UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Sistema de archivos paralelo para computación de alto desempeño

Informe de Proyecto de Grado presentado por

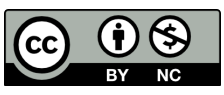
Santiago Freire López

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de la Facultad de Ingeniería de la Universidad de la República

Supervisor

Sergio Nesmachnow

Montevideo, 31 de julio de 2026



INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Prof. Jorge Merlino

Prof. Federico Rodríguez

Prof. Pablo Romero

Tutor: Prof. Sergio Nesmachnow

Dedicado a mis abuelos

Agradecimientos

A mi familia y amigos, por su constante apoyo y acompañamiento.

A mi tutor de tesis, por la guía e ideación del proyecto.

A mis compañeros del InCo, especialmente a quienes me ayudaron cargando servidores, reiniciando máquinas o dando apoyo.

A mis compañeros de ClusterUY.

A Leonardo Alberro y Eduardo Grampín, por prestarme dos servidores para utilizar como OSS.

A Ernesto Dufrechou e Ignacio Sastre, por prestarme los equipos para utilizar como clientes.

A Ángel Retali, por brindarme la aplicación de procesamiento de datos para DeepONet.

A Ángel Rosa, por brindarme la aplicación de procesamiento de datos de transporte público.

A Maximiliano González y Luis Bentancor, por brindarme la aplicación de calibración del modelo DayCent.

A los desarrolladores y mantenedores de GNU Screen.

Resumen

Muchas aplicaciones de computación de alto desempeño (HPC) imponen una alta carga sobre el subsistema de almacenamiento de sus entornos de ejecución y en consecuencia el almacenamiento frecuentemente se convierte en el principal factor limitante del rendimiento. Los sistemas de archivos paralelos permiten separar cargas de datos y metadatos, distribuir los datos entre múltiples servidores de almacenamiento y atender accesos concurrentes de múltiples clientes en simultáneo, para así superar las limitaciones de los sistemas de archivos centralizados o exportados desde un único servidor.

En este proyecto se estudiaron sistemas de archivos paralelos para entornos HPC y se realizó el despliegue y la evaluación experimental de uno de ellos sobre una infraestructura de hardware heterogéneo similar a la presente en el Centro Nacional de Supercomputación de Uruguay. A partir de una evaluación tecnológica de cuatro sistemas de archivos representativos (pNFS, CephFS, Lustre e IBM Storage Scale), se seleccionó Lustre por su adopción en centros de supercómputo, su carácter de código abierto y su soporte a distintos sistemas de archivos de base. El despliegue se realizó sobre cinco servidores y tres clientes interconectados por una red de 10 Gb/s y se evaluaron los dos sistemas de archivos de base soportados por Lustre, ldiskfs y ZFS (con y sin compresión).

La evaluación combinó tres suites de benchmark (fio, IOR y MDTest), que midieron el rendimiento de datos y metadatos bajo patrones controlados, con tres aplicaciones reales. Dos aplicaciones fueron intensivas en entrada/salida (E/S): la preparación de un conjunto de datos para el entrenamiento de un modelo subrogado DeepONet y el procesamiento de registros de viajes del transporte público de Montevideo. La tercera aplicación, intensiva en cómputo, fue la calibración paramétrica del modelo agronómico DayCent. El comportamiento de entrada y salida de las aplicaciones intensivas en E/S se caracterizó mediante rastreo dinámico con la herramienta bpftrace, se comparó el rendimiento del sistema de archivos paralelo con el del acceso vía Network File

System (NFS) y, para separar el efecto del paralelismo del efecto de la heterogeneidad del hardware, se introdujo una ponderación de los tiempos de ejecución según el rendimiento relativo de los objetivos de almacenamiento de objetos (OST) utilizados.

Los resultados mostraron que el número de OST fue el parámetro más influyente en el rendimiento de las aplicaciones intensivas en E/S, con la mayor parte de la mejora concentrada al escalar de uno a tres OST; en la aplicación dominada por el cómputo, en cambio, ni el número de OST ni la configuración de Lustre tuvieron un efecto apreciable. El sistema de archivos de base ZFS con compresión LZ4 superó a ldiskfs en las aplicaciones intensivas en E/S y resultó en tiempos de ejecución más estables, aunque el beneficio de la compresión dependió de la compresibilidad de los datos. El sistema de archivos ldiskfs mostró mayor rendimiento que ZFS en las operaciones de metadatos. La comparación con NFS resultó dependiente del perfil de cada aplicación: el sistema de archivos paralelo superó al acceso vía NFS solo en parte de los escenarios evaluados. Además, el reparto equitativo de los datos entre OST de rendimiento distinto provocó que el OST más lento limitara el rendimiento del conjunto, lo que sugiere estrategias de distribución ponderada como línea de trabajo futuro.

Palabras clave: computación de alto desempeño, sistemas de archivos paralelos, Lustre, evaluación de rendimiento.

Tabla de contenidos

Capítulo 1

Introducción

Las aplicaciones de computación de alto desempeño (HPC) resuelven problemas que, por su escala, no pueden abordarse con computadoras tradicionales. Las aplicaciones HPC se ejecutan sobre supercomputadoras, conjuntos de nodos de cómputo interconectados por redes de alta velocidad que colaboran para resolver en conjunto un problema computacional de alta complejidad. Muchas de las aplicaciones HPC realizan lecturas y escrituras intensivas de datos, por lo que el subsistema de almacenamiento es, en muchos casos, el principal factor limitante para el rendimiento.

Los sistemas de archivos tradicionales, centralizados o exportados desde un único servidor, no escalan adecuadamente ante el acceso concurrente de un gran número de procesos. En el Centro Nacional de Supercomputación de Uruguay (Nesmachnow y Iturriaga, 2019), el protocolo Network File System (NFS) se utiliza para servir los directorios home de los usuarios, pero dicho protocolo no está diseñado para sostener la alta concurrencia paralela y fuerte carga de datos y metadatos de las cargas de computación de alto desempeño. Los sistemas de archivos paralelos separan la carga de datos de la carga de metadatos, distribuyen los datos de los archivos entre múltiples servidores de almacenamiento, permiten el acceso concurrente de múltiples clientes a los mismos archivos y dan una mayor escalabilidad al sistema de almacenamiento.

El objetivo de este proyecto de grado fue desplegar y evaluar experimentalmente un sistema de archivos paralelo sobre una infraestructura de hardware heterogéneo similar a la existente en el Centro Nacional de Supercomputación de Uruguay y analizar su rendimiento bajo distintas configuraciones y cargas de trabajo. El trabajo aborda la selección de un sistema de archivos paralelo

a partir de una evaluación tecnológica, su despliegue sobre la infraestructura disponible, la comparación de los dos sistemas de archivos de base soportados y la evaluación del rendimiento de datos y metadatos mediante benchmarks y tres aplicaciones reales de perfiles diversos: dos aplicaciones intensivas en entrada/salida (E/S) y una aplicación intensiva en cómputo. El rendimiento del sistema de archivos paralelo se contrasta además con el rendimiento del acceso a dispositivos de almacenamiento a través de NFS.

Las principales contribuciones del trabajo son: (i) el despliegue de un sistema de archivos paralelo Lustre sobre hardware heterogéneo de prestaciones limitadas; (ii) una metodología de evaluación que combina benchmarks sintéticos, la caracterización de entrada y salida de aplicaciones reales mediante rastreo dinámico y una ponderación de los tiempos de ejecución por la heterogeneidad del hardware de almacenamiento, que permite distinguir el efecto del paralelismo de la disparidad de rendimiento entre los objetivos de almacenamiento de objetos; (iii) un análisis comparativo del comportamiento de los sistemas de archivos de base (ldiskfs y ZFS) bajo distintas configuraciones de Lustre y frente al acceso vía NFS; y (iv) la identificación de cuellos de botella y anomalías de rendimiento.

El documento se organiza de la siguiente manera. El Capítulo 2 presenta el marco teórico necesario para comprender el trabajo: computación de alto desempeño, sistemas de almacenamiento, sistemas de archivos y sistemas de archivos distribuidos y paralelos. El Capítulo 3 realiza una evaluación tecnológica de cuatro sistemas de archivos paralelos y analiza trabajos relacionados de despliegue y evaluación de rendimiento. El Capítulo 4 describe la solución propuesta: el sistema de archivos elegido, los sistemas de archivos de base, la infraestructura de despliegue, el rendimiento teórico máximo del almacenamiento y la configuración de los servidores. El Capítulo 5 detalla el diseño experimental: las herramientas de benchmark, las aplicaciones utilizadas, la metodología de caracterización de entrada y salida y de monitoreo de recursos y la ponderación por heterogeneidad de los OST empleada para calcular la aceleración y la eficiencia. El Capítulo 6 presenta y analiza los resultados de la evaluación experimental con benchmarks. El Capítulo 7 presenta y analiza los resultados de la evaluación experimental con las aplicaciones. Finalmente, el Capítulo 8 presenta las conclusiones y el trabajo futuro.

Capítulo 2

Marco teórico

En este capítulo se presentan los conceptos teóricos de las temáticas que trata el proyecto.

2.1. Computación de alto desempeño

La computación de alto desempeño es una rama de la computación que se centra en optimizar el rendimiento y reducir el tiempo requerido para resolver problemas computacionales mediante la combinación de múltiples disciplinas interrelacionadas. El propósito de la computación de alto desempeño es brindar respuestas a problemas que no pueden ser resueltos únicamente mediante el empirismo, la teoría o el uso de computadoras tradicionales (Sterling et al., 2017).

Una aplicación HPC comprende un problema y el conjunto de instrucciones o código que describe cómo realizar su procesamiento computacional. Para ejecutar el código de las aplicaciones HPC se utilizan máquinas digitales llamadas supercomputadoras, una combinación de múltiples nodos de alto poder de procesamiento interconectados entre sí mediante redes de alta velocidad, capaces de colaborar para resolver en conjunto un problema computacional. Un nodo de cómputo HPC es una computadora convencional con una alta capacidad de cómputo. Una supercomputadora se distingue de una computadora tradicional por la organización, la interconectividad, la escala de sus recursos de cómputo y el software de base que permite administrar el sistema a gran escala.

Para resolver los problemas computacionales, la computación de alto desempeño utiliza paralelismo físico, replicación de recursos de cómputo a través de los nodos de la computadora y paralelismo lógico, tareas ejecutadas concurrentemente. Los distintos procesos o tareas que ejecutan en la supercomputadora se comunican entre sí mediante pasaje de mensajes o memoria compartida. La organización paralela, la forma de coordinación entre los subsistemas para resolver un problema común y el soporte nativo para el paralelismo del software de base y los modelos de programación también diferencian la supercomputadora de las computadoras convencionales.

2.2. Sistemas de almacenamiento

En esta sección se presenta la definición de sistema de almacenamiento, los dispositivos físicos que componen el sistema y los arreglos de discos, que combinan varios dispositivos de almacenamiento para mejorar el rendimiento y la confiabilidad.

2.2.1. Definición

Un sistema de almacenamiento es el conjunto de tecnologías y componentes utilizados para persistir información aun cuando no haya alimentación eléctrica en los componentes (Arpaci-Dusseau y Arpaci-Dusseau, 2018), a diferencia de, por ejemplo, la memoria RAM, cuyos datos se pierden ante la pérdida de energía eléctrica.

Los sistemas de almacenamiento se componen de uno o más dispositivos de almacenamiento físicos que persisten datos. Los dispositivos de almacenamiento se pueden configurar de manera conjunta para formar un único dispositivo de almacenamiento lógico o virtual.

Las transferencias de datos desde y hacia los dispositivos físicos que conforman un sistema de almacenamiento se realizan en unidades de un tamaño fijo en bytes denominadas bloques. El tamaño de un bloque es usualmente de 512 o 4096 bytes pero puede tomar otros valores.

2.2.2. Dispositivos de almacenamiento

Un dispositivo de almacenamiento es el componente físico encargado de persistir datos dentro de un sistema de almacenamiento. En esta sección se describe la forma en que los dispositivos de almacenamiento se conectan y se comunican con el sistema operativo y se presentan dos tipos ampliamente utilizados, los discos magnéticos y los discos de estado sólido.

2.2.2.1. Conexión y comunicación

Los dispositivos de almacenamiento masivo típicamente se conectan mediante un bus de entrada y salida (Arpaci-Dusseau y Arpaci-Dusseau, 2018), ya sea de tipo general (por ejemplo, PCI) o periférico (SATA, SCSI, USB, etc.). La interacción con el dispositivo de almacenamiento masivo se realiza a través de comunicaciones con la controladora del dispositivo, un circuito que actúa como interfaz entre el sistema operativo y el hardware, encargado de controlar el dispositivo físicamente. El sistema operativo se comunica con la controladora a través de un programa denominado controlador de dispositivo, que conoce el formato de los mensajes y el protocolo específico para la comunicación con la controladora (Silberschatz et al., 2018).

Dos tipos de dispositivos utilizados en sistemas de almacenamiento son los discos magnéticos y los discos de estado sólido. Sus principales conceptos se presentan en 2.2.2.2 y 2.2.2.3.

2.2.2.2. Discos magnéticos

Los discos magnéticos (HDD), comúnmente llamados discos duros, han sido los principales medios de almacenamiento por décadas y parte del desarrollo de varios sistemas de archivos se ha basado en su comportamiento físico. Sus componentes principales son los platos, discos circulares planos donde los bits se almacenan de forma magnética, los cabezales de lectura y escritura que se desplazan sobre la superficie de cada plato y el brazo, que mueve todos los cabezales en conjunto. Cada plato está dividido lógicamente en pistas circulares, que a su vez se dividen en sectores. El conjunto de pistas que se sitúan en los cabezales de lectura-escritura en una determinada posición del brazo se denomina cilindro. El motor del disco gira todos los platos a la vez y a la misma velocidad (Silberschatz et al., 2018; Arpaci-Dusseau y Arpaci-Dusseau, 2018).

Para acceder a un bloque en el disco, se debe posicionar el cabezal sobre el sector correspondiente. El tiempo que tarda el cabezal en posicionarse en el sector consta de dos partes: el tiempo de búsqueda, tiempo requerido para mover el brazo del disco al cilindro deseado y la latencia rotacional, tiempo requerido para girar el plato hasta el sector correspondiente en la pista.

2.2.2.3. Discos de estado sólido

Los discos de estado sólido (SSD) son dispositivos de almacenamiento sin partes móviles basados en memoria flash NAND. Al eliminar los tiempos de posicionamiento y búsqueda de los discos magnéticos, presentan menores tiempos de acceso y en general un mayor rendimiento de lectura y escritura, especialmente en accesos aleatorios (Silberschatz et al., 2018). Su principal desventaja es el costo por unidad de capacidad, mayor al de los discos magnéticos.

2.2.3. Arreglos de discos

Un arreglo de discos es una configuración que combina múltiples dispositivos de almacenamiento físicos para que operen como una única unidad lógica, con el objetivo de mejorar el rendimiento, la confiabilidad o la capacidad respecto al uso de un disco individual. En esta sección se presenta la definición de los arreglos redundantes de discos independientes (RAID), sus niveles y las formas en que pueden implementarse.

2.2.3.1. Definición

Un arreglo redundante de discos independientes es un conjunto de técnicas de organización de discos que buscan mejorar el rendimiento y la confiabilidad del almacenamiento al utilizar múltiples discos a la vez. La confiabilidad se mejora con la introducción de redundancia, almacenar información adicional que permite reconstruir los datos en caso de falla de discos. El rendimiento se mejora mediante la técnica de striping, que consiste en distribuir los datos entre múltiples discos para poder realizar lecturas y escrituras en paralelo (Silberschatz et al., 2018).

2.2.3.2. Niveles de RAID

Los esquemas RAID se clasifican en niveles, cada uno con distintas características de redundancia, rendimiento y capacidad útil. Los distintos niveles se pueden combinar entre sí (Silberschatz et al., 2018).

RAID 0 organiza los discos en un arreglo con striping de bloques sin ningún mecanismo de redundancia. Ofrece el mayor rendimiento de todos los niveles al distribuir las operaciones entre todos los discos del arreglo, pero no tolera la falla de ningún disco.

RAID 1 implementa espejado de discos. Cada disco tiene una copia exacta en otro disco del arreglo y toda escritura se realiza en ambos discos simultáneamente. Ante la falla de un disco, los datos pueden leerse del disco espejo. La distribución de las lecturas entre ambas copias puede aumentar la tasa de lectura respecto a un disco único. Su principal desventaja es el costo en capacidad, ya que la mitad del espacio total del arreglo se destina a las copias.

En RAID 5 y RAID 6, se denomina raya al conjunto de bloques que ocupan la misma posición relativa en cada disco del arreglo y bloque de paridad al bloque que contiene información redundante calculada a partir de los bloques de datos de la raya.

RAID 5 distribuye los datos y bloques de paridad entre todos los discos del arreglo y tolera la falla de hasta un disco. Para cada raya, uno de los discos almacena el bloque de paridad y los restantes almacenan datos, rotando el disco que almacena la paridad entre rayas para evitar que un único disco se convierta en cuello de botella. Ante la falla de un disco, los bloques perdidos pueden reconstruirse a partir de los datos y los bloques de paridad de los discos restantes, aunque con una penalización de rendimiento, ya que cada lectura de un bloque en el disco perdido requiere leer todos los demás discos para reconstruir el bloque perdido. Las escrituras que no cubren una raya completa requieren leer los bloques originales y el bloque de paridad, recalcular la nueva paridad mediante una operación XOR y escribir los bloques actualizados, procedimiento que se denomina ciclo de lectura-modificación-escritura.

RAID 6 es similar a RAID 5, pero almacena dos bloques de paridad por raya en lugar de uno. Los bloques de paridad son calculados con códigos correctores de errores basados en operaciones matemáticas en campos de Galois. RAID 6 tolera la falla simultánea de hasta dos discos.

