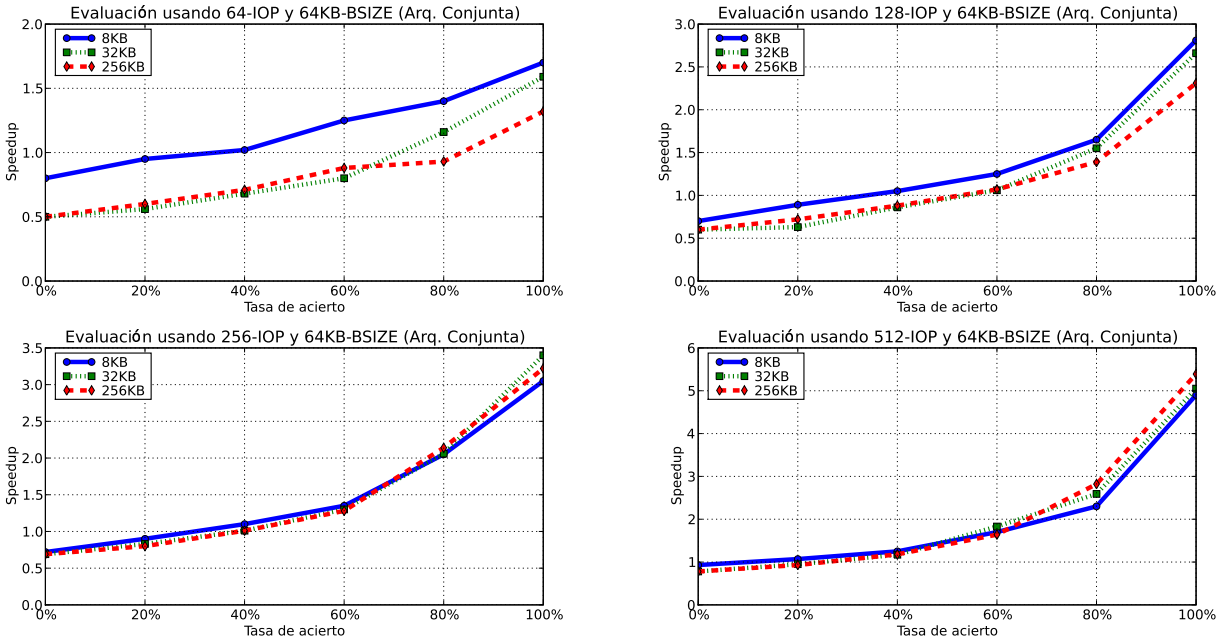

Figura 6.35: Resultados de las arquitecturas conjuntas, tamaño de reparto de 64KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

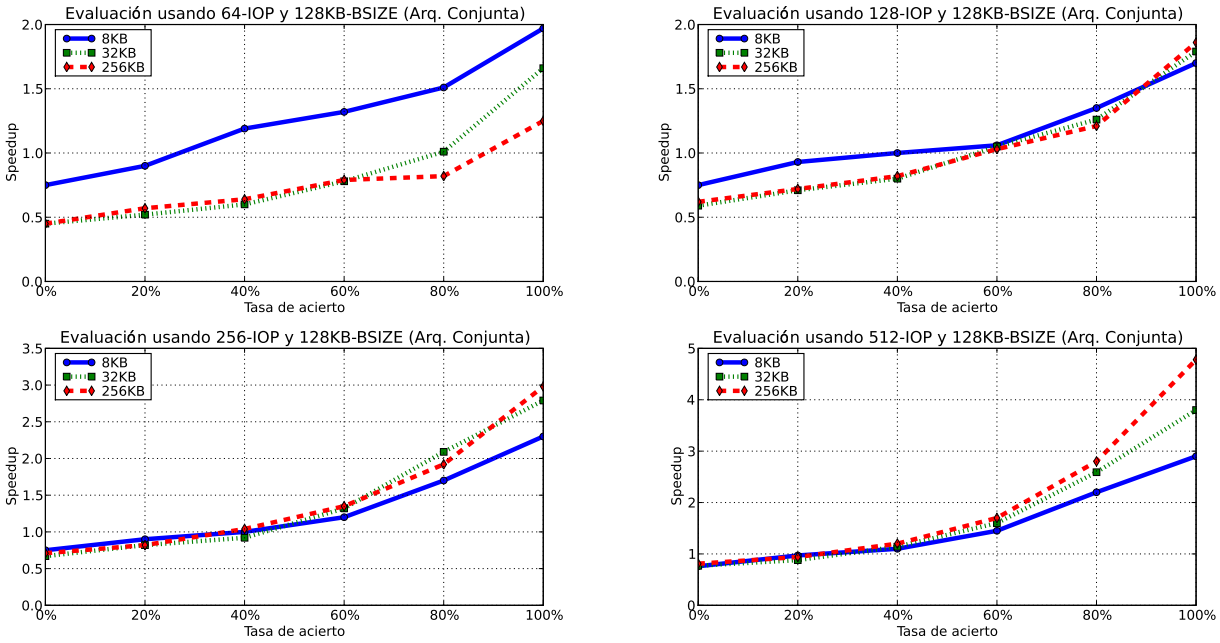

Figura 6.36: Resultados de las arquitecturas conjuntas, tamaño de reparto de 128KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

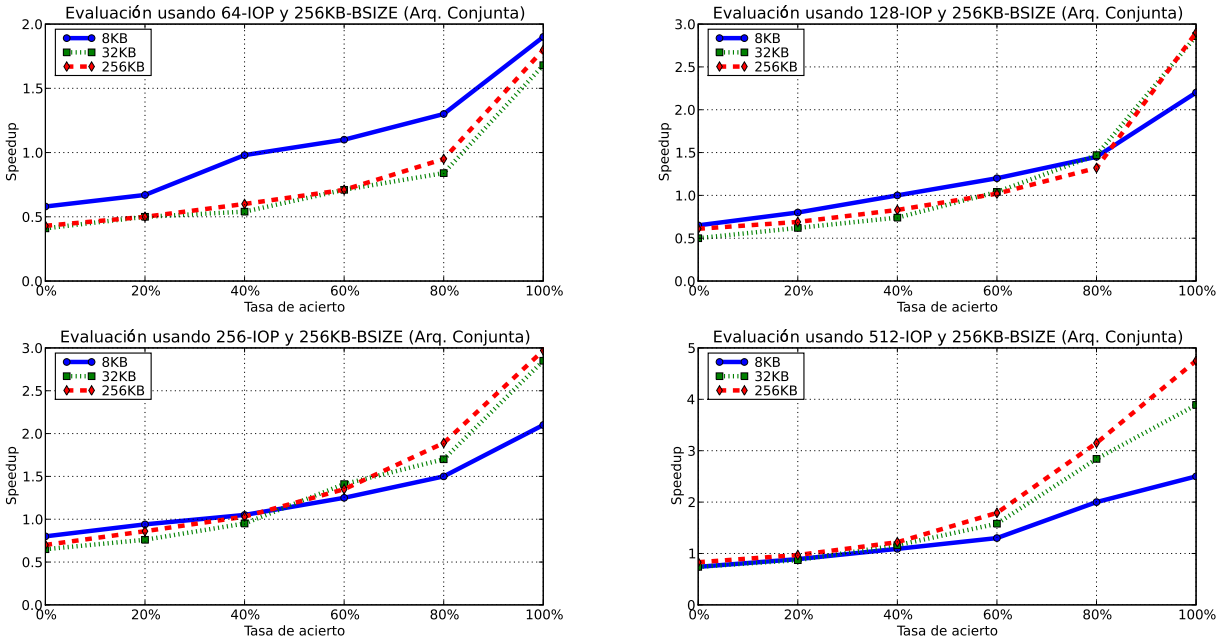

Figura 6.37: Resultados de las arquitecturas conjuntas, tamaño de reparto de 256KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

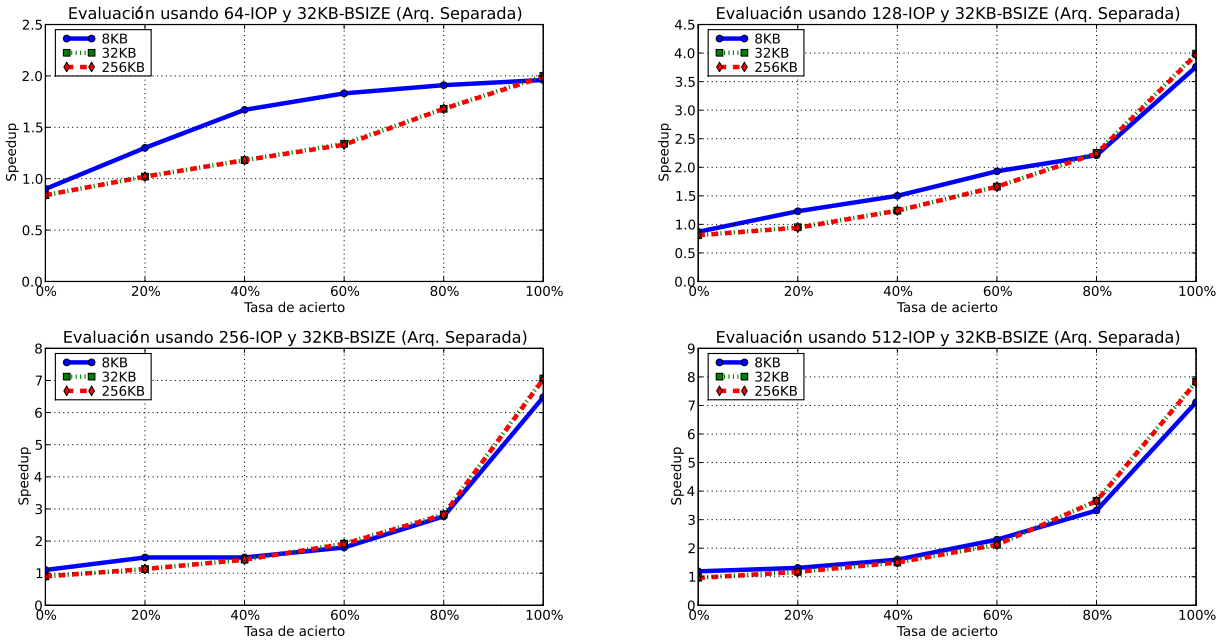

Figura 6.38: Resultados de las arquitecturas separadas, tamaño reparto 32KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

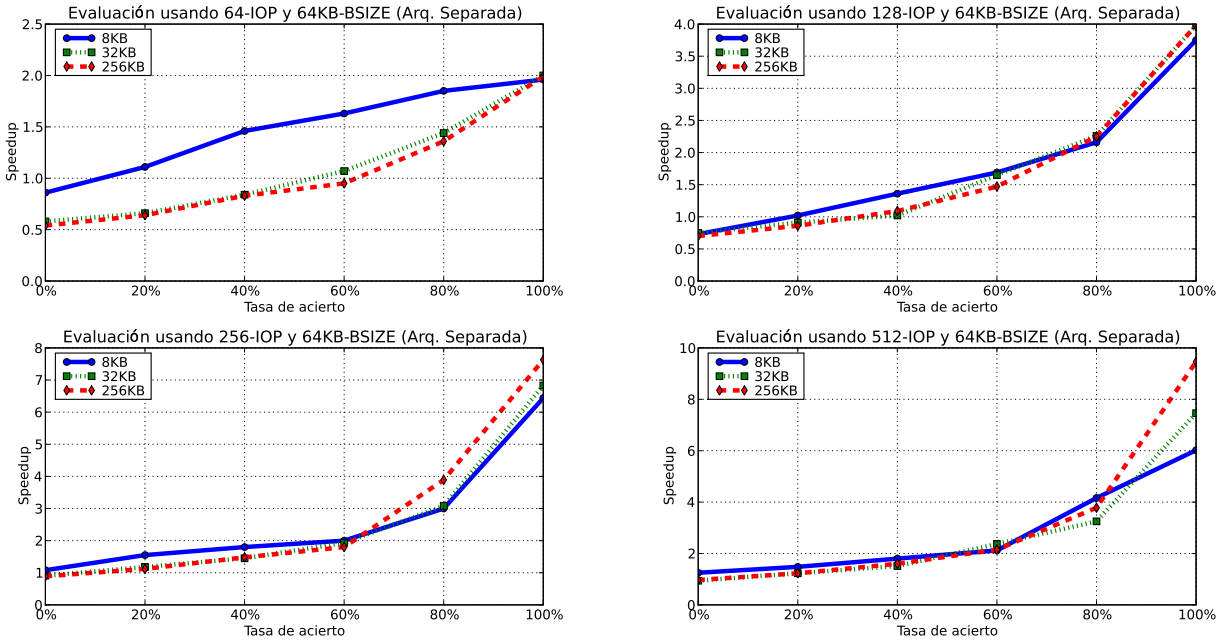

Figura 6.39: Resultados de las arquitecturas separadas, tamaño reparto 64KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