RAID 1+0 combina espejado y striping: los discos se organizan en pares espejados y luego los pares se distribuyen mediante striping. Ante la falla de un disco, el arreglo continúa accesible y operativo siempre que el otro disco del par espejado del disco fallado continúe funcionando.

2.2.3.3. RAID por hardware y por software

La funcionalidad RAID puede implementarse en distintas capas del sistema. En la implementación por software, el sistema operativo implementa RAID mediante software de gestión de volúmenes en el núcleo del sistema operativo sin requerir hardware especializado. En la implementación por hardware, una controladora de bus de host gestiona los discos conectados e implementa RAID directamente en el hardware sin intervención del sistema operativo. El sistema operativo crea sus sistemas de archivos sin conocer los detalles del arreglo subyacente (Silberschatz et al., 2018).

2.3. Sistemas de archivos

La capa lógica que permite acceder estructuradamente a los datos almacenados en los dispositivos físicos es el sistema de archivos. En esta sección se presenta su definición y estructura y luego las dos abstracciones que utiliza: los archivos y los directorios.

2.3.1. Definición y estructura

Los sistemas de archivos proporcionan un mecanismo para almacenar, organizar y acceder a datos y programas sobre dispositivos de almacenamiento persistente. Un sistema de archivos puede describirse en términos generales como una colección de archivos que contienen los datos almacenados y una estructura de directorios que organiza los archivos y mantiene la información necesaria para localizar cada archivo (Silberschatz et al., 2018).

Cada archivo tiene asociado un bloque de control de archivo que almacena metadatos como permisos, propietario, tamaño, fechas asociadas y la información necesaria para localizar sus datos. Según el sistema de archivos, esta información puede incluir referencias directas a bloques de datos o punteros a estructuras que permiten localizar los bloques de datos. El bloque de control de archivo vincula la representación lógica del archivo con la disposición

de sus datos en el sistema de almacenamiento físico. Los directorios también pueden representarse mediante estructuras de control equivalentes a las de los archivos.

Además de los bloques de control de archivo, el sistema de archivos mantiene otras estructuras persistentes, por ejemplo el bloque de control de volumen. El bloque de control de volumen contiene información global del volumen como el número de bloques, el tamaño de bloque y la disponibilidad de espacio libre. Si el volumen puede utilizarse para iniciar el sistema, también puede incluir un bloque de arranque con la información necesaria para comenzar el proceso de carga del sistema operativo.

2.3.2. Archivos y directorios

Los sistemas de archivos organizan y administran la información almacenada en los dispositivos físicos mediante abstracciones que separan la representación lógica de los datos de la disposición física en el hardware. Entre las principales abstracciones existentes se encuentran los archivos y los directorios.

2.3.2.1. Archivos

Los archivos son una representación lógica de los datos almacenados en el sistema de archivos. El sistema operativo abstrae las propiedades físicas de los dispositivos de almacenamiento para definir una unidad lógica de almacenamiento llamada archivo, la unidad mínima de almacenamiento accesible a los usuarios del sistema de archivos (Silberschatz et al., 2018).

Los archivos poseen atributos, como nombre, tipo, ubicación, tamaño y permisos. Los sistemas operativos interactúan con el sistema de archivos y proveen una interfaz para crear, abrir, escribir, leer y realizar otras acciones sobre los archivos.

2.3.2.2. Directorios

Un directorio, al igual que un archivo, es una abstracción. Su contenido es una colección de pares clave-valor: la clave es el nombre definido por el usuario del archivo o directorio contenido en el directorio y el valor es una referencia al bloque de control de archivo. Si se colocan directorios dentro de otros directorios se genera un árbol de directorios, que tiene un solo directorio en la cima llamado directorio raíz (Arpaci-Dusseau y Arpaci-Dusseau, 2018).

2.4. Sistemas de archivos distribuidos

Un sistema de archivos distribuido permite almacenar y acceder a archivos cuyos datos residen en múltiples nodos interconectados. En esta sección se presenta la definición, los componentes, las arquitecturas en que pueden organizarse, los mecanismos de acceso remoto a los archivos y los mecanismos de coherencia de caché en los sistemas de archivos distribuidos.

2.4.1. Definición

Un sistema de archivos distribuido es un sistema de archivos cuyos clientes, servidores y dispositivos de almacenamiento se encuentran distribuidos entre las computadoras de un sistema distribuido (un conjunto de procesadores que no comparten memoria o reloj y se comunican a través de una red) (Silberschatz et al., 2018). Los bloques de los archivos se distribuyen a lo largo de nodos de almacenamiento, se proveen mecanismos de control de concurrencia para soportar lecturas y escrituras concurrentes sobre los mismos archivos y se pueden proveer mecanismos de tolerancia a fallos para soportar pérdidas o fallos de nodos de almacenamiento (Pan et al., 2023).

A diferencia de un sistema de archivos centralizado, los sistemas de archivos distribuidos buscan almacenar y manejar datos a través de múltiples nodos interconectados. El sistema de archivos es servido a través de una red de computadoras y, en lugar de un único repositorio de datos centralizado, el sistema cuenta con múltiples dispositivos de almacenamiento independientes. El sistema de archivos distribuido debe orquestar la comunicación y la coordinación entre los nodos de almacenamiento para lograr el almacenamiento distribuido y el acceso y administración de archivos y directorios (Silberschatz et al., 2018).

Idealmente, un sistema de archivos distribuido debería presentarse ante sus clientes como un sistema de archivos centralizado y convencional. La interfaz que ofrece al cliente no debería diferenciar entre archivos locales y remotos y es responsabilidad del propio sistema de archivos localizar los archivos y organizar el transporte de los datos. Por lo general, el usuario no discierne que el sistema de archivos que utiliza es distribuido, ya que el sistema de archivos abstrae la complejidad y se presenta lógicamente de la misma manera que un sistema de archivos centralizado. A diferencia de un sistema de archivos convencional, donde el tiempo de atención a una solicitud se compone del

acceso al almacenamiento y un pequeño costo de procesamiento, en un sistema de archivos distribuido el acceso remoto añade el costo de entregar la solicitud al servidor, transmitir la respuesta de vuelta al cliente y ejecutar el software del protocolo de comunicación en ambos sentidos.

2.4.2. Componentes de un sistema de archivos distribuido

Un sistema de archivos distribuido se compone de distintos tipos de nodos (de gestión, de almacenamiento de metadatos, de datos y clientes), cada uno con responsabilidades específicas que se describen a continuación.

Nodos de gestión. Los nodos de gestión son los nodos centrales del sistema de archivos distribuido, responsables de mantener en memoria principal la estructura lógica y los metadatos del sistema de archivos. Los clientes envían solicitudes de operación sobre archivos a los nodos de gestión, que luego dirigirán la solicitud al nodo de datos apropiado (Pan et al., 2023).

Nodos de almacenamiento de metadatos. En los nodos de almacenamiento de metadatos se persisten los metadatos del sistema de archivos distribuido. Los metadatos del sistema de archivos distribuido, que también son almacenados en la memoria principal de los nodos de gestión, deben ser persistidos para tolerar fallos producidos por reinicios o fallas de los nodos de gestión.

Nodos de datos. Los nodos de datos almacenan los bloques de datos de los archivos y directorios y gestionan las solicitudes de lectura y escritura de bloques por parte de los clientes. Periódicamente, los nodos de datos reportan información sobre sus bloques a los nodos de gestión.

Clientes. Los clientes son nodos que utilizan la interfaz provista por el sistema de archivos distribuido. Inician solicitudes de acceso a datos como creación, eliminación, búsqueda o ubicación a los nodos de gestión, quienes a cambio devuelven el resultado de la operación junto a metadatos relacionados. Por ejemplo, la operación de ubicación devuelve la ubicación en los nodos de datos de los bloques de datos del archivo o directorio buscado. Con la información de ubicación obtenida, el cliente o el nodo de gestión, según del caso, se comunica con los nodos de datos correspondientes para realizar la lectura o escritura de los bloques de datos.

2.4.3. Arquitectura de sistemas de archivos distribuidos

Los sistemas de archivos distribuidos pueden organizarse en alguna de las cuatro arquitecturas (centralizada, distribuida, jerárquica y entre pares) (Pan et al., 2023) que se describen a continuación.

Arquitectura centralizada. En un sistema de archivos distribuido con una arquitectura centralizada se tiene un solo nodo de gestión central, responsable de manejar los metadatos del sistema de archivos distribuido y controlar las solicitudes de acceso a datos. El nodo central es responsable de dirigir las solicitudes a los nodos de datos correspondientes. Es fácil de implementar y mantener, pero el nodo central es un único punto de fallo y puede tornarse un cuello de botella.

Arquitectura distribuida. En una arquitectura distribuida, los metadatos y la gestión del sistema de archivos distribuido se distribuyen a través de múltiples nodos de gestión, cada uno capaz de procesar solicitudes de acceso a datos de manera independiente. La arquitectura distribuida elimina el cuello de botella y el único punto de fallo de la arquitectura centralizada y ofrece mayor escalabilidad, tolerancia a fallos y rendimiento a costa de un aumento en complejidad. Para coordinar y comunicar los nodos de gestión se utilizan mecanismos como tablas de hash distribuidas o algoritmos de consenso.

Arquitectura jerárquica. Una arquitectura jerárquica organiza el sistema de archivos en distintas capas, cada capa con distintas funciones y responsabilidades. Usualmente las capas de un sistema de archivos distribuido con arquitectura jerárquica son la capa de almacenamiento, la capa de nombres y la capa de acceso. La capa de almacenamiento se encarga de manejar y almacenar los datos del sistema de archivos distribuido, la capa de nombres se encarga de almacenar la estructura lógica y los metadatos del sistema de archivos distribuido y la capa de acceso procesa solicitudes de acceso a archivos por parte de los clientes y las redirige a los nodos de datos correspondientes.

Arquitectura entre pares. En una arquitectura entre pares, todos los nodos del sistema de archivos distribuido comparten las responsabilidades de gestión de metadatos, estructura lógica y almacenamiento de datos y cada nodo actúa como productor y consumidor de datos. La arquitectura entre pares ofrece alta descentralización y tolerancia a fallos, pero es más compleja de mantener y coordinar por la participación de la totalidad de los nodos en la coordinación del sistema de archivos distribuido.

2.4.4. Acceso a archivos remotos

Al acceder a un archivo en un sistema de archivos distribuido, una vez localizados los servidores que almacenan el archivo, se debe realizar la transferencia de los datos desde los nodos de datos hacia el cliente. Existen dos mecanismos para realizar las transferencias de datos: el servicio remoto y el uso de caché (Silberschatz et al., 2018). En el servicio remoto, las solicitudes de acceso se envían al nodo de datos, que realiza los accesos y envía los resultados al cliente. Una forma común de implementar el servicio remoto es mediante llamadas a procedimientos remotos (RPC). El segundo mecanismo es el uso de caché. Si los datos necesarios para satisfacer una solicitud de acceso no están en la caché del cliente, se traen los datos desde el o los servidores y los accesos se realizan sobre la copia en caché. A diferencia de la caché en sistemas de archivos convencionales, donde se busca reducir solamente la entrada y salida a disco, en los sistemas de archivos distribuidos se busca reducir el tráfico de red y la entrada y salida a disco. La granularidad de los datos en caché puede ir de bloques individuales hasta archivos completos. En la práctica, muchas implementaciones incorporan mecanismos de servicio remoto y mecanismos de caché.

2.4.5. Coherencia de caché

Cuando una copia en caché es modificada, los cambios deben reflejarse en los datos que residen en el o los nodos de datos. El problema de mantener las copias en caché consistentes con los nodos de datos se denomina problema de coherencia de caché (Silberschatz et al., 2018). Las principales políticas de coherencia de caché son write-through, write-back y write-on-close y se explican a continuación.

Write-through. Los datos se escriben en los servidores cuando se modifican en la caché. Es más confiable ante fallas en los clientes, pero penaliza el rendimiento ya que cada acceso de escritura debe esperar a que la información llegue a los servidores.

Write-back. Las modificaciones se escriben en la caché y se persisten en los servidores en un momento posterior. Acelera los accesos de escritura y permite que datos sobrescritos varias veces solo requieran una escritura, pero los datos no escritos se pierden si el cliente cae.

Write-on-close. Los datos se escriben en los servidores cuando el archivo se cierra. Es beneficiosa para archivos modificados con frecuencia y abiertos por largos períodos.

Para verificar la validez de los datos en caché existen dos enfoques: iniciado por el cliente e iniciado por los servidores. En el enfoque iniciado por el cliente, el cliente contacta a los servidores y verifica si los datos locales son consistentes con los datos persistidos. En el enfoque iniciado por los servidores, los servidores registran qué archivos almacenan en caché los clientes y reaccionan cuando detectan una potencial inconsistencia, por ejemplo cuando dos clientes almacenan en caché un mismo archivo de manera conflictiva.

2.5. Sistemas de archivos paralelos

Un sistema de archivos paralelo es un tipo especial de sistema de archivos distribuido con foco en el acceso de lectura o escritura concurrente por parte de múltiples clientes a los mismos archivos. Los sistemas de archivos únicamente distribuidos, a diferencia de los sistemas de archivos paralelos, se centran en soportar la estructura lógica compartida y tienen mejor rendimiento cuando los distintos clientes operan en conjuntos de archivos disjuntos entre sí. Los sistemas de archivos paralelos son más apropiados para aplicaciones de computación de alto desempeño, que suelen generar accesos concurrentes desde múltiples procesos sobre un archivo compartido o sobre conjuntos de archivos relacionados (Sterling et al., 2017).

En los sistemas de archivos paralelos se aplican técnicas para maximizar el rendimiento, por ejemplo striping de los bloques de los archivos entre los distintos dispositivos de almacenamiento físicos. Si la tasa de transferencia de un disco es de N bloques por segundo y el total de bloques de un archivo es B, si todos los bloques de un archivo están en un solo dispositivo de almacenamiento, el tiempo de transferencia de la totalidad de los datos es de B/N segundos (Sterling et al., 2017). En cambio, si hay M dispositivos de almacenamiento, cada uno con tasa de transferencia de N bloques por segundo y el archivo está distribuido equitativamente mediante striping entre los M dispositivos, cada dispositivo recibe aproximadamente B/M bloques. Por lo tanto, en el mejor caso el tiempo de transferencia es de $T = B/(MN)$ segundos. $B/(MN) < B/N \iff 1/M < 1$, que se cumple siempre que $M > 1$, por lo que el tiempo de transferencia al aplicar striping, en el mejor caso, se reduce

en un factor de M. El tamaño de stripe debe elegirse adecuadamente para no generar penalizaciones excesivas de búsquedas en disco y procesamiento en las controladoras de los dispositivos (cuando el tamaño del stripe es muy pequeño) o eliminar la ventaja de rendimiento del striping con la elección de un tamaño muy grande. Los archivos, además de ser divididos en stripes en los dispositivos de almacenamiento de un nodo, pueden dividirse en stripes entre múltiples nodos de almacenamiento para aumentar el rendimiento (Sterling et al., 2017).

Un sistema de archivos paralelo, al igual que en los sistemas de archivos distribuidos, debe abstraer la complejidad (arquitectura, características de los dispositivos físicos, tolerancia a fallos, ajustes internos del sistema de archivos) y presentarse en los clientes como un sistema de archivos regular. Usuarios que deseen optimizar el rendimiento de entrada y salida pueden modificar parámetros específicos del sistema de archivos. El sistema de archivos puede presentarse con una interfaz estándar como POSIX, que permite que aplicaciones existentes accedan al sistema de archivos paralelo sin modificaciones.

Los datos y metadatos (contenido de archivos, tamaños, permisos y demás atributos) deben mantener consistencia entre los distintos accesos concurrentes. Realizar solo lecturas concurrentes no genera problemas, ya que los datos no se modifican. Agregar procesos escritores sí puede generar problemas, ya que pueden interferir en la propagación y replicación de datos en los nodos que acceden a los mismos datos. Los sistemas de archivos paralelos, con el objetivo de dar mayor rendimiento de lectura y escritura, pueden relajar la atomicidad en los accesos: no garantizar que los datos leídos o escritos en una sola llamada no sean modificados por una precedente o subsecuente llamada que solape. Los algoritmos utilizados por el sistema de archivos paralelo deben escalar para soportar un gran número de clientes concurrentes, potencialmente la totalidad de nodos del sistema y además soportar el aumento o la adición de almacenamiento.

Capítulo 3

Evaluación tecnológica y análisis de trabajos relacionados

En este capítulo se presentan distintos sistemas de archivos paralelos existentes, se realiza un análisis comparativo de ellos y se plantea el análisis de trabajos relacionados.

3.1. Sistemas de archivos paralelos

En esta sección se presentan distintos sistemas de archivos paralelos y se explica, de manera general, el funcionamiento de cada uno.

3.1.1. Parallel Network File System

Network File System (NFS) es un protocolo de sistema de archivos distribuido que permite a múltiples clientes acceder de manera remota a archivos ubicados en un servidor remoto e interactúen con los archivos y directorios de la misma manera que si los archivos fueran locales. Al momento de realizar el proyecto, NFS se encuentra en su versión 4.2, fue desarrollado inicialmente por Sun Microsystems y se encuentra estandarizado por el IETF (Haynes y Noveck, 2015; Haynes, 2016).

Los objetivos de NFS son proporcionar interoperabilidad entre plataformas heterogéneas, asegurar la retrocompatibilidad con versiones anteriores del protocolo, permitir que los clientes interactúen con el sistema de archivos como si fuera local, ofrecer mecanismos de seguridad fuertes y dar un rendimiento aceptable.