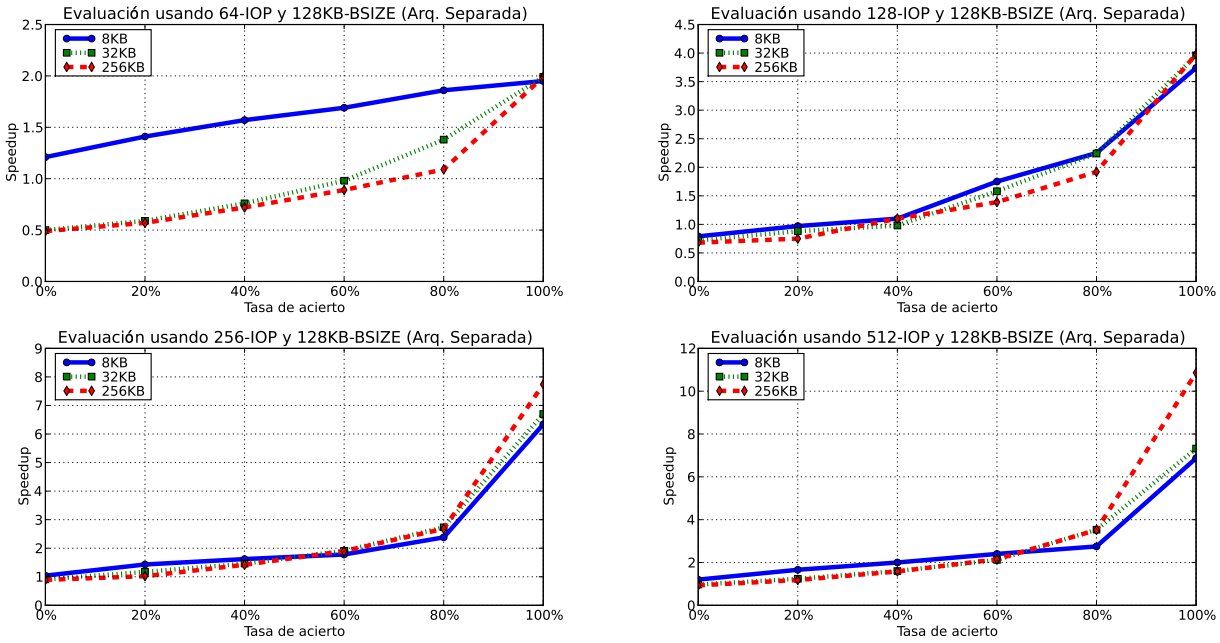

Figura 6.40: Resultados de las arquitecturas separadas, tamaño reparto 128KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

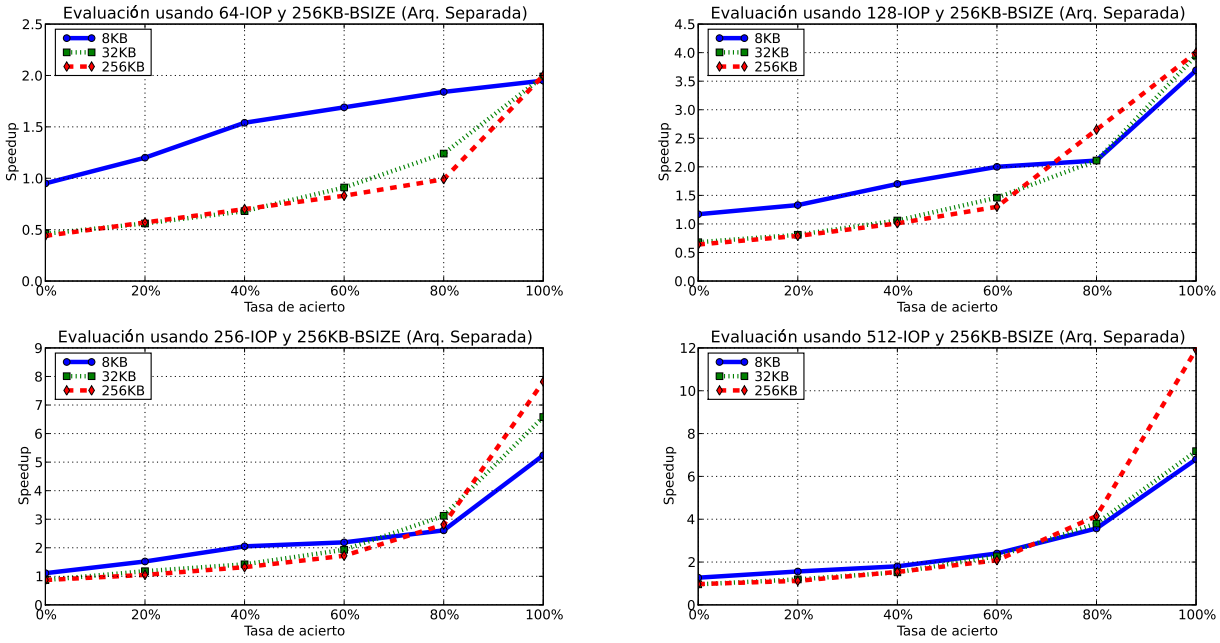

Figura 6.41: Resultados de las arquitecturas separadas, tamaño reparto 256KB para tamaños de acceso de 8 KB, 32 KB y 256 KB

6.3. Resumen

En este capítulo se han diseñado diferentes escenarios para la realización de las pruebas necesarias para medir y comprobar la eficacia y el rendimiento del sistema de ficheros Expand y de la arquitectura de E/S propuesta para sistemas de almacenamiento en *clusters*.

La evaluación del sistema de ficheros paralelo Expand no sólo ha conseguido demostrar que es posible el empleo de servidores de datos estándar para la construcción de sistemas de ficheros paralelos en entornos *cluster*, sino que además los resultados han sido muy satisfactorios y competitivos, como lo demuestra la comparación de estos con los obtenidos mediante dos sistemas de ficheros paralelos muy representativos y usados en entornos *cluster*: PVFS y GPFS.