NFS se basa en el modelo cliente-servidor; el servidor exporta uno o varios directorios del sistema de archivos que luego pueden ser montados por los clientes. Los clientes interactúan con los archivos remotos como si fueran locales mediante RPC con TCP como protocolo de transporte. En versiones anteriores a NFSv4 también se utilizaba UDP (Haynes y Noveck, 2015). En NFSv4, las llamadas se encapsulan en procedimientos denominados COMPOUND, que permiten agrupar múltiples operaciones lógicas (lectura, escritura, búsqueda, etc.) en una sola solicitud.

Los archivos y directorios se identifican mediante estructuras llamadas filehandles, generadas y manejadas por el servidor, sin que los clientes necesiten conocer su representación interna. Los clientes pueden enviarlas, recibirlas o almacenarlas para solicitudes posteriores, pero no deben interpretar su contenido. Los filehandles pueden ser persistentes o volátiles, según la capacidad del servidor de mantener la consistencia tras reinicios o cambios en el sistema de almacenamiento. En el cliente, las operaciones sobre archivos se realizan mediante uno o varios filehandles en lugar de rutas o nombres de archivo, con la excepción de la búsqueda de archivos y la obtención del filehandle del directorio raíz.

Para controlar la concurrencia se utilizan bloqueos por rango de bytes: un cliente bloquea un rango de bytes de un archivo para escritura. Mientras el cliente escribe en el rango, ningún otro cliente puede escribir dentro del rango hasta que el escritor libere el bloqueo. Para implementar los bloqueos el servidor otorga concesiones temporales de los archivos a los clientes que deben renovarse cada cierto tiempo.

NFS suele estar integrado por defecto en la mayoría de sistemas tipo Unix de uso habitual y es fácil de configurar y usar. Además, en redes de velocidad de 1 Gbps o superiores, para pocos clientes el rendimiento del sistema de archivos suele ser aceptable. NFS es libre y de código abierto.

A partir de NFSv4.1 se introdujo una característica opcional llamada parallel NFS (pNFS) que permite el acceso paralelo a los datos. El contenido de los archivos se puede distribuir entre múltiples servidores de almacenamiento (Noveck y Lever, 2020). En pNFS el procesamiento se divide en procesamiento de metadatos y procesamiento de datos: los metadatos son gestionados por uno o más servidores de metadatos, que actúan como servidores NFSv4.1 estándar y los datos de los archivos se almacenan en uno o más servidores de almacenamiento. Una vez que el servidor de metadatos proporciona al cliente la

información necesaria para localizar y acceder a los datos, el cliente puede comunicarse directamente con los servidores de almacenamiento sin comunicarse con el servidor de metadatos en cada operación de lectura o escritura. El estándar define tres modalidades de acceso al almacenamiento: por archivos, por bloques y por objetos. En la distribución por archivos, el cliente accede a los datos mediante el protocolo NFS y el contenido de los archivos puede distribuirse entre múltiples servidores de almacenamiento; en la distribución por bloques, el cliente accede directamente a dispositivos de bloques; y en la distribución por objetos, los datos se almacenan como objetos en un sistema de almacenamiento orientado a objetos.

3.1.2. CephFS

CephFS es un sistema de archivos distribuido y paralelo que tiene como objetivo ofrecer alta escalabilidad, rendimiento y confiabilidad en entornos donde se manejan altos volúmenes de datos y existe una gran cantidad de clientes (Ceph authors and contributors, 2026). La arquitectura de CephFS se basa en tres elementos principales: los clientes, un clúster de metadatos (MDS) y un clúster de almacenamiento de objetos (OSD). Los metadatos son gestionados por los MDS y el acceso a datos se realiza entre el cliente y los OSD, sin intermediación del MDS. Los clientes acceden a un sistema de archivos CephFS mediante una interfaz compatible con POSIX. Los datos y metadatos del sistema de archivos se almacenan en pools del almacén de objetos distribuido de Ceph (RADOS). El pool RADOS de los datos está separado del pool RADOS de los metadatos.

CephFS evita mantener una tabla central de ubicación para cada bloque de datos de los archivos y la reemplaza por un esquema de direccionamiento determinístico: los datos de cada archivo se dividen en stripes de tamaño fijo que se almacenan como objetos en RADOS, nombrados de manera predecible a partir del número de inodo del archivo y la posición del objeto dentro del archivo. A partir del nombre del objeto y los mapas del clúster obtenidos desde nodos monitores, el cliente puede calcular localmente, mediante el algoritmo Controlled Replication Under Scalable Hashing (CRUSH), los OSD que almacenan el objeto sin consultar una tabla central.

La asignación de objetos a OSD se realiza mediante CRUSH, que define una función pseudoaleatoria y descentralizada que distribuye los objetos considerando la topología física del clúster y los dominios de falla. Los dominios de falla son los niveles de la jerarquía física del clúster que CRUSH recorre para ubicar las réplicas de forma que no queden en componentes que puedan fallar de manera conjunta (Ceph authors and contributors, 2026; Weil et al., 2006).

El clúster de metadatos utiliza una técnica denominada particionado dinámico de subárboles, que consiste en dividir los metadatos del sistema de archivos entre varios servidores MDS en función de los accesos recientes (Weil et al., 2006). Si muchos clientes acceden a ciertas zonas del sistema de archivos, CephFS puede redistribuir o replicar los metadatos entre distintos MDS para balancear las cargas y evitar cuellos de botella, por lo que el rendimiento de la gestión de metadatos escala a medida que se agregan MDS al clúster (Ceph authors and contributors, 2026).

Los OSD se encargan de la replicación de objetos, la detección y recuperación ante fallos y la migración de datos. Para implementar la replicación, CephFS organiza los datos en grupos de colocación, unidades lógicas intermedias entre los objetos y los dispositivos de almacenamiento. Cada grupo de colocación se asigna a un conjunto de OSDs mediante el algoritmo CRUSH y uno de los OSD se define como OSD primario, responsable de coordinar todas las operaciones sobre objetos del grupo de colocación. Cuando un cliente realiza una escritura, envía la operación al OSD primario, que la escribe localmente y en paralelo la reenvía a los OSD secundarios que almacenan las réplicas del objeto. Cada OSD réplica escribe el objeto en su almacenamiento local y confirma la operación al OSD primario. Una vez que el OSD primario y todas las réplicas confirman que la escritura fue persistida, el OSD primario envía el resultado de la operación al cliente. El modelo de replicación descrito mantiene la consistencia entre réplicas sin coordinación explícita por parte del cliente. Para implementar la detección y recuperación ante fallos, cada OSD monitorea el estado de sus pares mediante mensajes de heartbeat; si un OSD deja de responder y el fallo persiste en el tiempo, se inicia un proceso de rereplicación para restaurar el nivel deseado de réplicas en otros OSD activos del sistema. Para preservar la integridad de los datos, los OSD ejecutan periódicamente un proceso que compara los objetos de un grupo de colocación con sus réplicas para detectar inconsistencias. Cada OSD almacena los objetos en un sistema de almacenamiento subyacente denominado BlueStore, que escribe

directamente sobre el dispositivo de bloques y elimina la capa de un sistema de archivos tradicional. BlueStore gestiona sus metadatos internos mediante una base de datos clave-valor y utiliza copy-on-write: en lugar de sobrescribir datos existentes en disco, escribe en una región no utilizada del almacenamiento. Además, incorpora checksums para verificar la integridad de los datos y tiene soporte para compresión.

Las principales ventajas de CephFS son su alta escalabilidad horizontal, su tolerancia a fallos automática y su adaptación a patrones de acceso mediante el particionado dinámico de subárboles. La principal desventaja de CephFS es su complejidad operativa, dada la cantidad de componentes que es necesario configurar y gestionar.

CephFS es software libre y de código abierto, distribuido bajo licencia LGPL y su desarrollo es impulsado por una comunidad de múltiples organizaciones. Existe además soporte comercial a través de proveedores como IBM y Canonical.

3.1.3. Lustre

Lustre es un sistema de archivos distribuido y paralelo orientado al uso en sistemas de computación de alto desempeño (Lustre developers, 2017). Su arquitectura es modular, escalable y de código abierto, cuenta con el respaldo de una gran comunidad y es ampliamente utilizado en centros de supercómputo y entornos empresariales. La compatibilidad de la interfaz de Lustre con el estándar POSIX permite utilizar Lustre sin modificaciones en las aplicaciones.

Los componentes de la arquitectura de Lustre son: los clientes, que montan y utilizan el sistema de archivos; el servidor de gestión (MGS), que almacena y distribuye la información de configuración del sistema de archivos; los servidores de metadatos (MDS), encargados de servir los metadatos del sistema de archivos; y los servidores de objetos (OSS), que gestionan el almacenamiento físico de los datos y atienden las operaciones de entrada y salida. La configuración se almacena físicamente en uno o más objetivos de gestión (MGT) gestionados por el MGS, los metadatos se almacenan físicamente en uno o más objetivos de metadatos (MDT) gestionados por los MDS y los datos residen en objetivos de almacenamiento de objetos (OST) gestionados por los OSS. En el cliente un volumen lógico de objetos (LOV) agrega los distintos OST y un volumen lógico de metadatos (LMV) agrega los distintos MDT para que

el cliente utilice un único espacio de nombres. Cada objetivo (MGT, MDT y OST) almacena sus datos en un sistema de archivos local que actúa como sistema de archivos de base; existen dos opciones, ldiskfs (una versión modificada de ext4) y ZFS.

Lustre implementa un modelo basado en objetos. Cada archivo se representa como uno o más objetos distribuidos en los OST y su ubicación se gestiona mediante metadatos almacenados en los MDT. Internamente, los archivos y objetos se identifican mediante un identificador de archivo (FID) de 128 bits único en todo el sistema de archivos. El FID cumple un rol análogo al del número de inodo en un sistema de archivos local, pero es independiente de los identificadores de los sistemas de archivos de base. La disposición de un archivo sobre los OST se almacena como un atributo extendido en el objeto del MDT correspondiente. Al crear un archivo, el cliente contacta al MDS, que le asigna su FID y su entrada de metadatos y delega en los OST la creación de los objetos correspondientes. Cuando el cliente necesita leer o escribir, primero obtiene del MDT la disposición del archivo y, a partir de ese momento, realiza la entrada y salida directamente con los OSS donde residen los objetos, sin intermediación del MDS.

La distribución de los datos se realiza mediante striping sobre los múltiples objetos que conforman un archivo, almacenados en distintos OST con un patrón RAID 0. El componente que gestiona el striping es el LOV. El número de objetos sobre los que se reparte un archivo (stripe count) y el tamaño de cada fragmento (stripe size) son configurables por archivo, por directorio o para todo el sistema de archivos. Aplicar striping entre los OST permite maximizar el rendimiento al realizar lecturas y escrituras en paralelo, superar el ancho de banda de un único OST y almacenar archivos cuyo tamaño excede la capacidad de un solo objetivo. De forma análoga, en las versiones modernas de Lustre el espacio de nombres de metadatos puede distribuirse sobre varios MDT mediante el entorno de espacio de nombres distribuido (DNE). DNE permite escalar horizontalmente la capacidad de metadatos al asignar distintos subárboles de directorios a distintos MDT e incluso repartir las entradas de un mismo directorio entre varios MDT.

Para controlar la concurrencia Lustre utiliza un gestor de bloqueos distribuido (LDLM) que garantiza la coherencia de los datos entre todos los clientes y servidores. El LDLM del MDT administra los bloqueos sobre los permisos de los inodos y las rutas del espacio de nombres y cada OST mantiene su propio

LDLM para los bloqueos de los fragmentos de archivo que aloja, por lo que el rendimiento del bloqueo escala a medida que el sistema de archivos crece. En los metadatos los MDS emplean además un mecanismo de bloqueo por intención: los clientes especifican no solo el recurso sobre el que desean operar, sino también la operación concreta que desean realizar. Así, el MDS otorga el bloqueo adecuado o realiza la operación e informa del resultado de manera atómica. La ejecución atómica evita múltiples intercambios de mensajes entre el cliente y el MDS y disminuye la latencia. La concurrencia sobre los datos de los archivos se gestiona en los OST mediante bloqueos de extensión. En lugar de bloquear el objeto completo, el LDLM otorga bloqueos sobre rangos de bytes, por lo que varios clientes pueden leer y escribir simultáneamente distintas regiones de un mismo archivo sin entrar en conflicto. Además, el OST puede conceder a un cliente una extensión mayor que la solicitada para anticiparse a accesos contiguos y reducir el número de pedidos de bloqueo. Ante un acceso conflictivo de otro cliente, la extensión otorgada se reduce a la región necesaria.

La comunicación entre clientes y servidores se realiza a través de Lustre Networking (LNet), la API de red propia de Lustre que transporta datos y metadatos. LNet admite simultáneamente distintos tipos de red (por ejemplo, InfiniBand y Ethernet) y permite el enrutamiento de tráfico entre ellas. El acceso directo a memoria remota (RDMA) permite al adaptador de red transferir datos entre la memoria de dos nodos sin intervención de la CPU ni del sistema operativo. Cuando la red subyacente soporta RDMA, LNet lo utiliza para reducir la latencia y descargar trabajo de la CPU. La pila de red se organiza en dos capas. La primera es el módulo LNet, asíncrono, sin conexión e independiente del medio físico. La segunda son los controladores de red de Lustre, módulos cargables y orientados a conexión que implementan el soporte para cada tipo de red concreto. Cada nodo se identifica dentro de una red mediante un identificador de red (NID) con la forma dirección@etiqueta, por ejemplo 10.0.0.1@tcp0. Mediante el enrutamiento propio de LNet el tráfico de Lustre puede atravesar varias redes LNet heterogéneas.

Lustre proporciona tolerancia a fallos a nivel del sistema de archivos mediante conmutación por error (failover), que se apoya en pares de servidores con acceso a un almacenamiento compartido. Los MDT pueden configurarse en modo activo/pasivo, donde un MDS secundario permanece como respaldo y asume el rol del primario si el MDS primario falla, o en sistemas con varios MDT en modo activo/activo, donde se reparte la carga de metadatos entre

los MDS. Los OST suelen configurarse en una disposición activa/activa con balanceo de carga: dos OSS comparten el acceso a los mismos OST; cada OSS actúa como primario de la mitad de los OST y como respaldo de la otra mitad de los OST; si un OSS falla, su par asume los OST afectados. Tras una falla, los clientes vuelven a aplicar las transacciones que estaban en curso y aún no se habían confirmado en disco. Lustre, por defecto, no implementa replicación de datos ni duplica los objetos entre distintos OST, sino que delega la redundancia en el almacenamiento de base, por ejemplo mediante arreglos RAID o la redundancia provista por ZFS. Por lo tanto, la conmutación por error protege frente a la caída del servidor que atiende el OST, pero no frente al fallo del medio físico.

3.1.4. IBM Storage Scale

IBM Storage Scale, antes IBM General Parallel File System (GPFS), es un sistema de archivos paralelo y distribuido diseñado por IBM para proporcionar acceso de alto rendimiento, confiabilidad y escalabilidad a volúmenes masivos de datos en entornos con múltiples clientes y cargas de trabajo concurrentes. Su uso está orientado a clústeres HPC, instalaciones empresariales, sistemas de inteligencia artificial y plataformas híbridas que combinan almacenamiento local, en la nube y en el borde, donde se debe poder escalar horizontalmente y tolerar fallos sin interrupciones de servicio (IBM, 2025).

La arquitectura de Storage Scale está basada en un modelo de nodos colaborativos, donde cada nodo puede desempeñar uno o varios roles. Existen nodos clientes que acceden a los datos, nodos servidores que permiten el acceso indirecto a los discos compartidos (NSD servers) y nodos con responsabilidades internas sobre la gestión del sistema de archivos. Storage Scale se instala como una extensión del núcleo del sistema operativo y opera como un sistema de archivos nativo. Al operar como sistema de archivos nativo, mantiene compatibilidad POSIX. En cada nodo se ejecuta un demonio de gestión (mmfsd), responsable de coordinar las operaciones del sistema de archivos. mmfsd mantiene sincronizados los datos y metadatos entre nodos, administra la caché, ejecuta las operaciones de escritura diferida y lectura anticipada y gestiona los mecanismos de recuperación. Cada nodo cuenta con una caché local de datos y metadatos, que se utiliza para ejecutar operaciones en memoria antes de su sincronización con el almacenamiento persistente.

Los datos en Storage Scale se almacenan y se distribuyen mediante un mecanismo interno de striping. Cada archivo se divide en bloques de tamaño configurable y los bloques se reparten secuencialmente entre los distintos discos disponibles. La ubicación de cada bloque dentro del sistema es determinista: a partir del número de bloque dentro del archivo y de la política de distribución de datos se puede calcular qué disco contiene el bloque. El mecanismo determinista evita el acceso a una base de datos centralizada de metadatos en cada acceso.

Para los metadatos, Storage Scale implementa una estructura de inodos similar a la de los sistemas UNIX: cada archivo es representado por un inodo que contiene la información necesaria para localizar los bloques de datos. Para archivos pequeños, los datos se pueden almacenar directamente en el inodo, que mejora el rendimiento al evitar lecturas adicionales. Los archivos de mayor tamaño utilizan punteros directos, indirectos y doblemente indirectos según su longitud. El sistema permite definir el tamaño mínimo de asignación, que puede ser menor al bloque físico; este mecanismo reduce la fragmentación interna y, al poder transferir archivos de tamaño menor a un bloque sin tener que enviar el bloque entero, mejora la utilización del disco y el rendimiento para archivos pequeños.

La gestión de acceso concurrente a archivos se basa en un sistema de tokens distribuido. Cada operación que se realiza sobre un archivo (lectura, escritura, cambio de atributos, etc.) requiere la posesión de uno o más tokens. Los tokens son gestionados por el File System Manager (FSM), que puede ser cualquier nodo del clúster y su elección depende de la configuración y del estado actual del sistema. Cuando un nodo desea realizar una operación para la cual no tiene tokens válidos, solicita su asignación al FSM, que puede revocar tokens en poder de otros nodos si es necesario. El control de concurrencia y la coherencia distribuida de los datos se logra sin necesidad de un mecanismo centralizado de bloqueos. Múltiples procesos pueden realizar operaciones paralelas sobre un mismo archivo, siempre que no sean conflictivas entre sí.

El acceso físico a los datos se realiza mediante discos presentados al sistema como Network Shared Disks (NSD). Los discos pueden ser accedidos directamente por los nodos que disponen de conectividad física hacia ellos o indirectamente a través de NSD servers, que actúan como intermediarios para los nodos sin acceso directo al almacenamiento. Así, incluso nodos sin conexión física a los dispositivos de almacenamiento pueden participar del sistema

de archivos. En caso de que un nodo pierda acceso directo a un disco, puede redirigir sus operaciones hacia otro NSD server que tenga conectividad con el disco. La tolerancia a fallos se extiende a todos los componentes críticos del sistema: si un nodo FSM falla, otro nodo puede asumir su rol automáticamente. Si se pierde acceso a una réplica de un bloque de datos, el sistema recurre a otras réplicas disponibles.

El sistema de archivos permite replicar datos y metadatos y puede configurarse para mantener hasta tres copias de cada bloque en discos pertenecientes a diferentes grupos de falla. Los grupos de falla son definidos por el administrador y representan conjuntos de discos que comparten un punto de fallo común, por ejemplo un nodo, una controladora o un rack. Al distribuir las réplicas entre distintos grupos se garantiza la disponibilidad de datos incluso ante fallos catastróficos de hardware. El sistema detecta automáticamente la pérdida de una réplica y activa un proceso de re-replicación que restaura el nivel deseado de protección en segundo plano.

El sistema de archivos se gestiona mediante una serie de comandos de línea que permiten controlar la configuración del clúster, definir políticas de almacenamiento, monitorear el estado de los componentes y gestionar la expansión o reconfiguración del sistema. Storage Scale permite realizar operaciones de mantenimiento sin interrupciones, como el agregado o retiro de discos, la migración de datos entre pools de almacenamiento y la redistribución de datos para optimizar el uso del espacio.

Storage Scale también integra una funcionalidad de exportación de datos mediante protocolos estándar como NFS, SMB, S3 y HDFS. Con esta funcionalidad se puede exponer el sistema de archivos a aplicaciones y clientes que no utilizan Storage Scale de forma nativa. Sin embargo, Storage Scale es una solución de carácter propietario, cuyo uso requiere la previa adquisición de licencias comerciales, por lo que se eleva el costo de implementación en comparación con alternativas de código abierto. Al ser de código cerrado, no se puede modificar el funcionamiento interno de Storage Scale ni adaptar el sistema a requerimientos específicos.

3.1.5. Análisis comparativo

Los cuatro sistemas presentados comparten el objetivo de ofrecer acceso paralelo, escalable y tolerante a fallos sobre grandes volúmenes de datos, pero difieren en la forma en que organizan la arquitectura, distribuyen y localizan los datos, escalan la gestión de metadatos, controlan la concurrencia y proveen redundancia.

En la gestión de metadatos, pNFS, CephFS y Lustre separan explícitamente el procesamiento de metadatos del de datos mediante componentes dedicados: pNFS utiliza un servidor de metadatos separado de los servidores de almacenamiento, CephFS separa un clúster de MDS de un clúster de OSD y Lustre separa los MDS/MDT de los OSS/OST. Storage Scale, en cambio, utiliza un modelo de nodos colaborativos sobre discos compartidos sin un clúster de metadatos dedicado y representa los archivos mediante inodos al estilo UNIX. En los cuatro sistemas de archivos, una vez resuelta la ubicación de los datos, el cliente realiza la entrada y salida directamente hacia los nodos de almacenamiento, sin interactuar con el sistema de metadatos.

Para que el cliente determine la ubicación de cada archivo en los nodos de almacenamiento, CephFS y Storage Scale calculan la ubicación de forma determinística y descentralizada (CephFS mediante el algoritmo CRUSH a partir del nombre del objeto y Storage Scale a partir del número de bloque y la política de distribución) y se evita consultar una estructura central en cada acceso. Lustre y pNFS, en cambio, almacenan explícitamente la disposición del archivo en el subsistema de metadatos. En cuanto a los esquemas de reparto de stripes, Lustre aplica striping configurable al estilo RAID 0 por archivo, directorio o sistema de archivos, Storage Scale reparte bloques de tamaño configurable de forma secuencial, CephFS divide los archivos en stripes almacenados en objetos de RADOS y pNFS admite distribución a nivel de archivo o de bloques.

Respecto a la escalabilidad del sistema de metadatos, pNFS (salvo implementaciones muy específicas) utiliza un único servidor de metadatos, por lo que el servidor de metadatos puede convertirse en un cuello de botella. Los otros tres permiten distribuir los metadatos entre varios nodos; CephFS emplea particionado dinámico de subárboles, redistribuyendo o replicando porciones del espacio de nombres entre los MDS según los patrones de acceso recientes; Lustre utiliza el entorno de espacio de nombres distribuido para asignar subárboles

o entradas de un mismo directorio a distintos MDT; Storage Scale delega la coordinación en un FSM que puede ser cualquier nodo del clúster y reasignarse dinámicamente. Salvo pNFS, todos escalan el rendimiento del sistema de metadatos a medida que se agregan nodos.

Para el control de concurrencia, pNFS utiliza, al igual que NFS, bloqueos por rango de bytes. Lustre implementa LDLM, que combina bloqueos de extensión sobre rangos de bytes en los OST con bloqueos por intención en los metadatos para reducir la latencia de las operaciones del MDS. Storage Scale resuelve la coherencia mediante un sistema de tokens distribuido administrado por el FSM, sin un mecanismo centralizado de bloqueos. En CephFS la consistencia entre réplicas se garantiza a través del OSD primario, que coordina las escrituras sin requerir coordinación explícita del cliente. Lustre y pNFS, al operar a nivel de rango de bytes, permiten que varios clientes escriban simultáneamente en regiones distintas de un mismo archivo sin conflicto.

En la replicación y tolerancia a fallos, CephFS y Storage Scale replican los datos de forma nativa (CephFS mediante grupos de colocación coordinados por un OSD primario, Storage Scale al almacenar hasta tres copias distribuidas entre grupos de falla) y ambos detectan automáticamente la pérdida de una réplica e inician la re-replicación en segundo plano. Lustre, en cambio, no replica los datos, delega la redundancia en el almacenamiento de base y provee disponibilidad mediante conmutación por error sobre pares de servidores con almacenamiento compartido. El failover de Lustre protege frente a la caída de un servidor, pero no frente al fallo del medio físico. En pNFS, la redundancia y tolerancia a fallos dependen exclusivamente de la redundancia del almacenamiento de base.

La compatibilidad con POSIX permite utilizar pNFS, CephFS, Lustre y Storage Scale sin modificaciones en las aplicaciones. pNFS se integra fácilmente en entornos con soporte para NFS y, debido a su arquitectura sencilla, constituye la opción más simple de instalar y configurar, aunque con un rendimiento orientado a escenarios de menor escala. En cambio, CephFS, Lustre y Storage Scale implementan arquitecturas distribuidas más complejas, con un mayor número de componentes que administrar.

Respecto al modelo de desarrollo y la disponibilidad de los sistemas de archivos, las implementaciones de pNFS y los sistemas de archivos CephFS y Lustre son software libre y cuentan con documentación pública, comunidades de desarrollo activas y la posibilidad de inspeccionar, modificar y adaptar el

código fuente a necesidades específicas. El carácter libre facilita su adopción en entornos académicos y en organizaciones que requieren personalizar el sistema o integrarlo con otras herramientas. Storage Scale, en cambio, es un producto propietario desarrollado por IBM, distribuido bajo licencia comercial. Si bien dispone de documentación técnica y es ampliamente utilizado en entornos empresariales y centros de supercomputación, su código fuente no es público y su uso depende de licencias comerciales.

pNFS es adecuado cuando se prioriza la interoperabilidad y la facilidad de despliegue en entornos de escala moderada; CephFS se orienta a escenarios de gran escala con una gran cantidad de clientes y mecanismos integrados de tolerancia a fallos; Lustre apunta a maximizar el ancho de banda agregado en cargas de HPC, con la redundancia delegada al almacenamiento de base; y Storage Scale ofrece un conjunto integrado de características de nivel empresarial con soporte comercial al precio de un modelo propietario.

3.2. Análisis de trabajos relacionados

En esta sección se presentan cinco trabajos relacionados al despliegue y evaluación del rendimiento de sistemas de archivos paralelos.

Wang et al. (2011) evaluaron Lustre 1.8.3 en Red Hat Enterprise Linux (RHEL) 5.4 sobre tres topologías diferentes: la primera con un MDS, tres OSS con tres OST cada uno y un cliente; la segunda con un MDS, dos OSS con tres OST cada uno y dos clientes; la tercera con un MDS, seis OST y dos clientes, sin especificar la cantidad de OSS. En los tres escenarios se utilizó ldiskfs como sistema de archivos de base. En una primera fase se evaluó el rendimiento del sistema de archivos local con la suite de benchmark Iozone y luego se midió el límite de transmisión de la red con la herramienta ib_send_bw, específica para Infiniband. Los experimentos de rendimiento del sistema de archivos paralelo también se realizaron con Iozone. Se varió el tamaño de registro, cantidad de clientes y parámetros de striping. Con dos clientes y seis OST se alcanzaron valores de rendimiento cercanos al límite impuesto por el ancho de banda de los discos pero lejanos al límite de la red. Los autores concluyeron que el ancho de banda del almacenamiento es el cuello de botella principal y que el número de clientes y de stripes son los factores determinantes del rendimiento.

Zhao y Hu (2010) presentaron una evaluación de rendimiento de Lustre y un modelo de predicción de rendimiento basado en teoría gris, una técnica estadística que permite analizar relaciones entre variables con pocas muestras, con el objetivo de analizar correlaciones, estimar el rendimiento al variar parámetros y determinar qué parámetros influyen más en el rendimiento. Los factores considerados fueron el número de OSS, el tipo de registro de transacciones (journals) de los OST, el tipo de discos, el tipo de conexión del almacenamiento y la cantidad de hilos de E/S generados por el benchmark hacia cada OST. Zhao y Hu llevaron a cabo experimentos sobre cuatro escenarios distintos: un OSS contra dos OSS, registro de transacciones interno contra registro de transacciones externo en otro dispositivo físico, discos SAS contra discos SATA y conexiones de los OST al OSS directas contra conexiones daisy-chain. Las operaciones evaluadas fueron de lectura, escritura y reescritura, ejecutadas en dos equipos Sun Fire X4240 interconectados por una red de 10 Gb/s. Utilizaron el sistema operativo RHEL 5.2 y Lustre 1.6.6. Para medir el rendimiento del sistema de archivos utilizaron el benchmark OBDFilter-survey. Los autores notaron que Lustre mostró mayor rendimiento al incrementar la cantidad de hilos de E/S generados por el benchmark hacia cada OST y al agregar más OSS. Además, las configuraciones con discos SAS, registros de transacciones externos y conexiones directas superaron a las configuraciones con discos SATA, registros de transacciones internos y conexiones daisy-chain. Finalmente, propusieron un modelo basado en teoría gris capaz de predecir el rendimiento con errores relativos del 4 % al 11 %.

Saini et al. (2012) estudiaron el rendimiento de Lustre en el supercomputador Pleiades de la NASA. El trabajo consistió en cuatro etapas: caracterizar los patrones de acceso a datos mediante la ejecución de cinco aplicaciones científicas representativas de la carga de trabajo institucional, desarrollar una herramienta para recolectar métricas de rendimiento de Lustre, comparar su desempeño con NFS y analizar el efecto de parámetros como el tamaño y número de stripes sobre la eficiencia de lectura y escritura. Los autores desarrollaron la herramienta NAS Lustre Performance Package, que extrae para cada trabajo métricas como el volumen de datos leído y escrito, la cantidad de aperturas y cierre de archivos y la distribución de tamaños de llamadas remotas entre el cliente y los OST. Además, compararon latencias entre Lustre y NFS mediante trazas de llamadas al sistema. Luego analizaron el impacto del tamaño y número de stripes, el uso de múltiples OST y las políticas de caché en la

ejecución de las aplicaciones y en la ejecución de los benchmarks paralelos IOR y SWR. Los resultados mostraron cargas de E/S principalmente secuenciales centralizadas en un proceso maestro y una gran cantidad de llamadas remotas de 4 KB asociadas a patrones ineficientes. Se mejoró el ancho de banda al aumentar el número de OST y al ajustar el tamaño de stripe y el recuento de stripes al tamaño de transferencia de datos efectivo de las aplicaciones utilizadas. Además, en los experimentos con IOR se halló que en general, los procesos que leyeron y escribieron su propio archivo obtuvieron mejor rendimiento que los procesos que accedieron en paralelo a un único archivo compartido.

Wang et al. (2013) evaluaron el rendimiento y la escalabilidad del sistema de archivos paralelo Ceph en entornos HPC. El trabajo tuvo como objetivos analizar el comportamiento de Ceph en operaciones de bloque y de archivos en una infraestructura equivalente a la de un centro de supercómputo y estudiar su viabilidad como alternativa a Lustre. Los autores realizaron los experimentos en el Oak Ridge Leadership Computing Facility en un sistema de almacenamiento DDN SFA10K conectado a cuatro servidores de datos con conectividad Infiniband QDR. Midieron primero el rendimiento de los discos y las controladoras de forma local para obtener una cota superior del rendimiento del hardware y luego evaluaron la capa RADOS y la CephFS mediante los benchmarks RADOS Bench e Interleaved or Random (IOR) respectivamente. Encontraron que el autotuning TCP limitó considerablemente el rendimiento de lectura en el benchmark RADOS y que la escalabilidad aumentó casi linealmente a medida que se agregaron OSD hasta un máximo de nueve y continuó aumentando, no linealmente, hasta 11. Se encontró que el rendimiento obtenido en la capa de objetos no se correspondió a un buen rendimiento en el sistema de archivos, dado que las lecturas pequeñas presentaron un comportamiento anómalo y las lecturas y escrituras grandes alcanzaron aproximadamente la mitad del rendimiento medido en RADOS Bench. Posteriormente se realizaron ajustes en la configuración de búferes TCP, las políticas de registros de transacciones (journaling) y el número de hilos de operación en los OSD, se desactivó el cálculo de CRC32 en los clientes, se instaló un núcleo del sistema operativo más reciente y se aumentó el tamaño de la caché read-ahead del núcleo. Luego de las optimizaciones, el rendimiento de Ceph alcanzó aproximadamente 70 % del rendimiento bruto del hardware en RADOS y 62 % al acceder mediante CephFS. Además, se identificó que el registro de transacciones de metadatos y datos es apropiado para servidores de almacenamiento de

uso general pero en entornos de computación de alto rendimiento se convierte en un cuello de botella. Tener registros de transacciones independientes para metadatos y datos requiere aproximadamente el doble de ancho de banda que utilizar un único registro de transacciones.

Chowdhury et al. (2019) presentaron una caracterización y evaluación de rendimiento del sistema de archivos paralelo BeeGFS en un entorno de computación de alto desempeño utilizado para la ejecución de aplicaciones de aprendizaje profundo. Los experimentos se ejecutaron en el clúster Catalyst, con BeeGFS 7.1.2 y doce servidores interconectados por Infiniband QDR. La instalación contó con doce procesos de servidor de almacenamiento de objetos y 36 procesos de servidor de metadatos distribuidos equitativamente entre los hosts. Cada OSS administró dos OST de 15 TB cada uno sobre discos mecánicos y cada MDS administró un MDT de 721 GB sobre unidades de estado sólido. El trabajo tuvo como objetivos caracterizar el rendimiento de datos y metadatos y analizar si BeeGFS es idóneo para cargas de aprendizaje profundo que utilizan conjuntos de datos formados por millones de archivos pequeños. Los autores emplearon el benchmark IOR para medir el rendimiento de datos y el benchmark MDTest para medir el rendimiento de metadatos en los casos de un archivo por proceso (FPP) y un archivo compartido por todos los procesos (SSF). Los autores ejecutaron cargas de trabajo basadas en AlexNet, ResNet-50 y un pipeline distribuido basado en TensorFlow y Horovod. En los experimentos se varió el número de clientes, la cantidad de procesos por cliente, el tamaño de stripe y se realizó la evaluación de escalabilidad en hasta 24 nodos. Se encontró que BeeGFS ofreció buena escalabilidad de ancho de banda en cargas FPP de lectura y escritura y que en los accesos SSF el rendimiento fue menor, con la mitad del ancho de banda alcanzado en FPP. También se halló que aumentar la cantidad de stripes por encima de valores moderados no mejoró linealmente el rendimiento debido al costo adicional de coordinación entre OST. Además, la organización jerárquica de los archivos del conjunto de datos de entrenamiento en múltiples directorios distribuidos mejoró el rendimiento de las operaciones de metadatos en comparación con una organización en un único directorio plano o con un archivo compartido por todos los procesos. Aunque las aplicaciones de aprendizaje profundo impusieron una carga elevada al sistema de archivos y redujeron el rendimiento respecto del medido con los benchmarks, BeeGFS mantuvo un rendimiento aceptable.

Los trabajos revisados coinciden en señalar al almacenamiento como el principal factor limitante del rendimiento en sistemas de archivos paralelos para HPC y en identificar al número de OST/OSD, al número de stripes y al modo de acceso como los parámetros más influyentes. Varios de ellos mencionan que el rendimiento del modo de acceso de un archivo por proceso superó al de archivo compartido y que el rendimiento del sistema de archivos (FS) dependió fuertemente del sistema de archivos de base y de su política de registro de transacciones.