Por otra parte, el empleo de un sistema de almacenamiento a dos niveles, uno perteneciente al sistema de almacenamiento del *cluster*, y otro virtual creado mediante la unión de distintos nodos de cómputo del *cluster*, se ha mostrado muy eficiente en variadas situaciones presentados en el capítulo. Se han podido conseguir grandes incrementos en el rendimiento de los sistemas de almacenamiento estándar. Además, mediante un modelo analítico descrito en este mismo capítulo, se han podido realizar evaluaciones teóricas de los sistemas de almacenamiento pertenecientes a grandes *clusters*, de manera que se puede determinar la factibilidad de la arquitectura propuesta en este tipo de entornos de computación.

Capítulo 7

Conclusiones y trabajo futuro

La consecución de la presente tesis ha alcanzado de una forma adecuada los objetivos fijados al comienzo:

- Proporcionar un acceso homogéneo a los datos en entornos heterogéneos, usando tecnologías estándar de E/S para la construcción del sistema de almacenamiento.

- Equilibrar la carga de E/S producida por las aplicaciones y eliminar la sobrecarga de los sistemas de almacenamiento en entornos de gran escala.

Los resultados obtenidos en la tesis doctoral permiten afirmar que estos objetivos de la tesis se han conseguido: *se ha propuesto un sistema de ficheros paralelo, denominado Expand, que permite la escalabilidad de los sistemas de almacenamiento en entornos cluster* y *se ha definido una arquitectura escalable para grandes entornos cluster que amplía la jerarquía de memoria de los computadores.*

A continuación se recopilan las conclusiones obtenidas. En este capítulo se procederá a repasar las contribuciones de la tesis. Posteriormente, se indicarán las publicaciones realizadas sobre los trabajos expuestos. Por último se enumerarán los posibles trabajos futuros que podrían continuar con el realizado, los cuales serán de gran interés para nuevas investigaciones.

7.1. Contribuciones de la tesis

Esta tesis doctoral presenta contribuciones al estudio, diseño y mejora de los sistemas de almacenamiento para *clusters*. Realiza dos importantes aportaciones para estos sistemas:

- La definición de un sistema de E/S paralela usando como base servicios de E/S estándar, que permite la compatibilidad y adaptibilidad del sistema en entornos de computación heterogéneos.

- El diseño de una nueva arquitectura de E/S para grandes entornos de computación que posibilita mejorar la escalabilidad de los sistemas de almacenamiento, equilibrando, agrupando y gestionando la carga de E/S generada por las aplicaciones hacia los nodos de E/S de los sistemas de almacenamiento.

Estos dos enfoques proporcionan nuevas oportunidades en la construcción de sistemas de almacenamiento para entornos *cluster*.

A continuación, se resumirán las aportaciones presentadas y desarrolladas en la tesis:

- *Definición de un sistema de ficheros paralelo basado en el uso de servidores de datos estándar.*

  Se ha diseñado un sistema de ficheros paralelo que utiliza servidores de datos estándar para su construcción. En el diseño de este sistema de ficheros paralelo se han estudiado distintos aspectos, como son: las definiciones de partición y fichero en Expand, los diferentes mecanismos para la distrbución y localización de datos y metadatos, la estructura de directorios, etc.

- *Diseño de distintas interfaces de E/S en Expand.*

  Se han definido y diseñado distintas interfaces de E/S para el sistema de ficheros paralelo Expand. Estas interfaces se han adaptado e incorporado en otras interfaces estándar como son MPI-IO o FUSE, lo que permite su uso en distintos entornos de computación.

- *Evaluación de Expand y comparativa con otros sistemas de almacenamiento.*

  Se ha realizado un estudio comparativo de distintos sistemas de ficheros usados en entornos *cluster*. Para ello se han evaluado diferentes aspectos, como son: el rendimiento de las operaciones de E/S y de la gestión de metadatos, el uso de estos sistemas en entornos de computación paralela, etc. Las conclusiones de esta evalución permiten afirmar que el uso del sistema de fichero paralelo Expand tiene rendimientos similares o incluso superiores a sistemas de ficheros paralelos que utilizan servidores de E/S dedicados.

- *Diseño de una arquitectura de almacenamiento para grandes entornos cluster basada en el uso de nodos intermedios de almacenamiento.*

  Se ha diseñado una arquitectura de almacenamiento escalable para grandes entornos de computación, basada en el uso de sistemas intermedios de almacenamiento, los cuales se encuentran localizados en los nodos de cómputo. Esta solución arquitectónica, permite reducir los accesos al sistema de almacenamiento, evitando cuellos de botella e incrementando el rendimiento global

del mismo. En este diseño se han definido políticas de distribución de datos y de transferencia para el intercambio de datos entre los sistemas de almacenamiento y se han establecido sistemas de redundancia de datos para dar soporte a la tolerancia a fallos.

- *Definición de un modelo analítico y estudio teórico del rendimiento en grandes entornos de computación.*

  Se ha definido un modelo para el análisis de grandes infraestructuras *cluster*, que permite estudiar el comportamiento de un sistema de almacenamiento estándar. Posteriormente, se ha adaptado el modelo para evaluar el comportamiento de la arquitectura escalable propuesta, realizando una comparativa entre ambas arquitecturas. Los resultados teóricos obtenidos han reflejado un incremento del rendimiento al utilizar la arquitectura escalable propuesta.

- *Evaluación de la arquitectura de almacenamiento basada en nodos intermedios de almacenamiento y comparativa con arquitecturas de almacenamiento estándar.*

  Se ha realizado un estudio comparativo de la arquitectura basada en sistemas intermedios de almacenamiento con una arquitectura de almacenamiento estándar. Estas evaluaciones permiten: determinar las ventajas de la arquitectura propuesta, tanto en entornos de alto rendimiento (HPC) como en entornos de alta productividad (HTC) y medir el incremento el rendimiento de las aplicaciones al reducir la carga de los sistemas de almacenamiento.

7.2. Publicaciones relacionadas

Durante el desarrollo de esta tesis se han publicado diferentes trabajos que parten de los desarrollos y resultados obtenidos. A continuación se presentan las publicaciones, clasificados en cuatro grupos: artículos en revistas, artículos en libros, publicaciones en congresos internacionales y publicaciones en congresos nacionales.