La Tabla 3.1 presenta una comparación de los cinco trabajos relacionados.

trabajo	*sistema de archivos*	*configuración*	*método*	*hallazgos*
Wang et al. (2011)	Lustre 1.8.3 (ldiskfs)	RHEL 5.4, InfiniBand, hasta seis OST	Iozone, ib_send_bw	El almacenamiento fue el cuello de botella. Los factores más determinantes fueron el número de clientes y stripes.
Zhao y Hu (2010)	Lustre 1.6.6	2× Sun Fire X4240, red 10 Gb/s	OBDFilter-survey	Más hilos de E/S por OST y más OSS mejoraron el rendimiento. SAS, registro de transacciones externo y conexión directa superaron a SATA, registro de transacciones interno y conexión daisy-chain.
Saini et al. (2012)	Lustre	Pleiades (NASA)	IOR, NLPP, trazas	Ajustar stripes al tamaño de transferencia mejoró el rendimiento. El rendimiento de un archivo por proceso superó al rendimiento de un archivo compartido.
Wang et al. (2013)	Ceph	OLCF; DDN SFA10K, InfiniBand QDR	RADOS Bench, IOR	Escalabilidad casi lineal hasta nueve OSD. Tras ajustes el rendimiento de CephFS alcanzó 62 % del hardware. Registro de transacciones de datos y metadatos fue un cuello de botella.
Chowdhury et al. (2019)	BeeGFS 7.1.2	Cluster Catalyst; InfiniBand QDR, hasta 24 nodos	IOR, MDTest; AlexNet, ResNet-50, TF+ Horovod	Buena escalabilidad FPP, en SSF el rendimiento cayó a la mitad. Aumentar la cantidad de stripes no escaló linealmente. La jerarquía de directorios mejoró el rendimiento de metadatos.

Tabla 3.1: Resumen de los trabajos relacionados de despliegue y evaluación de sistemas de archivos paralelos.

Capítulo 4

Diseño de la solución

En este capítulo se presenta la solución propuesta para alcanzar los objetivos del proyecto. En primer lugar se describe el sistema de archivos paralelo seleccionado y el funcionamiento de los sistemas de archivos de base que utiliza. A continuación se detalla la infraestructura empleada para su despliegue, incluyendo el hardware, el software y la configuración de red. Finalmente se presenta la configuración de los servidores de almacenamiento utilizada en los experimentos y el rendimiento máximo teórico del sistema de almacenamiento.

4.1. Sistema de archivos elegido

El sistema de archivos paralelo elegido para el despliegue y la evaluación experimental de este proyecto fue Lustre. La elección está basada en el análisis comparativo presentado en la Sección 3.1.5 y en las características de la infraestructura disponible. Lustre está diseñado para entornos de computación de alto desempeño, es libre, de código abierto y es uno de los sistemas de archivos paralelos más utilizados en centros de supercómputo: más del 60 % de los centros de supercómputo que operan algunas de las 100 supercomputadoras más rápidas del mundo utilizan Lustre (OpenSFS, 2026). Existe una comunidad activa, documentación abundante y un conjunto de trabajos previos de despliegue y evaluación (varios analizados en la Sección 3.2) que sirvieron de referencia metodológica. Los otros sistemas de archivos paralelos considerados no se ajustan al objetivo del trabajo. pNFS es una extensión de NFS que habilita el acceso paralelo a los datos, pero mantiene un diseño orientado principalmente a la interoperabilidad y la compatibilidad antes que a maximizar

el rendimiento. CephFS ofrece una escalabilidad casi lineal y una arquitectura muy flexible, pero está orientado de forma más general al almacenamiento masivo y de objetos. IBM Storage Scale, si bien es un sistema utilizado en varios centros de supercómputo, de calidad empresarial y de altas prestaciones, es de licencia propietaria y requiere adquirir licencias comerciales que limitan su disponibilidad y su estudio en un proyecto académico.

Además, Lustre soporta dos sistemas de archivos de base de características distintas para utilizar en los OST y MDT: ldiskfs y ZFS. Por lo tanto permite comparar, bajo idénticas condiciones de hardware, red y carga, el comportamiento de un sistema de archivos de base basado en bloques (ldiskfs) frente a uno con copy-on-write y compresión (ZFS). Como desventaja, Lustre no provee replicación de datos a nivel del sistema de archivos, pero los mecanismos de recuperación de fallas de los sistemas de archivos no se estudiaron en este proyecto.

4.2. Sistemas de archivos de base

Lustre no accede directamente a los dispositivos de almacenamiento, sino que utiliza un sistema de archivos de base que ejecuta localmente en cada servidor de almacenamiento y es responsable de la organización física de los datos en el disco. Cada OST y cada MDT se puede formatear con uno de los dos sistemas de archivos de base soportados por Lustre: ldiskfs o ZFS. A continuación se describen sus características generales y se explica su funcionamiento.

4.2.1. ext4/ldiskfs

Fourth extended filesystem (ext4) es un sistema de archivos con registro de transacciones utilizado en sistemas Linux, sucesor de ext3 y ext2. ext4 organiza el espacio del dispositivo en bloques de tamaño fijo (típicamente 4 KiB) y mantiene la información de cada archivo en una estructura denominada inodo, que almacena los atributos del archivo (permisos, propietario, marcas de tiempo, tamaño) y la ubicación de sus bloques de datos en el dispositivo. El dispositivo se divide en una serie de grupos de bloques y al asignar bloques nuevos se procura mantener los bloques de cada archivo dentro de un mismo grupo a fin de reducir los tiempos de búsqueda. Cada grupo dispone de su propio mapa de bits de bloques, su mapa de bits de inodos y su tabla de inodos. Además,

algunos grupos contienen copias redundantes del superbloque y de los descriptores de grupo para proteger la información del sistema de archivos frente al deterioro del comienzo del dispositivo. El superbloque almacena información global, como la cantidad de bloques, la cantidad de inodos y otros datos de mantenimiento (The Linux Kernel Developers, 2026; Silberschatz et al., 2018).

Para mantener la consistencia ante fallos, ext4 utiliza un registro de transacciones. El registro de transacciones protege al sistema de archivos frente a inconsistencias en los metadatos en caso de una caída del sistema: antes de aplicar una modificación a las estructuras del sistema de archivos, se registra la operación en un área reservada del dispositivo para que, ante un corte de energía o un fallo, se pueda reconstruir un estado consistente reproduciendo o descartando las transacciones incompletas. Por defecto se registran únicamente los metadatos, pero también es posible registrar los datos completos, a cambio de un menor rendimiento.

Una diferencia de ext4 respecto a sus predecesores es el uso de extents. Un extent describe un rango contiguo de bloques físicos mediante su bloque lógico inicial, su longitud y su bloque físico inicial. En lugar de asociar cada bloque lógico de un archivo a un bloque físico mediante bloques indirectos, el uso de un único extent para representar regiones contiguas reduce la cantidad de metadatos necesarios para representar archivos grandes. Cuando un archivo requiere más extents de los que caben en el inodo, los extents se organizan en un árbol de extents.

En ext4 se utilizan mecanismos orientados a reducir la fragmentación externa. Uno de ellos es la asignación múltiple de bloques, que permite reservar de forma anticipada regiones contiguas del dispositivo para un archivo al asignarle espacio, con el fin de favorecer que sus datos queden almacenados en uno o pocos extents contiguos. Otro mecanismo es la asignación diferida; el sistema de archivos pospone la decisión sobre la ubicación física de los bloques hasta el momento en que los datos se escriben en el dispositivo para seleccionar ubicaciones que reduzcan la fragmentación.

ldiskfs es un sistema de archivos de base utilizado por Lustre y consiste en una versión de ext4 con un conjunto de modificaciones para aumentar el rendimiento del sistema de archivos (The Lustre Developers, 2025).

4.2.2. ZFS

ZFS es un sistema de archivos que integra en una misma capa las funciones de sistema de archivos y de gestor de volúmenes. ZFS administra directamente el conjunto de dispositivos físicos en lugar de operar sobre un volumen lógico provisto por una capa inferior.

ZFS organiza el almacenamiento en un conjunto denominado pool (o zpool) que se construye con la combinación de uno o más dispositivos virtuales (vdevs, agrupaciones lógicas de dispositivos físicos). Un vdev puede ser un único dispositivo o una agrupación que aporte redundancia, como un espejo o un arreglo raidz. Un arreglo raidz emplea paridad para tolerar fallos de manera similar a RAID 5/6 y admite uno, dos o tres discos de paridad. La redundancia se establece dentro de cada vdev y los datos del pool se distribuyen mediante striping entre todos sus vdevs. Cuando un vdev falla (pero el pool conserva redundancia suficiente), ZFS continúa operando con los vdevs restantes y, al reemplazar el dispositivo defectuoso, reconstruye automáticamente los datos sobre el nuevo dispositivo (OpenZFS, 2020).

Al crear cada vdev se establece un valor para una propiedad llamada ashift, que representa el menor tamaño posible para cada operación de E/S para el vdev (2^{ashift} bytes). En el sistema de archivos se establece un valor para una propiedad llamada recordsize, que representa la unidad básica de datos que ZFS utiliza en las operaciones internas sobre los archivos. El valor por defecto de recordsize es 128 KiB y puede fijarse en cualquier potencia de dos entre 512 bytes y 1 MiB. Habilitar extensiones permite valores de recordsize de hasta 16 MiB.

ZFS aplica copia en escritura (copy-on-write) sobre los archivos: en vez de sobrescribir datos existentes, cuando un bloque se modifica, se escribe una nueva copia en otra ubicación del dispositivo y, una vez completada la escritura, se actualizan los metadatos para que el archivo apunte al nuevo bloque. Por lo tanto, nunca se confirma una escritura parcial: ante un fallo queda siempre la versión previa y correcta de los datos. Las escrituras se organizan en un modelo transaccional. Las operaciones se agrupan en transacciones, que a su vez se agrupan en grupos de transacciones. Cuando un grupo de transacciones se sincroniza a disco, todas las transacciones que contiene se consideran completas. Cuando el almacenamiento físico no logra atender las escrituras al ritmo

que llegan, ZFS introduce un retardo controlado sobre la asignación de nuevas transacciones, que ajusta la tasa de ingreso al rendimiento del dispositivo.

La integridad de los datos se verifica mediante sumas de comprobación. La suma de comprobación de cada bloque se calcula en el momento de la escritura y se almacena en el puntero que referencia el bloque. Cuando el pool dispone de redundancia, al detectar una discrepancia en la suma de comprobación calculada con la almacenada se pueden reparar los datos automáticamente. Además, periódicamente se realizan recorridos de verificación de los datos, donde se detectan y corrigen errores.

Para acelerar los accesos a datos y metadatos, ZFS mantiene en memoria principal una caché llamada Adaptive Replacement Cache (ARC). ARC almacena datos y metadatos y puede ampliarse mediante un dispositivo de caché de segundo nivel (L2ARC) (OpenZFS, 2020).

ZFS admite compresión en el almacenamiento de datos y metadatos. Cuando la compresión está habilitada, cada bloque lógico puede ocupar un número menor de sectores físicos.

4.3. Infraestructura utilizada

El despliegue de Lustre se realizó en una infraestructura similar a ClusterUY, servidores interconectados por una red de alta velocidad. La infraestructura utilizada para los servidores de almacenamiento y metadatos consistió en cinco servidores, de marcas Dell y HPE. Para los clientes que utilizaron el sistema de archivos, por limitaciones de recursos de hardware no fue posible utilizar servidores. Por lo tanto, se optó en su lugar por utilizar equipos de escritorio, en su mayoría de altas prestaciones. En infraestructuras de cómputo de alto desempeño es frecuente la coexistencia de nodos con configuraciones de CPU, memoria y almacenamiento distintos entre sí. De la misma manera, la infraestructura en la que se desarrollaron los experimentos se construyó como un sistema heterogéneo, con servidores de distintas generaciones y recursos.

Todos los dispositivos fueron equipados con una tarjeta de red PCIe de 10 Gb/s y luego fueron interconectados mediante un switch Ethernet marca GoodTop de siete puertos RJ45 y un puerto SFP+, de 10 Gb/s de velocidad y 160 Gb/s de capacidad de conmutación. La capacidad de conmutación fue suficiente para que todos los puertos operen en modo full-duplex a la mayor velocidad posible sin que la capacidad de conmutación sea un cuello de bote-

lla. En un switch con ocho puertos de 10 Gb/s, la capacidad de conmutación requerida es calculada como la suma de las velocidades de transmisión y recepción de todos los puertos: ocho puertos recibiendo y enviando 10 Gb/s equivale a un total de 160 Gb/s, que es igual a la capacidad de conmutación del switch utilizado. Para verificar la velocidad de interconexión se midió la velocidad de transmisión entre los nodos conectados a la red de 10 Gb/s con la herramienta iperf3 y se obtuvieron resultados coherentes con una interconexión de 10 Gb/s sobre TCP.

Los servidores utilizados poseían una interfaz de red dedicada a la gestión remota, con funcionalidades como apagado y encendido, gestión de dispositivos conectados, diagnóstico y reporte de errores y consola remota. Las interfaces de gestión fueron conectadas entre sí mediante un switch Ethernet de 1 Gb/s y, para no utilizar un puerto del switch de 10 Gb/s para interconectar ambos switches, se conectó un puerto del switch de gestión a una interfaz de red de 1 Gb/s utilizable por el sistema operativo en uno de los servidores. Desde el servidor conectado al switch de gestión, con las reglas de reenvío correctamente configuradas para enrutar la subred de gestión por la interfaz conectada al switch de gestión, fue posible acceder a las interfaces de gestión de todos los servidores.

Cuatro de los servidores se utilizaron como OSS y cada uno alojó un OST. Los OST fueron formados por varios discos mecánicos configurados con RAID por hardware, RAID por software o sin RAID según el caso. Un servidor con dos discos sólidos se utilizó como MDS y MGS; un disco fue utilizado exclusivamente como MDT y una partición del disco sólido utilizado por el sistema fue utilizada para el MGT. Por limitaciones de recursos los discos del MDS/MGS no fueron replicados con RAID u otra alternativa similar. El rol dual del servidor MDS/MGS no afecta el rendimiento, ya que las interacciones con el MGS por parte de los dispositivos son muy esporádicas y los requerimientos de espacio y procesamiento para que funcione son mínimos.

Para la numeración de las interfaces de red de los dispositivos se partió del prefijo privado 10.0.0.0/8, se utilizó el prefijo 10.0.0.0/24 para la subred de datos (10 Gb/s) y se utilizó el prefijo 10.0.1.0/24 para la subred de gestión (1 Gb/s).

La Figura 4.1 presenta el diagrama de red que contiene los nodos de la infraestructura del entorno experimental, sus roles y conexiones.

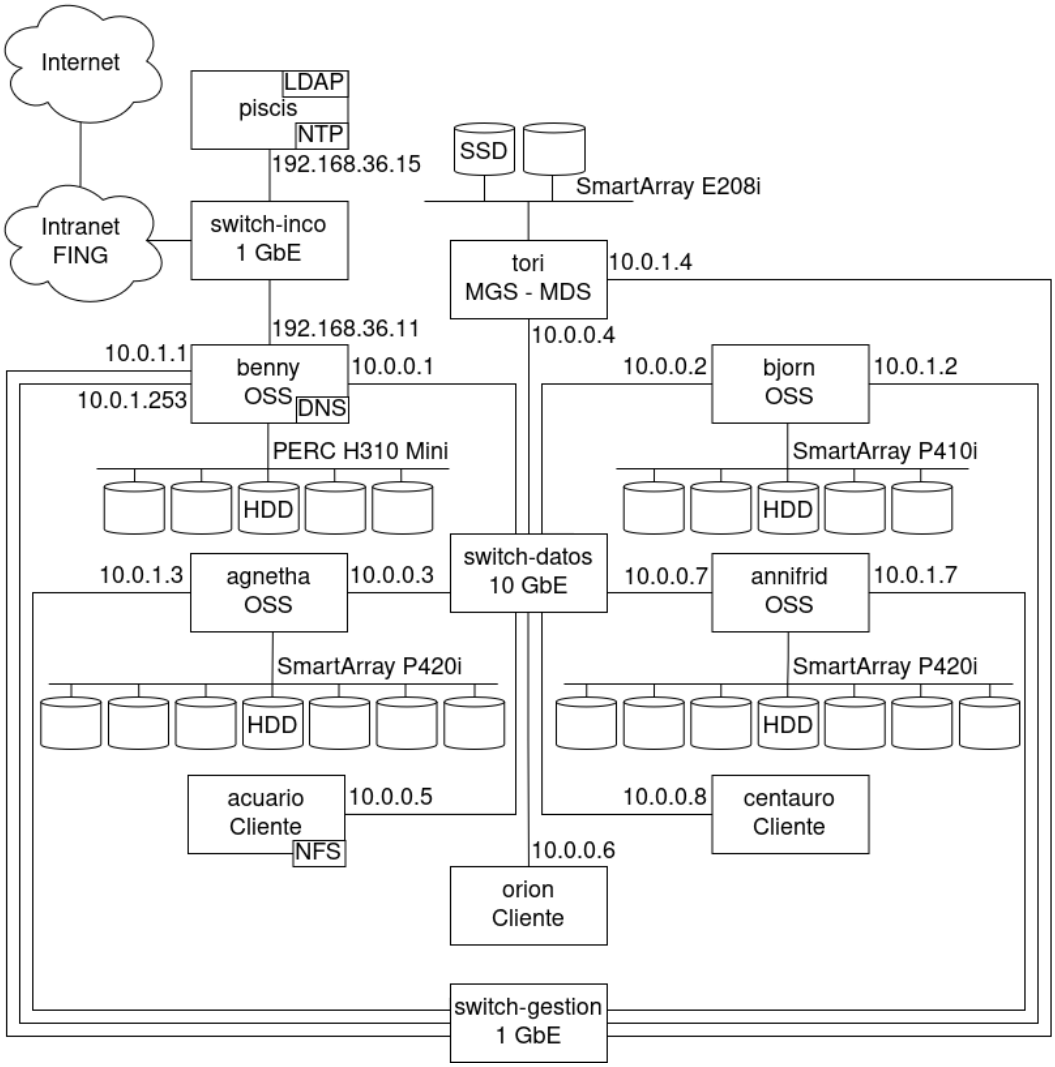


Figura 4.1: Infraestructura del entorno experimental

La Tabla 4.1 detalla las características de cómputo de cada nodo (modelo, procesador y memoria) y su rol en el sistema de archivos paralelo desplegado.