- **Artículos en revistas**:

  - *Fault tolerant file models for parallel file systems: introducing distribution patterns for every file.* [130]
    ISSN: 0920-8542
    Revista: J. Supercomputing. vol. 47, no3, pp. 312-334
    Editor: Kluwer Academic Publishers
    Año: 2009

  - *A global and parallel file system for grids.* [131]
    ISSN: 0167-739X
    Revista: Future Generation Computer Systems. vol. 23, no1, pp. 116-122
    Editor: Elsevier
    Año: 2008

- **Artículos en libros**:

  - *Evaluation of the Expand parallel file system on a novel cluster-wide I/O architecture.* [132]
    ISBN: 88-86037-17-1
    Libro: Science and Supercomputing in Europe, 2005 Annual Report.
    Editor: CINECA
    Año: 2006

- **Artículos en congresos internacionales**:

  - *Improving the Performance of Cluster Applications through I/O Proxy Architecture.* [133]
    Congreso: CLUSTER 06
    Año: 2006

  - *A New I/O Architecture for Improving the Performance in Large Scale Clusters.* [107]
    Congreso: ICCSA 2006
    Año: 2006

  - *High Performance Java Input/Output for Heterogeneous Distributed Computing.* [114]
    Congreso: 10th IEEE Symposium on Computers and Communications (ISCC 2005).
    Año: 2005

  - *Evaluating Expand: A Parallel File System Using NFS Servers.* [114]
    Congreso: 6th World Multiconference on Systemics, Cybernetics and Informatics. (SCI'02).
    Año: 2002

- **Artículos en congresos nacionales**:

  - *Descripción de una nueva arquitectura de E/S para grandes clusters.* [134]
    Congreso: XIX Jornadas de Paralelismo.
    Año: 2008

  - *Arquitectura escalable para E/S de altas prestaciones en sistemas heterogéneos.* [111]
    Congreso: XV Jornadas de Paralelismo.
    Año: 2004

7.3. Trabajo futuro

El trabajo presentado en esta tesis puede ser profundizado en diferentes aspectos. Los más destacados son:

- Respecto al sistema de ficheros Expand:
  - Estudio, diseño e integración de nuevos protocolos para el acceso a datos, como por ejemplo, CIFS, etc.
  - Integración en el núcleo del sistema de ficheros.
  - Diseño de herramientas para la monitorización, administración y control de los servidores de datos, fichero y los clientes de Expand.
  - Estudio y diseño de sistemas inteligentes para configuración de particiones automáticas, que se basan en pistas proporcionadas por los clientes o las aplicaciones.
  - Despliegue de Expand en entornos distribuidos como Grid.
  - Diseño e integración de operaciones colectivas dentro de Expand para mejorar el acceso a los datos.
- Respecto a la arquitectura basada en el uso de sistemas intermedios de almacenamiento:
  - Despliegue y evaluación de la arquitectura en grandes entornos.
  - Diseño de políticas de *prefetching* y sincronización con los datos de los sistemas de almacenamiento.
  - Estudio de mecanismos inteligentes para el diseño automático de los espacios de almacenamiento en base a los accesos realizados por las aplicaciones.
  - Análisis y estudio de la arquitectura de E/S con los sistemas de colas, para permitir la reordenación de trabajos en base a los datos usados por los mismos y aprovecha mejor de los datos almacenados en los espacios virtuales de almacenamiento.
  - Evaluación de la arquitectura de E/S en sistemas de almacenamiento GPFS.
  - Diseño de mecanismos de tolerancia a fallos en la arquitectura.
  - Despliegue y evaluación de la arquitectura en entornos a gran escala como Grid o entornos colaborativos.

Bibliografía

[1] A. Shoshani, L. Bernardo, H. Nordberg, D. Rotem, and A. Sim, "Storage management for high energy physics applications," 1998. Computing in High Energy Physics, 1998.

[2] M. Ripeanu, "A note on Zipf Distribution in Top500 Supercomputers List.," 2006. IEEE Distributed Systems Online.

[3] Top500.org, "Top 500 supercomputer sites (http://www.top500.org/)," 2007.

[4] R. S. C. Cluster, "Institute of physical and chemical res. (riken)." (http://www.riken.go.jp/).

[5] V. W. Ross, "Heterogeneous high performance computer," in *DOD_UGC'05: Proceedings of the 2005 Users Group Conference*, (Washington, DC, USA), p. 304, IEEE Computer Society, 2005.

[6] Barcelona Supercomputing Center (BSC, "IBM marenostrum." (http://www.bsc.org.es/).

[7] LLNL, "IBM ASC Purple." (http://www.llnl.gov/asci/platforms/purple/).

[8] B. P. et al., "The NFS version 4 protocol," *Proceedings of the 2nd international system administration and networking conference (SANE2000)*, pp. 94–95, 2000.

[9] P.J. Leach and D. C. Naik, "A common internet file system (CIFS/1.0) protocol," 1997. Microsoft Corporation.

[10] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. of the First Conference on File and Storage Technologies (FAST)*, pp. 231–244, Jan. 2002.

[11] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, (Atlanta, GA), pp. 317–327, USENIX Association, 2000.

[12] ORACLE, "Oracle Cluster File System (OCFS)." (http://oss.oracle.com/projects/ocfs/).

[13] Cesvima:, "Magerit." (http://www.cesvima.upm.es/informacion/magerit).

[14] R. Griswold, "Storage topologies," *Computer*, vol. 35, no. 12, pp. 56–63, 2002.

[15] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun Network Filesystem," in *Proceedings Summer 1985 USENIX Conf.*, pp. 119–130, 1985.

[16] Mahadev Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 447–459, 1990.

[17] T. W. Page, G. J. Popek, R. G. Guy, and J. S. Heidemann, "The Ficus distributed file system: Replication via stackable layers," Tech. Rep. CSD-900009, University of California, 1990.

[18] L.-F. Cabrera and D. D. E. Long, "SWIFT: using distributed disk striping to provide high I/O data rates," Tech. Rep. UCSC-CRL-91-46, University of California, Santa Cruz, 1991.

[19] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless network file systems," in *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pp. 109–126, ACM Press, 1995.

[20] E. R. Zayas, "AFS-3 Programmer's reference: Architectural overview," 2001. Version 1.0. Transarc Corporation.

[21] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The sprite network operating system," *Computer Magazine of the Computer Group News of the IEEE Computer Group Society, ; ACM CR 8905-0314*, vol. 21, no. 2, 1988.

[22] P. J. Braam, "The coda distributed file system," *Linux Journal*, vol. 1998, no. 50es, p. 6, 1998.

[23] P. Braam and P. Nelson, "Removing bottlenecks in distributed filesystems: Coda intermezzo as examples," in *Proceedings of the 5th Annual Linux Expo*, pp. 131–139, 1999.

[24] Intermezzo, "Intermezzo Web page (http://www.inter-mezzo.org/)," 2004.

[25] Sun Microsystems, "SUN Web page (http://www.sun.com/)," 2004.

[26] Sun Microsystems, "RPC: remote procedure call protocol specification," Request for Comments 1050, Internet Engineering Task Force, 1988.

[27] Open Group, "Open group Web page: The NLM protocol," 2004. (http://www.opengroup.org/onlinepubs/009629799/chap10.htm).

[28] Open Group, "Open group Web page: The NSM protocol," 2004. (http://www.opengroup.org/onlinepubs/009629799/chap11.htm).

[29] Linux NFS, "Linux NFS (http://nfs.sourceforge.net/)," 2004.

[30] G. H. Kim, R. G. Minnich, and L. Mcvoy, "Bigfoot-nfs: A parallel file-striping nfs server (extended abstract)," tech. rep., Sun Microsystems Computer Corp., 1994.

[31] J.S Chase, D. Anderson, and A. Vahdat, "Interposed request routing for scalable network storage," in *Fourth Symposium on Operating System Design and Implementation (OSDI2000)*, pp. 259–272, 2000.

[32] R. Kassick, C. Machado, E. Hermann, R. Avila, P. Navaux, and Y. Denneulin, "Evaluating the performance of the dnfsp file system," in *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, (Washington, DC, USA), pp. 1032–1038, IEEE Computer Society, 2005.

[33] D. Hildebrand and P. Honeyman, "Exporting storage systems in a scalable manner with pnfs.," in *MSST*, pp. 18–27, IEEE Computer Society, 2005.

[34] S.R. Soltis, T. Ruwart, and M. O'Keefe, "The global file system," 1996. Proceedings of the Fith NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies.

[35] A. Tridgell, "SAMBA." http://www.samba.org/.

[36] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 29–43, ACM Press, 2003.

[37] H. Eckardt, "Investigation of distributed disk I/O concepts," 1990. Informe técnico, ESPRIT PUMA, Siemens.

[38] Kai Hwang et. al., "Designing SSI clusters with hierarchical checkpointing and single I/O space," *IEEE Concurrency*, vol. Volume 7, Number 1, 1999.

[39] "Lustre." (http://www.lustre.org).

[40] "Panasas." (http://www.panasas.com).

[41] "Ibrix fusion." (http://www.ibrix.com/pages/products.php).

[42] F. Isaila and W. F. Tichy, "Clusterfile: A flexible physical layout parallel file system," in *Proceedings of IEEE Cluster Computing Conference*, Jan. 2001.

[43] N. Nieuwejaar and D. Kotz, "Performance of the galley parallel file system," in *Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 83–94, 1996.

[44] C. L. Elford, J. Huber, C. Kuszmaul, and T. Madhyastha, "Portable parallel file system detailed design," tech. rep., Department of Computer Science, University of Illinois, Sept. 1994.

[45] G. A. Gibson, D. Stodolsky, P. W. Chang, W. V. Courtright II, C. G. Demetriou, E. Ginting, M. Holland, Q. Ma, L. Neal, R. H. Patterson, J. Su, R. Youssef, and J. Zelenka, "The Scotch parallel storage systems," in *Proceedings of 40th IEEE Computer Society International Conference (COMPCON 95)*, pp. 403–410, 1995.

[46] P. Corbett, S. Johnson, and D. Feitelson, "Overview of the Vesta parallel file system," 1993. ACM Computer Architecture News, 21(5):7–15.

[47] P. F. Corbett and D. G. Feitelson, "The vesta parallel file system," *ACM Trans. Comput. Syst.*, vol. 14, no. 3, pp. 225–264, 1996.

[48] J. Carretero, F. Pérez, P. de Miguel, F. García, and L. Alonso, "ParFiSys: A parallel file system for MPP," 1996. ACM Operating Systems Review, 30(2):74–80.

[49] F. Pérez, J. Carretero, F. García, P. de Miguel, and L. Alonso, "Evaluating ParFiSys: a high-performance and distributed file system," 1997. Journal of Systems Architecture. Elsevier. Vol. 43.

[50] S. A. Moyer and V. S. Sunderam, "PIOUS: A scalable parallel I/O system for distributed computing environments," in *Proceedings of the Scalable High-Performance Computing Conferece*, pp. 71–78, 1994.

[51] R. Oldfield and D. Kotz, "Armada: A parallel file system for computational grids," in *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, (Washington, DC, USA), p. 194, IEEE Computer Society, 2001.

[52] T. Jones, A. Koniges, and R. K. Yates, "Performance of the IBM general parallel file system," in *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, (Washington, DC, USA), p. 673, IEEE Computer Society, 2000.

[53] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, and A. Koniges, "MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, (New York, NY, USA), pp. 17–17, ACM Press, 2001.

[54] A. J. Smith, "Disk cache miss ratio analysis and design considerations," *ACM Trans. Comput. Syst.*, vol. 3, no. 3, pp. 161–203, 1985.

[55] D. Kotz and N. Nieuwejaar, "File system workload on a scientific multiprocessor," in *IEEE Parallel and Distributed Technology. Systems and Applications*, pp. 134–154, 1995.

[56] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing parallel file-access patterns on a large-scale multiprocessor," in *Proceedings of the Ninth International Parallel Processing Symposium*, pp. 165–172, IEEE Computer Society Press, 1995.

[57] F. García Carballeira, "Soluciones al problema de la E/S en sistemas distribuidos y paralelos," Tech. Rep. FIM/105.1/DATSI/98, Facultad de nformatica (UPM), 1998.

[58] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *USENIX Summer*, pp. 197–207, 1994.

[59] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the Sprite network file system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 134–154, 1988.

[60] John H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a distributed file system," *ACM Transactions On Computer Systems*, vol. 6, no. 1, pp. 51–81, 1988.

[61] Marshall K. McKusick, W. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for UNIX," *Computer Systems*, vol. 2, no. 3, pp. 181–197, 1984.