La Tabla 4.2 detalla las controladoras de almacenamiento y los discos físicos en los servidores de almacenamiento.

En el nodo bjorn, aunque los discos utilizados soportaban 6 Gb/s de velocidad, el enlace se negoció a 3 Gb/s ya que la controladora solo era compatible con velocidades de 3 Gb/s.

nodo	*rol*	*modelo*	*procesador*	*RAM*
tori	MDS, MGS	HPE ProLiant DL380 Gen10	2× Intel Xeon Gold 6138 2,0 GHz, 20 núcleos, 40 hilos	128 GB DDR4-2666 ECC
agnetha	OSS	HPE ProLiant DL385p Gen8	2× AMD Opteron 6376 2,3 GHz, 16 núcleos, 16 hilos	128 GB DDR3-1600 ECC
annifrid	OSS	HPE ProLiant DL385p Gen8	2× AMD Opteron 6366 HE 1,8 GHz, 16 núcleos, 16 hilos	128 GB DDR3-1600 ECC
benny	OSS	Dell PowerEdge R720	2× Intel Xeon E5-2650 2,0 GHz, 8 núcleos, 16 hilos	64 GB DDR3-1333 ECC
bjorn	OSS	HPE ProLiant DL385 Gen7	2× AMD Opteron 6172 2,1 GHz, 12 núcleos, 12 hilos	24 GB DDR3-1333 ECC
acuario	Cliente	PC genérica	1× Intel Core i9-13900 2,0 GHz, 24 núcleos, 32 hilos	64 GB DDR5-4400
centauro	Cliente	PC genérica	1× Intel Core i7-8700 3,2 GHz, 6 núcleos, 12 hilos	16 GB DDR4-2400
orion	Cliente	PC genérica	1× Intel Core i9-13900 2,0 GHz, 24 núcleos, 32 hilos	64 GB DDR5-4400

Tabla 4.1: Características de hardware de los nodos

nodo	*controladora*	*dispositivos*	*rol*
tori	HPE Smart Array E208i-a SR Gen10. 8 líneas SAS de 12 Gb/s, 8 líneas PCIe 3.0. Sin caché	2×SSD SATA 240 GB, 6 Gb/s	MGS, MDS
agnetha	HPE Smart Array P420i. 8 líneas SAS de 6 Gb/s, 8 líneas PCIe 3.0. 1 GB caché	6×HDD SATA 6 TB 5400–5700 rpm, 6 Gb/s	OSS
annifrid	HPE Smart Array P420i. 8 líneas SAS de 6 Gb/s, 8 líneas PCIe 3.0. Sin caché	6×HDD SATA 6 TB 5700 rpm, 6 Gb/s	OSS
benny	Dell PERC H310 Mini. 8 líneas SAS de 6 Gb/s, 8 líneas PCIe 2.0. Sin caché	4×HDD SATA 10 TB 7200 rpm, 6 Gb/s	OSS
bjorn	HPE Smart Array P410i. 8 líneas SAS de 6 Gb/s, 8 líneas PCIe 2.0. Sin caché	4×HDD SATA 10 TB 7200 rpm, enlace SATA negociado a 3 Gb/s	OSS

Tabla 4.2: Características de almacenamiento y controladoras de los servidores de almacenamiento

4.4. Instalación y configuración de software

El sistema operativo elegido para la infraestructura fue Rocky Linux, una distribución basada en RHEL. Los servidores de almacenamiento debieron utilizar una versión del núcleo de Linux específica para Lustre y la última versión de Rocky Linux compatible con Lustre al momento del despliegue era la 8.10, por lo que para los servidores de almacenamiento se utilizó Rocky Linux 8.10. En los clientes se utilizó Rocky Linux 9.4, la versión más reciente soportada por los binarios de cliente de Lustre al momento del despliegue. Rocky Linux y otras distribuciones basadas en RHEL son ampliamente utilizadas en entornos de cómputo de alto desempeño y entornos empresariales; Lustre publica binarios empaquetados en un repositorio para RHEL, por lo que utilizar RHEL simplifica la instalación y actualización del software, sin necesidad de compilar desde código fuente en cada instalación o actualización. Para todos los nodos se utilizó la versión 2.15.7 de Lustre.

En los clientes y servidores se realizó la instalación del mismo software de base sobre el sistema operativo. Se configuró la autenticación de usuarios mediante un servidor Lightweight Directory Access Protocol (LDAP) para utilizar los mismos identificadores de usuario y grupo (UID/GID) en todos los nodos de la infraestructura; el servidor LDAP se alojó en una computadora conectada a la red local que, por escasez de puertos en el switch de datos, se debía acceder a través de uno de los servidores que reenviaba el tráfico. La misma computadora alojó además un servidor Network Time Protocol (NTP), utilizado para uniformizar la fecha y hora del sistema entre todos los participantes del sistema de archivos paralelo. El directorio home del usuario utilizado para los experimentos, donde se almacenaron los resultados, se alojó en un disco NVMe de alta velocidad en una de las computadoras utilizadas como cliente y el directorio se compartió al resto de los nodos por NFS. Se agregó el repositorio de paquetes de Lustre, para cliente o servidor según el caso y se instalaron herramientas básicas y los programas a utilizar en las mediciones. En los servidores se instaló un núcleo provisto por Lustre, basado en la versión 4.18 del núcleo de Linux, junto con los módulos necesarios para el funcionamiento de Lustre, LNet y el sistema de archivos ldiskfs en formato precompilado (kmod) para la versión exacta del núcleo utilizado. Si bien es posible utilizar un núcleo estándar y compilar los módulos mediante DKMS, esta alternativa incrementa la complejidad del despliegue y el tiempo de instalación, dado que cada actua-

lización del núcleo o de los módulos requiere una recompilación. Además, no todas las versiones del núcleo son compatibles con los módulos de Lustre, por lo que el uso de la versión provista garantiza la compatibilidad de Lustre con el sistema operativo.

Los dispositivos fueron instalados en una subred privada sin conexión a Internet. El nodo benny estaba conectado a Internet por otra interfaz a través de una subred bloqueada por direcciones MAC, por lo que para descargar los paquetes y dar conexión a Internet a todos los dispositivos se configuró Network Address Translation (NAT) (en la interfaz de benny con la subred conectada a Internet) y se configuró la IP de benny en la subred del proyecto como puerta de enlace en el resto de los dispositivos. En benny también se desplegó un servidor Domain Name System (DNS) local para traducir los nombres de host de la subred privada en direcciones IP.

4.5. Configuración de los MDS, MGS y OSS

La primera decisión a tomar respecto a la configuración de los servidores de almacenamiento fue el nivel de RAID a utilizar para las configuraciones de almacenamiento que utilicen RAID. Los niveles de RAID a utilizar en un sistema de archivos Lustre para un entorno de computación de alto desempeño deben tener un buen rendimiento en operaciones de E/S, tener tolerancia a fallas de discos (Lustre no provee replicación de datos a nivel de sistema de archivos, por lo que la tolerancia a fallos debe implementarse directamente en los OSS) y una rápida recuperación frente a fallas de discos. Se analizaron RAID 1+0, RAID 5 y RAID 6 debido a su simplicidad, su capacidad de tolerancia a fallos y a la reducida cantidad de discos necesaria para implementar cada nivel.

En los casos en que se utilizó RAID, se optó por agrupar los discos de los OSS en arreglos RAID 1+0 dado su buen rendimiento en pequeñas escrituras y su baja penalización de rendimiento al ocurrir fallas de discos. En RAID 5 o RAID 6, cada escritura que no abarca una raya completa requiere leer el dato y la paridad existentes, recalcular la paridad y escribir el dato y la nueva información de paridad. Por lo tanto, para pequeñas escrituras en RAID 5 se realizan cuatro operaciones de E/S (dos lecturas y dos escrituras) y en RAID 6 se realizan seis (tres lecturas y tres escrituras, debido a la presencia de dos bloques de paridad). RAID 1+0 escribe los datos directamente en ambos miembros de un espejo. Ante la pérdida de un disco en RAID 5 o RAID 6, la

reconstrucción de un disco implica leer la totalidad del resto de los discos para recalcular la paridad, un proceso potencialmente largo (varias horas o días) y costoso (aumenta la probabilidad de un segundo fallo durante la reconstrucción). En RAID 1+0, ante el fallo de un disco no se debe recalcular paridad y el arreglo continúa funcionando normalmente, aunque puede disminuir ligeramente el rendimiento dado que no se pueden paralelizar operaciones entre miembros de un mismo espejo. Además, la recuperación implica solo copiar los datos del otro miembro del espejo a un nuevo disco. La contrapartida es que la capacidad útil del arreglo en RAID 1+0 es considerablemente menor a las capacidades útiles de RAID 5 y RAID 6. Si N es el número de discos (por simplicidad se asume que todos los discos tienen la misma capacidad, C), la capacidad útil del arreglo RAID 1+0 es $(N \times C)/2$, frente a $(N - 1) \times C$ en RAID 5 y $(N - 2) \times C$ en RAID 6. Además, en RAID 1+0 se requiere que el número de discos sea par. En el MDT no se utilizó RAID, ya que para su despliegue se contó con un único disco.

En agnetha, annifrid y bjorn se utilizó RAID por hardware a través de sus controladoras HPE Smart Array y en benny el RAID se implementó por software mediante la utilidad de Linux Multiple Device Administrator (mdadm). Si bien la controladora Dell PERC H310 de benny permitía configurar RAID 1+0 por hardware, se optó por utilizar RAID por software para evaluar el rendimiento del RAID por software y analizar las posibles diferencias que puedan surgir con los nodos con RAID por hardware.

Para los experimentos con ZFS, para realizar una correcta comparación con ldiskfs, se buscó una configuración lo más similar posible a RAID 1+0. La configuración funcionalmente equivalente a RAID 1+0 consiste en un único pool compuesto por vdevs que contienen dos discos espejados cada uno. Idealmente todos los discos utilizados deben ser accesibles directamente por el sistema operativo, por ejemplo configurando la controladora de RAID en modo pass-through, para que ZFS gestione internamente la redundancia y el acceso a cada disco físico. La configuración descrita solo pudo aplicarse en el nodo benny, en el que la controladora de RAID permitió exponer los discos en modo pass-through y los cuatro discos de datos pudieron ser manejados individualmente por ZFS y ensamblados en dos vdevs espejados. En el resto de los OSS la configuración debió adaptarse a las limitaciones de sus respectivas controladoras de RAID, cuyo firmware no permitió exponer los discos en modo pass-through. En agnetha cada uno de los seis discos se configuró como un

disco lógico RAID 0 y luego ZFS ensambló los seis discos lógicos individuales como tres vdevs espejados, lo más cerca posible a la configuración similar a RAID 1+0 descrita. En annifrid y bjorn la controladora restringía el número de arreglos de discos a un máximo de dos; al estar un arreglo de discos reservado para el sistema operativo, la única opción fue crear el mismo arreglo RAID 1+0 por hardware utilizado en los experimentos con ldiskfs y que ZFS acceda al único disco lógico expuesto. En todos los casos, los pools de ZFS se crearon con un tamaño de bloque lógico de 4 KiB (parámetro ashift=12) para alinear las escrituras con el tamaño de sector físico de los discos. Para los experimentos con ZFS con compresión se eligió utilizar el algoritmo lz4, dado que tiene una buena tasa de compresión con bajo uso de CPU. Las CPU de algunos de los OSS no eran potentes, por lo que elegir otros algoritmos (por ejemplo, zstd en medios o altos niveles de compresión) podría tener un efecto contraproducente al saturar los procesadores. Para el MDT el pool se formó con un solo vdev formado por el disco sólido dedicado para metadatos. El tamaño del bloque lógico de ZFS (parámetro recordsize) fue el que Lustre utiliza por defecto al realizar el formateo: 1 MiB para los OST y 128 KiB para los MDT.

El MGT, en todos los casos, se mantuvo como una partición fija sin RAID ni configurada en un pool ZFS. Al almacenar datos de gestión de Lustre, es accedido mínimamente, aloja muy pocos datos y su configuración no tiene efecto en el rendimiento, por lo que su sistema de archivos de base no se consideró un factor relevante para evaluar.

La Tabla 4.3 presenta las configuraciones de los discos en los OSS para cada sistema de archivos de base.

nodo	*ldiskfs*	*ZFS*
agnetha	RAID 1+0 por hardware	6 discos lógicos RAID 0 3 vdevs espejados
annifrid	RAID 1+0 por hardware	1 disco lógico RAID 1+0 por hardware 1 vdev
benny	RAID 1+0 por software	4 discos en pass-through 2 vdevs espejados
bjorn	RAID 1+0 por hardware	1 disco lógico RAID 1+0 por hardware 1 vdev

Tabla 4.3: Configuración lógica de discos por OSS en ldiskfs y ZFS.

4.6. Rendimiento teórico máximo de los OST

En un servidor de almacenamiento, los discos se conectan a la controladora de almacenamiento a través de un componente pasivo denominado backplane, que actúa como placa de interconexión entre las bahías de discos y la controladora. En algunas configuraciones, el backplane incorpora expansores SAS, circuitos activos que multiplexan varios discos sobre un número reducido de enlaces físicos hacia la controladora. Esta multiplexación puede convertirse en un cuello de botella cuando varios discos transfieren datos en simultáneo. Los nodos utilizados en este proyecto emplearon backplanes de conexión directa (pasivos), sin expansores SAS, por lo que cada bahía dispuso de un enlace físico dedicado hacia la controladora. Por lo tanto, el ancho de banda entre cada disco y la controladora quedó determinado exclusivamente por el estándar de interfaz negociado (SATA a 3 Gb/s o 6 Gb/s según el nodo) y no por limitaciones del backplane. La conexión directa permitió tratar el enlace de cada disco de forma independiente al calcular el rendimiento teórico máximo de los OST.

Para calcular el rendimiento teórico máximo de los OST se tomó como cota superior el mínimo entre los límites impuestos por los discos, sus enlaces dedicados con la controladora y la conexión PCIe de la controladora hacia el servidor. Sea D el conjunto de discos que componen el OST y $d_i \in D$ los discos individuales. Para cada disco, sea V_{d_i} su velocidad máxima de transferencia y $B_{\text{enlace},i}$ el ancho de banda del enlace entre d_i y la controladora a través del backplane del servidor. El aporte máximo de cada disco queda acotado por $S_i = \text{mín}(V_{d_i},\ B_{\text{enlace},i})$ y la velocidad de transmisión agregada máxima aportada por los discos se define como $V_{\text{discos}} = \sum_{d_i \in D} S_i$. Para la controladora se consideró el ancho de banda de la conexión PCIe hacia el servidor B_{PCIe}. El rendimiento teórico máximo agregado del OST se definió como $R_{\text{OST}} = \text{mín}(B_{\text{PCIe}},\ V_{\text{discos}})$. Como en este proyecto cada OSS alojó exactamente un OST, el valor R_{OST} coincide con el rendimiento teórico máximo del correspondiente OSS.

La Tabla 4.4 presenta los discos, la velocidad máxima de transferencia de cada disco y el ancho de banda del enlace para cada nodo. Los V_{d_i} se tomaron de las hojas de datos de los discos (Seagate Technology LLC, 2017; Western Digital Corporation, 2018a,b, 2025). $B_{\text{enlace},i}$ corresponde a la tasa de la interfaz SATA negociada entre el disco y la controladora a través del backplane, obtenida a partir de la generación SATA: 600 MB/s para SATA

de 6 Gb/s y 300 MB/s para SATA de 3 Gb/s. Todos los discos exponen una interfaz SATA de 6 Gb/s según sus hojas de datos, por lo que $B_{\text{enlace},i} = 600$ MB/s, salvo en bjorn, donde la controladora negoció el enlace a 3 Gb/s y $B_{\text{enlace},i} = 300$ MB/s.

nodo	*modelo de disco*	*cantidad*	V_{d_i} *(MB/s)*	$B_{enlace,i}$ *(MB/s)*	S_i *(MB/s)*
agnetha	WD60EFRX-68L	1	175	600	175
agnetha	WD60EFPX-68C	1	180	600	180
agnetha	WD60EZRZ-00G	4	170	600	170
annifrid	WD60EZRZ-00G	6	170	600	170
benny	ST10000VN0004-1Z	3	210	600	210
benny	ST10000VN0004-2G	1	210	600	210
bjorn	ST10000VN0004-1Z	3	210	300	210
bjorn	ST10000VN0004-2G	1	210	300	210

Tabla 4.4: Discos utilizados en cada OST y cota superior de rendimiento por disco.

Se consideró que el enlace PCIe se negocia al mínimo entre la generación y la cantidad de carriles soportados por la controladora y por el puerto PCIe del servidor donde está instalada, por lo que $B_{\text{PCIe}} = \text{mín}(\text{PCIe}_{\text{controladora}}, \text{PCIe}_{\text{puerto}})$. Cada ancho se obtuvo multiplicando la cantidad de carriles por la tasa útil por carril según la codificación de cada generación: 500 MB/s por carril en PCIe 2.0 y 984,6 MB/s por carril en PCIe 3.0. Las generaciones PCIe de las controladoras se tomaron de sus hojas de datos (Dell Inc., 2012; Hewlett Packard, 2013b; Hewlett Packard Enterprise, 2016) y las generaciones de los puertos PCIe de los servidores de sus especificaciones técnicas (Hewlett Packard Enterprise, 2017; Hewlett Packard, 2013a; Dell Inc., 2013). En los cuatro nodos el enlace efectivo fue PCIe 2.0 ×8 carriles = 4000 MB/s: en agnetha y annifrid porque los servidores HPE ProLiant DL385p Gen8 solo disponen de slots PCIe 2.0 y en benny y bjorn porque las controladoras son PCIe 2.0.

La Tabla 4.5 presenta el pico teórico máximo de rendimiento de cada OSS y el total del sistema.

nodo	V_{discos} (MB/s)	B_{PCIe} (MB/s)	R_{OST} (MB/s)
agnetha	1035	4000	1035
annifrid	1020	4000	1020
benny	840	4000	840
bjorn	840	4000	840
Pico teórico total del sistema (MB/s)			3735
Pico teórico total del sistema (MiB/s)			3562

Tabla 4.5: Cotas teóricas agregadas por OSS.

El valor R_{OST} representa un máximo teórico calculado a partir de las especificaciones del hardware y no constituye un rendimiento alcanzable en la práctica, sino una cota superior de referencia. Además, su cálculo permite identificar, para cada nodo, el recurso que limita el rendimiento. Como en los cuatro OSS se cumple que $V_{\text{discos}} < B_{\text{PCIe}}$, entonces $R_{\text{OST}} = V_{\text{discos}}$, por lo que el factor limitante fueron los discos y no la conexión PCIe ni la controladora. Al representar el máximo rendimiento teórico del sistema de almacenamiento, R_{OST} sirve para validar los resultados de los benchmarks; cualquier resultado superior a R_{OST} (con excepción de los experimentos con compresión habilitada) no refleja el rendimiento del sistema de almacenamiento, sino efectos de caché o errores de medición.

R_{OST} supone que cada byte transferido por los discos corresponde a un byte útil para el sistema de archivos, suposición que no se cumple en las configuraciones descritas en la Sección 4.5, ya que todas emplean espejado de datos (en ldiskfs mediante RAID 1+0 y en ZFS mediante vdevs espejados o, por limitaciones de hardware, también RAID 1+0). En todos los casos cada bloque lógico se mantiene en dos copias físicas, por lo que el factor de redundancia es de dos. El espejado afecta de forma asimétrica a las operaciones de lectura y de escritura; en escritura, cada operación lógica corresponde a dos escrituras físicas, una por cada copia del espejo, por lo que el ancho de banda útil de escritura queda acotado por la mitad del ancho de banda agregado de los discos. En lectura, en cambio, el dato solicitado se encuentra en ambos espejos y puede leerse de cualquiera de ellos. En el mejor caso, un esquema de balanceo de lectura entre los dos miembros del espejo permite repartir las soli-

citudes entre ambos discos, por lo que la cota de lectura en el mejor caso no se ve penalizada por la redundancia y mantiene el valor V_{discos}. Las cotas ajustadas por operación se definieron como $R_{\text{OST}}^{\text{lect}} = \text{mín}(B_{\text{PCIe}}, V_{\text{discos}})$ para lectura y $R_{\text{OST}}^{\text{esc}} = \text{mín}(B_{\text{PCIe}}, V_{\text{discos}}^{\text{esc}})$ para escritura. El término $V_{\text{discos}}^{\text{esc}} = \sum_{p \in P} \text{mín}_{d_i \in p} S_i$ es la suma, sobre el conjunto de espejos P que componen el OST, del aporte del disco más lento de cada espejo bajo el mejor caso en el que los espejos agrupan discos de igual o similar velocidad.

En un espejo cada escritura se completa cuando ambas copias fueron escritas, por lo que el rendimiento de cada par queda limitado por el disco más lento del par. En los nodos con discos homogéneos (annifrid, benny y bjorn) ambos miembros de cada espejo presentan el mismo rendimiento, por lo que $V_{\text{discos}}^{\text{esc}} = V_{\text{discos}}/2$. En agnetha, en cambio, los discos son heterogéneos y el mejor emparejamiento es $\{180, 175\}, \{170, 170\}, \{170, 170\}$, cuyo aporte agregado de escritura es 515 MB/s, levemente inferior a $V_{\text{discos}}/2 = 517{,}5$ MB/s.

El término B_{PCIe} en $R_{\text{OST}}^{\text{esc}}$ supone que la duplicación de datos ocurre del lado de la controladora, como sucede en un RAID por hardware, donde cada escritura lógica atraviesa el enlace PCIe una sola vez y la controladora genera ambas copias hacia los discos. Cuando el espejado se realiza por software, como en los vdevs espejados de ZFS, ambas copias atraviesan el enlace PCIe, por lo que el límite útil de escritura impuesto por dicho enlace se reduce a $B_{\text{PCIe}}/2$. En las configuraciones evaluadas, la duplicación del tráfico a través del enlace PCIe no modifica la cota de escritura, ya que $V_{\text{discos}}^{\text{esc}}$ es inferior a B_{PCIe} e inferior a $B_{\text{PCIe}}/2 = 2000$ MB/s en todos los nodos. Por lo tanto, el rendimiento máximo de escritura permanece limitado por los discos en todas las configuraciones. La Tabla 4.6 presenta las cotas teóricas por OSS ajustadas por el esquema de espejado.

nodo	R_{OST}^{lect} *(MiB/s)*	R_{OST}^{esc} *(MiB/s)*
agnetha	987,0	491,1
annifrid	972,7	486,4
benny	801,1	400,5
bjorn	801,1	400,5
Total del sistema	3562,0	1778,5

Tabla 4.6: Cotas teóricas por OSS ajustadas.

Capítulo 5

Diseño experimental

En este capítulo se presenta el diseño experimental empleado para evaluar el sistema de archivos paralelo desplegado. En primer lugar se justifica la elección del software utilizado en los experimentos y se describe cada uno de los benchmarks empleados junto con los experimentos definidos. A continuación se presenta la metodología utilizada para caracterizar el comportamiento de E/S de las aplicaciones y se describen las aplicaciones utilizadas. Por último, se explican las configuraciones del sistema de archivos evaluadas en las ejecuciones de las aplicaciones.

5.1. Software elegido

La evaluación del sistema de archivos paralelo se organizó en dos etapas. En la primera etapa se utilizaron tres benchmarks para evaluar, bajo patrones de acceso controlados y reproducibles, el rendimiento de las distintas partes del sistema.

El benchmark Flexible I/O Tester (fio) (Axboe, 2025) se utilizó para evaluar el rendimiento de los dispositivos de almacenamiento desde el mismo servidor que los sirvió; el benchmark Interleaved or Random (IOR) (IOR developers, 2026) se utilizó con el sistema de archivos configurado y montado en los clientes para evaluar el rendimiento del sistema de archivos en su totalidad; y el benchmark MDTest (MDTest project, 2010) se utilizó para evaluar el rendimiento del subsistema de metadatos del sistema de archivos paralelo. En la segunda etapa, para complementar la evaluación basada en benchmarks y analizar el comportamiento del sistema de archivos ante cargas representativas, se utiliza-

ron tres aplicaciones reales de computación de alto desempeño: dos aplicaciones que imponen una carga intensiva en lectura y escritura de datos sobre el sistema de archivos, una de procesamiento de datos para una red de aprendizaje profundo DeepONet y otra de procesamiento de datos de transporte público uruguayo y una aplicación intensiva en cómputo, de calibración de un modelo agronómico. Los benchmarks y los experimentos definidos para cada uno se describen en la Sección 5.2, las aplicaciones se describen en la Sección 5.3 y la metodología empleada para caracterizar su comportamiento de entrada y salida (E/S) se describe en la Sección 5.4.

5.2. Benchmarks

Para evaluar el rendimiento del sistema de almacenamiento desplegado se utilizaron tres herramientas de benchmark complementarias: fio, IOR y MDTest. Los benchmarks fio e IOR caracterizan el rendimiento del subsistema de datos y el benchmark MDTest caracteriza el rendimiento del subsistema de metadatos. En esta sección se describen las cargas de trabajo y la metodología experimental definidas para cada herramienta.

5.2.1. Flexible I/O Tester

fio es una herramienta de medición de rendimiento de E/S que ejecuta en un único nodo y permite caracterizar por separado cada capa de la arquitectura (el dispositivo de bloques en crudo, el sistema de archivos de base del OST montado localmente y el rendimiento del OSS individual en Lustre desde un cliente). La herramienta permite definir cargas de trabajo de forma declarativa mediante archivos de trabajo, en los que se especifican parámetros como el patrón de acceso (secuencial o aleatorio, lectura o escritura), el tamaño de bloque, la profundidad de cola de pedidos de E/S (número máximo de operaciones de E/S que fio mantiene pendientes en simultáneo), el número de procesos concurrentes y el volumen total de datos a transferir. A partir de la especificación del trabajo, fio sintetiza el patrón de E/S correspondiente, ejecuta el trabajo y reporta métricas de ancho de banda, operaciones de E/S por segundo y latencia.

Para analizar el rendimiento de los servidores de datos bajo una carga secuencial se definieron dos cargas de trabajo secuenciales, una de escritura y otra de lectura (ejecutadas en ese orden) con tamaño de bloque de 1 MiB, profundidad de cola de E/S de 32, un único proceso y un volumen total de 128 GiB por trabajo. El volumen se eligió por ser igual o mayor al total de memoria RAM disponible en los OSS, para eliminar posibles interferencias de la caché de páginas (caché en memoria RAM utilizada para almacenar datos de disco recientemente accedidos) en la fase de lectura. El primer trabajo realiza una escritura secuencial y el segundo trabajo realiza la lectura secuencial análoga.

Se definieron tres familias de experimentos para las cargas de trabajo: la primera, llamada rawdev, evalúa el rendimiento por OST directamente sobre el dispositivo de bloques en crudo, sin sistema de archivos intermedio. rawdev sirve para obtener una cota superior del rendimiento del disco lógico del OST independientemente de Lustre y es el valor de base con el que se comparan el resto de los experimentos. La segunda, llamada localmount, evalúa el rendimiento de los sistemas de archivos de los OSS localmente; cada OSS monta su OST de manera local y los experimentos se realizan sobre el montaje; los datos no atraviesan la red, por lo que los experimentos de esta familia miden el overhead agregado por el sistema de archivos de base del OST al dispositivo en crudo. La tercera, llamada client, evalúa el rendimiento por OSS en un sistema de archivos Lustre montado en un cliente; los datos atraviesan la red y utilizan el sistema de archivos Lustre configurado para que todas las partes de los archivos del experimento se almacenen en el OSS específico que se deseaba evaluar.

Se ejecutaron tres repeticiones con las cargas de trabajo descritas. Para la familia client, los experimentos se ejecutaron en los clientes acuario u orion. Los experimentos de las familias localmount y client se ejecutaron sobre los tres sistemas de archivos de base evaluados: ldiskfs, ZFS sin compresión y ZFS con compresión. El benchmark se configuró para registrar E/S almacenada en caché y E/S directa y se reportan solo los resultados con E/S directa. La E/S directa evita la caché de páginas del núcleo, por lo que evalúa el rendimiento del sistema de archivos sin intervención de la caché de páginas del sistema operativo. Como pueden existir otras cachés aparte de la caché de páginas, para eliminar su posible interferencia en los resultados, previo a cada experimento en el cliente y los OSS se realizó un vaciado de las cachés del sistema, luego se realizó la fase de escritura, luego se eliminaron las cachés nuevamente en el

cliente y los OSS y luego se realizó la fase de lectura. Además, el volumen de datos escrito, excepto en un caso, fue igual o mayor a la cantidad de memoria RAM disponible en los OSS. Para los experimentos con ZFS, se deshabilitó la caché ARC de datos. Con este procedimiento, la lectura no se beneficia de potencialmente tener en caché los datos escritos recientemente.

Los experimentos de la familia rawdev se ejecutaron sobre el arreglo RAID 1+0 (por hardware en agnetha, annifrid y bjorn, por software en benny) sin sistema de archivos encima, con el objetivo de evaluar el rendimiento del dispositivo lógico que utilizaron los sistemas de archivos de base. Analizar los discos por separado y sumar su rendimiento daría un valor inalcanzable por cualquier configuración; todas las familias escriben sobre discos con redundancia, por lo que la cota superior se toma siempre sobre el arreglo completo. Para el sistema de archivos de base ldiskfs el sistema de archivos se despliega sobre el mismo arreglo RAID 1+0 cuyo rendimiento fue analizado por la familia de experimentos rawdev, por lo que el resultado de rawdev es una cota superior para ldiskfs y las diferencias con localmount son atribuibles al overhead del sistema de archivos de base. Para el sistema de archivos de base ZFS, el resultado de rawdev no siempre es una cota superior. En annifrid y bjorn, por limitaciones de la controladora, el pool ZFS se construyó sobre el mismo arreglo RAID 1+0 configurado como un único vdev, por lo que la cota superior continúa siendo el resultado de rawdev. En agnetha y benny, nodos con acceso individual a los discos (en modo pass-through o como RAID 0 individuales por disco) ZFS gestiona la redundancia internamente y los discos son configurados como vdevs espejados. En agnetha y benny no hay un dispositivo de bloques equivalente al dispositivo lógico ZFS que pueda medirse en crudo, pero un pool de vdevs espejados es funcionalmente equivalente a un RAID 1+0, opera sobre los mismos discos físicos y la capacidad útil es la misma, por lo que el resultado de rawdev es el valor de rendimiento crudo de referencia más cercano.

En todas las familias cada experimento operó sobre 128 GiB de datos. La excepción fueron los experimentos ZFS y ZFS con compresión de la familia client en los que, debido a las bajas tasas de escritura y para evitar tiempos excesivos de ejecución, el volumen de datos se redujo a 10 GiB. Se realizaron tres repeticiones para cada experimento y se reportan la media y desviación estándar muestral.

5.2.2. Interleaved or Random

IOR es una herramienta de benchmark utilizada para evaluar el rendimiento de sistemas de archivos (IOR developers, 2026). Su función principal es medir el ancho de banda de escritura y lectura sostenido que un sistema de almacenamiento puede entregar bajo distintas condiciones de acceso concurrente. La coordinación mediante Message Passing Interface (MPI) de múltiples procesos que operan simultáneamente sobre el sistema de archivos simula la carga de trabajo de aplicaciones paralelas. Cada proceso MPI realiza operaciones de lectura y escritura sobre bloques de datos de tamaño configurable, sobre archivos independientes o sobre un único archivo compartido. IOR repite cada experimento un número definido de veces y reporta la velocidad de lectura y escritura promedio y la desviación estándar muestral.

Para evaluar el sistema de archivos paralelo desplegado se definieron experimentos que combinaron distintos valores de determinados parámetros, con el objetivo de caracterizar el comportamiento del sistema de archivos Lustre ante cambios en el número de procesos concurrentes accediendo al sistema de archivos, el modo de acceso, la configuración de stripe, el número de clientes y el sistema de archivos de base. Cada parámetro afecta de manera distinta el rendimiento del sistema de archivos. Para poder aislar y cuantificar el efecto individual de cada parámetro, se adoptó una estrategia de análisis de un factor a la vez: en cada experimento se varió un único parámetro y los demás se mantuvieron fijos en valores previamente definidos. En todos los experimentos se utilizó un volumen de datos total de 160 GiB (repartidos equitativamente entre los procesos), un tamaño de transferencia de 1 MiB, se reorganizó la asignación de archivos entre las fases de escritura y lectura para evitar que cada proceso leyera el archivo que había escrito, se forzó la escritura de los datos al almacenamiento persistente en la fase de escritura y se ejecutaron tres repeticiones de cada experimento. Previo a la ejecución de cada experimento se limpiaron las cachés de los clientes, OSS y MDS mediante el comando `echo 3 > /proc/sys/vm/drop_caches`, luego se realizó la fase de escritura, luego se limpiaron las cachés de los clientes, OSS y MDS con el mismo comando y luego se realizó la fase de lectura. En ZFS se deshabilitó la caché ARC (caché de lectura en RAM) para datos. De esta manera, los resultados no se ven afectados por las cachés del sistema operativo.

Los parámetros evaluados fueron:

- *Efecto del número de OST utilizados para striping.* Se varió el número de OST sobre los que se distribuyó cada archivo, con configuraciones de uno, dos, tres y cuatro OST. En la configuración de un OST cada archivo se almacena íntegramente en alguno de los OST del sistema y en las configuraciones restantes los datos de cada archivo se distribuyen entre la cantidad correspondiente de OST. Esta comparación permite determinar en qué medida el paralelismo adicional que ofrece la distribución entre un número creciente de OST se traduce en una mejora efectiva del rendimiento respecto a utilizar un único OST por archivo.
- *Número de procesos.* Representa la cantidad de procesos MPI que participan en la operación de lectura y escritura de manera concurrente. A mayor número de procesos, mayor es la concurrencia simultánea sobre el sistema de archivos.
- *Modo de acceso.* Se evaluaron dos modos de acceso a los archivos: FPP, donde cada proceso MPI lee y escribe en un archivo propio y SSF, donde todos los procesos MPI leen y escriben sobre un único archivo compartido.
- *Configuración de stripe.* Lustre distribuye los datos de un archivo entre uno o más servidores de objetos en bloques de tamaño fijo denominados stripes. La configuración de stripe tiene dos dimensiones: el número de OST sobre los que se distribuye el archivo (stripe count) y el tamaño de cada bloque (stripe size). En los experimentos realizados se evaluaron cinco combinaciones: almacenar cada archivo en un solo OST (puede ser cualquiera de los OST presentes en el sistema de archivos, el OST elegido varía entre archivos) y repartir los datos de los archivos entre cuatro OST con un tamaño de stripe de 256 KiB, 1 MiB, 4 MiB y 16 MiB.
- *Número de clientes.* Indica cuántos nodos cliente participan en las operaciones de lectura y escritura. Se evaluó el uso de uno, dos y tres clientes. Por limitaciones de hardware (equipos y puertos del switch de datos de 10 GbE) no se pudieron evaluar escenarios con más clientes. Los experimentos con un cliente se realizaron en acuario, los experimentos con dos clientes se realizaron en acuario y centauro y los experimentos con tres clientes se realizaron en acuario, centauro y orion.