[62] L. W. McVoy and S. R. Kleiman, "Extent-like performance from a UNIX file system," in *Proceedings of the USENIX Winter 1991 Technical Conference*, (Dallas, TX, USA), pp. 33–43, 21–25 1991.

[63] C. S. Ellis and D. Kotz, "Prefetching in file systems for MIMD multiprocessors," in *Proceedings of the 1989 International Conference on Parallel Processing*, (St. Charles, IL), pp. I:306–314, Pennsylvania State Univ. Press, 1989.

[64] T. Kimbrel, P. Cao, E. Felten, A. Karlin, and K. Li, "Integrated parallel prefetching and caching," 1996.

[65] Hai Jin, T. Cortes, and R. Buyya, *High Performance Mass Storage and Parallel I/O*. IEEE Computer Society Press and Wiley, 2001.

[66] K. M. Curewitz, P. Krishnan, and J. S. Vitter, "Practical prefetching via data compression," in *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 257–266, ACM, 1993.

[67] P. Krishnan and J. S. Vitter, "Optimal prediction for prefetching in the worst case," *SIAM J. Comput.*, vol. 27, no. 6, pp. 1617–1636, 1998.

[68] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *J. ACM*, vol. 43, no. 5, pp. 771–793, 1996.

[69] V. S. Pai, A. A. Schäffer, and P. J. Varman, "Markov analysis of multiple-disk prefetching strategies for external merging," *Theoretical Computer Science*, vol. 128, no. 1–2, pp. 211–239, 1994.

[70] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, "A study of integrated prefetching and caching strategies," in *Measurement and Modeling of Computer Systems*, pp. 188–197, 1995.

[71] M. L. Norman, J. Shalf, S. Levy, and G. Daues, "Diving deep: Data-management and visualization strategies for adaptive mesh refinement simulations," *Computing in Science and Engg.*, vol. 1, no. 4, pp. 36–47, 1999.

[72] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed, "I/O requirements of scientific applications: An evolutionary view," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pp. 576–594, IEEE Computer Society Press and Wiley, 2001.

[73] Y. E. Cho, *Efficient resource utilization for parallel I/O in cluster environments*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999. Adviser-Marianne Winslett.

[74] "IPARS: integrated parallel accurate reservoir simulation." (http://www.ticam.utexas.edu/CSM/ACTI/ipars.html).

[75] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo, "FLASH: An adaptive mesh hydrodynamics code for modelling astrophysical thermonuclear flashes," *Astrophysical Journal Suppliment*, pp. 131–273, 2000.

[76] A. Ching, A. Choudhary, K. Coloma, W. K. Liao, R. Ross, and W. Gropp, "Noncontiguous access through MPI-IO," in *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003.

[77] A. Ching, A. Choudhary, W. K. Liao, R. Ross, and W. Gropp, "Efficient structured data access in parallel file systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.

[78] "ROMIO: A high-performance, portable MPI-IO implementation." (http://www.mcs.anl.gov/romio).

[79] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective I/O in ROMIO," in *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pp. 182–189, Argonne National Laboratory, 1999.

[80] J. del Rosario, R. Bordawekar, and A. Choundary, "Improved parallel I/O via a two-phase run-time access strategy," in *ACM Computer Architecture News*, vol. 21, pp. 31–38, 1993.

[81] R. Bordawekar, "Implementation of collective I/O in the intel paragon parallel file system: initial experiences," in *ICS '97: Proceedings of the 11th international conference on Supercomputing*, (New York, NY, USA), pp. 20–27, ACM, 1997.

[82] D. Kotz, "Disk-directed I/O for mimd multiprocessors," in *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pp. 61–74, 1994.

[83] K. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett, "Server-directed collective I/O in panda," *Proceedings of Supercomputing '95*, 1995.

[84] R. Thakur, W. Gropp, and E. Lusk, "On implementing MPI-IO portably and with high performance," in *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 23–32, 1999.

[85] T. Cortes, S. Girona, and J. Labarta, "PACA: A cooperative file system cache for parallel machines," in *Euro-Par, Vol. I*, pp. 477–486, 1996.

[86] T. Cortes, S. Girona, and J. Labarta, "Avoiding the Cache-Coherence problem in a Parallel/Distributed File System," in *Proceedings of High-Performance Computing and Networking*, pp. 860–869, 1997.

[87] K. Coloma, A. Choudhary, W. K. Liao, L. Ward, and S. Tideman, "Dache: Direct access cache system for parallel I/O," in *Proceedings of the International Supercomputer Conference*, 2005.

[88] K. Coloma, A. Choudhary, W. K. Liao, L. Ward, E. Russell, and N. Pundit, "Scalable high-level caching for parallel I/O," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2004.

[89] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*. Message Passing Interface Forum, 1994.

[90] R. Thakur, W. Gropp, and E. Lusk, "Achieving high performance with MPI-IO," Tech. Rep. ANL/MCS-P742-0299, Argonne National Laboratory, 1999.

[91] K. Kennedy, C. Koelbel, and H. Zima, "The rise and fall of high performance fortran: an historical object lesson," in *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, (New York, NY, USA), pp. 7–1–7–22, ACM, 2007.

[92] R. Rew and G. Davis, "Data management: Netcdf: an interface for scientific data access," *IEEE Comput. Graph. Appl.*, vol. 10, no. 4, pp. 76–82, 1990.

[93] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific I/O interface," in *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, (Washington, DC, USA), p. 39, IEEE Computer Society, 2003.

[94] "HDF5 home page." (http://hdf.ncsa.uiuc.edu/HDF5/).

[95] E. Smirni, R. A. Aydt, A. A. Chen, and D. A. Reed, "I/O Requirements of Scientific Applications: An Evolutionary View," in *HPDC '96: Proceedings of the High Performance Distributed Computing (HPDC '96)*, (Washington, DC, USA), p. 49, IEEE Computer Society, 1996.

[96] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, pp. 1075–1089, 1996.

[97] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Characterization of a suite of Input/Output intensive applications," in *Proceedings of Supercomputing'95*, 1995.

[98] H. Simitci and D. A. Reed, "A comparison of logical and physical parallel I/O patterns," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 12, pp. 364–380, Fall 1998.

[99] E. Smirni and D. A. Reed, "Workload characterization of input/output intensive parallel applications," in *Proceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, vol. 1245, pp. 169–180, Springer-Verlag, 1997.

[100] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, and M. Z. et al, "FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes," in *Astrophysical Journal Suppliment*, vol. 131, pp. 273–334, nov 2000.

[101] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the SUN network filesystem," 1985. In Proceedings of the 1985 USENIX Conference. USENIX.

[102] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster, "Secure, efficient data transport and replica management for high-performance data-intensive computing," in *MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, (Washington, DC, USA), p. 13, IEEE Computer Society, 2001.

[103] V. Springel, "The cosmological simulation code GADGET-2," *Monthly Notices of the Royal Astronomical Society*, vol. 364, p. 1105, 2005.

[104] Barcelona Supercomputing Center, "Marenostrum Galaxy Formation Simulation." (http://astro.ft.uam.es/marenostrum/galaxy/index.html).

[105] Michael Zingale, "FLASH I/O Benchmark Routine – Parallel HDF 5," 2002. http://flash.uchicago.edu/.

[106] HLRS, "NEC cacau." (http://www.hlrs.de/hw-access/platforms/cacau/).

[107] L. M. S. García, F. Isaila, F. García, J. Carretero, R. Rabenseifner, and P. Adamidis, "A new I/O architecture for improving the performance in Large Scale Clusters," in *ICCSA 2006*, vol. 3984 of *LNCS*, pp. 108–117, Springer, 2006.

[108] FUSE, "Filesystem in Userspace." (http://fuse.sourceforge.net).

[109] A. Calderón, F. García, J. Carretero, J. M. Pérez, and J. Fernández, "An implementation of MPI-IO on expand: A parallel file system based on nfs servers," in *Proceedings of the 9th European PVM/MPI Users'Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, (London, UK), pp. 306–313, Springer-Verlag, 2002.

[110] "Extensible markup language (xml) 1.0." (www.w3.org/TR/REC-xml/).

[111] L. M. S. García, J. M. Pérez, A. Calderón, F. G. Carballeira, and J. Carretero, "Arquitectura escalable para E/S de altas prestaciones en sistemas heterogéneos," 2004. XV Jornadas de Paralelismo.

[112] "The parallel virtual file system 2 (PVFS2)." (http://www.pvfs.org/pvfs2/).

[113] R. Card, T. Tsó, and S. Tweedie, "Design and implementation of the second extended filesystem," in *Proceedings of the First Dutch International Symposium on Linux*, 1994.

[114] J. Pérez, L. M. Sánchez, F. García, A. Calderón, and J. Carretero, "High performance Java Input/Output for heterogeneous distributed computing," *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, pp. 969–974, 27-30 June 2005.

[115] R. Thakur, W. Gropp, and E. Lusk, "On implementing MPI-IO portably and with high performance," in *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 23–32, 1999.

[116] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snir, B. Traversat, and P. Wong, "Overview of the MPI-IO parallel I/O interface," in *Proceedings of the IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems*, pp. 1–15, 1995.

[117] Message Passing Interface Forum, "MPI-2: Extensions to the message-passing interface," 1997. (http://www.mpi-forum.org).

[118] LLNL, "IOR bench." (http://www.llnl.gov/asci/purple/benchmarks/).

[119] R. Rabenseifner and A. E. Koniges, "Effective Communication and File-I/O Bandwidth Benchmarks," in *Proceedings of the 8th European PVM/MPI Users'Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, (London, UK), pp. 24–35, Springer-Verlag, 2001.

[120] CLUSTER Resources, "TORQUE Batch System." (ttp://www.clusterresources.com/).

[121] A. Varga, "The OMNET++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference*, pp. 319–324, SCS – European Publishing House, June 2001.

[122] Edward K. Lee and R. H. Katz, "An analytic performance model of disk arrays," in *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 98–109, 1993.

[123] R. Barve, E. Shriver, P. B. Gibbons, B. K. Hillyer, Y. Matias, and J. S. Vitter, "Modeling and optimizing I/O throughput of multiple disks on a bus," in *SIGMETRICS '99: Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 83–92, ACM, 1999.

[124] M. Uysal, G. Alvarez, and A. Merchant, "A modular, analytical throughput model for modern disk arrays," *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pp. 183–192, 2001.

[125] E. Varki, A. Merchant, J. Xu, and X. Qiu, "Issues and challenges in the performance analysis of real disk arrays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 6, pp. 559–574, 2004.

[126] D. Feng, H. Jiang, and Y. Zhu, "I/O Response Time in a Fault-Tolerant Parallel Virtual File System," in *NPC* (H. Jin, G. R. Gao, Z. Xu, and H. Chen, eds.), vol. 3222 of *Lecture Notes in Computer Science*, pp. 248–251, Springer, 2004.

[127] H. Simitci, *Adaptive Disk Striping For Parallel Input/Output*. B.S., Bilkent University, 1992. Thesis, University of Illinois at Urbana-Champaign, 2000.

[128] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. New York, NY, USA: Wiley-Interscience, May 1991. Winner of "1991 Best Advanced How-To Book, Systems" award from the Computer Press Association.

[129] Jlich Supercomputer Center (JSC), "JUGENE: Juelicher Blue Gene/P." (http://www.fz-juelich.de/).

[130] A. Calderón, F. García-Carballeira, L. M. Sánchez, J. D. García, and J. Fernandez, "Fault tolerant file models for parallel file systems: introducing distribution patterns for every file," *J. Supercomput.*, vol. 47, no. 3, pp. 312–334, 2009.

[131] F. García-Carballeira, J. Carretero, A. Calderón, J. D. García, and L. M. Sanchez, "A global and parallel file system for grids," *Future Gener. Comput. Syst.*, vol. 23, no. 1, pp. 116–122, 2007.

[132] L. M. S. García, "Evaluation of the Expand parallel file system on a novel cluster-wide I/O architecture," in *Science and Supercomputing in Europe, 2005 Annual Report* (P. Alberigo, G. Erbacci, and F. Garofalo, eds.), pp. 281–285, Italy: CINECA, 2006. ISBN 88-86037-17-1.

[133] L. M. Sánchez, F. Isaila, A. Calderón, D. E. Singh, and J. D. García, "Improving the performance of cluster applications through I/O proxy architecture," in *CLUSTER*, 2006.

[134] L. M. S. García, B. B. Guerra, A. Calderón, F. García-Carballeira, and J. Carretero, "Descripción de una nueva arquitectura de E/S para grandes clusters," 2008. XIX Jornadas de Paralelismo.