- *Sistema de archivos de base.* Lustre permite utilizar distintos sistemas de archivos de base en los OST. Se evaluaron tres configuraciones: ldiskfs, ZFS sin compresión y ZFS con compresión LZ4. Esta comparación permite determinar en qué medida las funcionalidades adicionales de ZFS, con y sin compresión, impactan el rendimiento respecto a ldiskfs.

Se definió una condición de referencia que actuó como punto fijo en todos los experimentos: en cada uno se mantuvieron fijos los valores de referencia y se varió únicamente el parámetro que se desea evaluar. Los valores de referencia seleccionados fueron doce procesos concurrentes, dos clientes, striping en cuatro OST con tamaño de stripe de 4 MiB y modo de acceso file per process.

Se eligieron doce procesos concurrentes por ser representativos de una carga de trabajo en un sistema de archivos compartido, en el que múltiples usuarios o procesos acceden concurrentemente al sistema de archivos desde más de un cliente físico. Además, se eligió utilizar dos clientes como referencia, dado que la memoria caché de páginas del sistema operativo del nodo cliente puede interferir en las mediciones de lectura y en consecuencia se podrían generar resultados que no reflejen el rendimiento real del almacenamiento. Con dos o más clientes y los procesos MPI correctamente configurados, el efecto de la caché de páginas (en los clientes) no es un problema, ya que un proceso en un cliente lee datos que fueron escritos por otro cliente y que por lo tanto no están en su memoria caché.

Para eliminar el efecto de la caché de páginas en los OSS y MDS, se realizó la limpieza de cachés descrita anteriormente previo a la etapa de escritura y previo a la etapa de lectura. Se eligió utilizar la configuración de striping entre cuatro OST con stripe size de 4 MiB para lograr un balance entre paralelismo de E/S y overhead de coordinación, sin concentrar la carga en un único OST ni fragmentar excesivamente los accesos. Se eligió el modo de acceso file per process por ser el modo de acceso más simple respecto a la ejecución de múltiples procesos concurrentes y por presentar un comportamiento más predecible a medida que se escala. Se eligió escalamiento fuerte para medir la misma carga total de trabajo a través de las ejecuciones, independientemente de que el número de procesos varíe.

5.2.3. MDTest

MDTest es una herramienta de benchmark diseñada para medir el rendimiento de las operaciones de metadatos en sistemas de archivos paralelos (MDTest project, 2010). La herramienta utiliza la interfaz MPI para coordinar múltiples procesos que operan concurrentemente sobre el sistema de archivos. En cada iteración, cada proceso MPI crea, consulta atributos y elimina un número definido de archivos o directorios. Para cada operación, MDTest reporta las tasas de operaciones realizadas por segundo y los valores máximo, mínimo, promedio y desviación estándar muestral de las iteraciones ejecutadas.

MDTest se utilizó para evaluar el rendimiento del subsistema de metadatos del sistema de archivos paralelo desplegado. Se definieron cuatro escenarios experimentales y cada uno se ejecutó sobre los tres sistemas de archivos de base evaluados para el MDT: ldiskfs, ZFS sin compresión y ZFS con compresión. Los experimentos de un cliente se realizaron en acuario y los experimentos de dos clientes se realizaron en acuario y orion. Los escenarios definidos fueron:

- *Escalamiento débil con directorios únicos.* Cada proceso opera sobre su propio directorio, con 20 000 archivos por proceso. La carga total crece proporcionalmente con el número de procesos.
- *Escalamiento débil con directorio compartido.* Todos los procesos operan sobre un único directorio compartido, con 20 000 archivos por proceso. El uso de un directorio compartido introduce mayor contención sobre las estructuras de metadatos dada la modificación concurrente.
- *Escalamiento fuerte con directorios únicos.* El número total de archivos se mantiene fijo en 1 000 000 y el número de archivos se reparte equitativamente entre los procesos. A medida que aumenta el número de procesos se reduce la carga por proceso, pero aumenta la concurrencia sobre el servidor de metadatos.
- *Experimento de estrés.* Se operó sobre 9 600 000 archivos en total, repartidos entre 32 procesos sobre un único directorio. El objetivo del experimento fue someter al servidor de metadatos a una carga elevada.

5.3. Aplicaciones

Los benchmarks descritos en la Sección 5.2 permiten evaluar el rendimiento del sistema de archivos bajo patrones de acceso controlados y reproducibles pero no reflejan el comportamiento de E/S de aplicaciones reales, que combinan lectura y escritura, alternan fases de cómputo y de entrada y salida y acceden a conjuntos de archivos cuya cantidad, tamaño y distribución difieren de los patrones sintéticos. Por lo tanto, se eligieron tres aplicaciones reales para estudiar el comportamiento del sistema de archivos paralelo frente a la carga de E/S que generan. En esta sección se describe el funcionamiento de cada aplicación, la ponderación por heterogeneidad de rendimiento de los OST aplicada a los tiempos resultantes y se definen las métricas de aceleración y eficiencia. Las aplicaciones se ejecutaron indistintamente en los clientes acuario u orion, ya que poseen las mismas características de hardware.

5.3.1. Procesamiento de datos para DeepONet

La primera aplicación de consumo masivo de datos estudiada se encarga de transformar las salidas de simulaciones de dinámica de fluidos computacional en un conjunto de datos estructurado para el entrenamiento del modelo subrogado DeepONet, destinado a la detección de daño por heladas agrometeorológicas. Un modelo subrogado es una aproximación de bajo costo computacional que, una vez entrenada con datos de una simulación de alta fidelidad, permite predecir los resultados de una simulación de forma rápida. Las simulaciones de origen modelan el enfriamiento nocturno sobre distintas zonas rurales y cada caso simulado corresponde a una configuración geográfica particular que se almacena en un directorio independiente. Para cada caso, la simulación produce un conjunto de archivos en un formato binario comprimido. Los archivos comprimidos contienen la información geométrica del terreno y los valores de las variables físicas calculadas en cada instante de tiempo; la aplicación transforma el conjunto de archivos comprimidos de entrada en un formato tabular uniforme, más simple de manipular en las etapas posteriores.

Para realizar la conversión, la aplicación descomprime cada archivo en almacenamiento temporal, interpreta su estructura interna y extrae las variables de interés: por un lado los datos geométricos de la malla de simulación y, por otro, las variables físicas relevantes, principalmente la velocidad, la presión y la temperatura. Cada caso da lugar a un único archivo de salida en formato .mat,

que agrupa de forma separada los datos geométricos y los datos físicos. Una vez generados los archivos .mat, se aplica una etapa de reducción de dimensionalidad, que conserva únicamente las capas más cercanas al nivel del suelo, las relevantes para evaluar el riesgo de helada sobre los cultivos. Finalmente, una etapa de análisis exploratorio extrae estadísticas descriptivas de cada archivo y las consolida en archivos tabulares, que sirven de base para el estudio de correlaciones entre variables, la caracterización de los casos y la partición del conjunto de datos en los subconjuntos de entrenamiento, validación y prueba utilizados por el DeepONet.

El procesamiento se ejecuta de forma paralela. La aplicación recorre el conjunto de casos, descarta los que figuran en una lista de exclusión y construye una lista de tareas que luego se procesan. Cada tarea genera un archivo de salida a partir de un caso. Las tareas se reparten entre un conjunto de procesos trabajadores que se ejecutan de forma independiente, cada uno encargado de procesar las tareas asignadas y escribir sus propios archivos de salida. El esquema incorpora una política de reintentos ante fallos transitorios de lectura o escritura. El proceso paralelo de conversión es el que se estudia en este proyecto. Para los experimentos se utilizaron dieciséis procesos trabajadores concurrentes.

5.3.2. Procesamiento de datos de transporte público

La segunda aplicación de consumo masivo de datos estudiada procesa los registros de viajes del sistema de transporte público urbano de Montevideo, con el objetivo de reorganizar los registros para que todos los viajes realizados con una misma tarjeta de transporte queden agrupados. Cada registro corresponde a un evento de viaje asociado a una tarjeta de transporte y contiene, además del identificador de la tarjeta, información sobre la fecha, la línea, la empresa, el recorrido y otros atributos del viaje. Los registros de entrada se encuentran almacenados en un único archivo de texto en formato CSV, en el que los viajes de una misma tarjeta aparecen dispersos a lo largo de todo el archivo. El reordenamiento se realiza en dos etapas que ejecutan de manera secuencial: una de particionado y otra de ordenamiento.

La etapa de particionado recorre el archivo de entrada y distribuye los registros en cien archivos de partición. El archivo de destino de cada registro se determina a partir del número de tarjeta con el cálculo del resto de su división

entre cien. Todos los viajes de una misma tarjeta quedan asignados a la misma partición y el conjunto de tarjetas se reparte entre los cien archivos. Los registros que no tienen una tarjeta asociada se descartan. La etapa de ordenamiento procesa cada partición de forma independiente. Para cada partición se construye un índice que asocia a cada tarjeta las posiciones de todos sus registros dentro del archivo; luego se recorre el índice y, para cada tarjeta, se leen sus registros saltando a las posiciones indicadas y se los reescribe de forma contigua. El resultado es un conjunto de cien archivos de salida en los que los registros aparecen agrupados por tarjeta.

El procesamiento de ambas etapas se ejecuta, dentro de cada una, de forma paralela mediante un conjunto de procesos trabajadores independientes. En el particionado, el archivo de entrada se divide en tantos fragmentos contiguos como procesos y cada trabajador recorre su fragmento y acumula los registros en tandas que se vuelcan sobre los archivos de partición. Como varios trabajadores pueden escribir sobre un mismo archivo de salida, el acceso a cada uno se coordina mediante un mecanismo de exclusión mutua. En la etapa de ordenamiento, las cien particiones se reparten entre los procesos trabajadores y cada uno construye los índices y genera los archivos de salida de las particiones que le fueron asignadas. El procesamiento de ambas etapas es el que se estudia en este proyecto. Para los experimentos se utilizaron dieciséis procesos trabajadores concurrentes.

5.3.3. Calibración del modelo DayCent

La tercera aplicación estudiada realiza la calibración paramétrica de DayCent, un modelo de simulación de la dinámica de carbono en el suelo, utilizado en un estudio agronómico que compara distintos sistemas de manejo agrícola en relación a su efecto sobre la reserva de carbono del suelo. La aplicación invoca el programa DayCent como un ejecutable y se comunica con él a través del sistema de archivos. Calibrar el modelo consiste en buscar, entre varias combinaciones posibles de sus parámetros, las que producen simulaciones más próximas a un conjunto de observaciones de campo. A diferencia de las dos aplicaciones anteriores, su costo está dominado por el cómputo de las sucesivas ejecuciones del modelo más que por el volumen de datos procesado. El ejecutable de DayCent es un binario de Windows, por lo que se ejecutó en Linux mediante el programa Wine.

Los parámetros del modelo se almacenan en archivos de texto de configuración. Una primera etapa genera, mediante un muestreo por hipercubo latino sobre los rangos predefinidos de cada parámetro, una matriz de calibración en la que cada fila representa una combinación de valores. A continuación, la etapa iterativa procesa cada combinación mediante la misma secuencia de pasos: se reescriben los archivos de parámetros con los valores correspondientes a la fila y se ejecuta el modelo, que produce un archivo binario de salida. Luego, un segundo ejecutable traduce el archivo binario a un archivo de texto legible, que se lee para calcular una medida de ajuste entre la simulación y las observaciones y se persiste únicamente la medida junto con el vector de parámetros empleado. Antes de pasar a la siguiente realización, los archivos de salida generados en la iteración (el binario y su traducción textual) se eliminan. En cada iteración se escribe y luego se elimina una cantidad de datos del orden de una decena de megabytes por lo que, aunque el tamaño de los datos residentes en disco es reducido, el proceso genera un flujo continuo de escrituras, lecturas y borrados de archivos temporales. El número de iteraciones es un parámetro configurable, que en este proyecto se fijó en 3200.

El procesamiento se ejecuta de forma paralela mediante un conjunto de procesos trabajadores independientes. Antes de iniciar la ejecución, el directorio de instalación del modelo se replica una vez por cada proceso trabajador, por lo que cada uno opera sobre su propia copia y no existe competencia por la escritura sobre los archivos del modelo. La matriz de calibración se divide en tantos fragmentos contiguos como procesos trabajadores y cada fragmento se asigna a un proceso. El proceso al que se asigna el fragmento recorre sus combinaciones aplicando de forma independiente la secuencia de reescritura de parámetros, ejecución del modelo, lectura de la salida, cálculo del ajuste y eliminación de los archivos transitorios. Al finalizar, las tablas de resultados parciales producidas por los procesos trabajadores se concatenan en orden en una única tabla de salida y se eliminan las copias del directorio del modelo. El proceso paralelo de calibración descrito es el que se estudia en este proyecto. Para los experimentos se utilizaron veinte procesos trabajadores concurrentes.

5.3.4. Ponderación por heterogeneidad de los OST

Los cuatro OST empleados no ofrecieron el mismo rendimiento de lectura y escritura entre sí, por lo que el tiempo de una ejecución paralela no dependió solo del grado de paralelismo sino también de cuáles OST participaran en ella. Una configuración que involucre OST lentos queda penalizada frente a otra que utilice los más veloces y ambas quedan a su vez penalizadas o favorecidas frente a una ejecución que utilice un único nodo concreto. Comparar directamente los tiempos crudos mezclaría el efecto del paralelismo con el efecto de la heterogeneidad del hardware, de modo que, para aislar el efecto del paralelismo, se introdujo una ponderación que normaliza los tiempos paralelos según la velocidad relativa de los OST involucrados.

A partir de los resultados de la Sección 6.1, se obtuvo el ancho de banda medio de lectura y escritura para cada OST al promediar los tres sistemas de archivos de base medidos para los experimentos de la familia localmount. Los promedios se normalizaron de forma independiente al dividir por el valor mínimo de cada operación, por lo que el OST más lento en cada caso recibió una razón de velocidad relativa de 1 y el resto un valor proporcional a cuántas veces más rápido es; formalmente, para el OST i se define $\rho_i^{\mathrm{r}} = \bar{v}_i^{\mathrm{r}} / \min_k \bar{v}_k^{\mathrm{r}}$ y $\rho_i^{\mathrm{w}} = \bar{v}_i^{\mathrm{w}} / \min_k \bar{v}_k^{\mathrm{w}}$. $\bar{v}_i^{\mathrm{r}}$ y $\bar{v}_i^{\mathrm{w}}$ son los anchos de banda medios de lectura y escritura del OST i. La Tabla 5.1 presenta los valores obtenidos.

OST	*razón*	
	ρ^{r}	ρ^{w}
agnetha	2,01	3,29
annifrid	1,57	1,00
benny	1,17	2,29
bjorn	1,00	1,34

Tabla 5.1: Razones de rendimiento de cada OST normalizadas respecto al OST más lento.

Se definió el costo ponderado de ejecutar la carga completa sobre un único OST i como la suma del trabajo de lectura y de escritura física, medido según la metodología de la Sección 5.4.2, cada uno dividido por la razón correspondiente del OST: $C_i = n_{\mathrm{r}}/\rho_i^{\mathrm{r}} + n_{\mathrm{w}}/\rho_i^{\mathrm{w}}$. Para una configuración paralela que reparte la carga sobre un conjunto $\mathcal{O}$ de N OST se asume a la larga un reparto uniforme, por lo que cada OST atiende $\lceil n_{\mathrm{r}}/N \rceil$ lecturas y $\lceil n_{\mathrm{w}}/N \rceil$ escrituras y

el costo ponderado agregado resulta $C_{\mathcal{O}} = \sum_{i \in \mathcal{O}} (\lceil n_{\mathrm{r}}/N \rceil / \rho_i^{\mathrm{r}} + (\lceil n_{\mathrm{w}}/N \rceil)/\rho_i^{\mathrm{w}})$. El costo ponderado es una medida del trabajo total de E/S de la configuración, expresado en unidades del OST más lento de cada operación. Ante OST homogéneos se cumple $C_{\mathcal{O}} = C_i = n_{\mathrm{r}} + n_{\mathrm{w}}$, por lo que la ponderación es neutra y solo introduce correcciones en presencia de heterogeneidad.

El tiempo de ejecución paralelo T_p se normalizó al costo de una ejecución de la carga completa sobre un OST de referencia r mediante el factor $f(r) = C_r/C_{\mathcal{O}}$, donde C_r es el costo ponderado de ejecutar la carga completa sobre el nodo r, es decir C_i evaluado en $i = r$. La normalización resultó en un tiempo ponderado $T_p^\star(r) = f(r)\, T_p = (C_r/C_{\mathcal{O}})\, T_p$. $T_p^\star(r)$ es el tiempo que la configuración emplearía en completar la misma cantidad de trabajo ponderado que una ejecución de la carga completa sobre el nodo r, por lo que ambas se comparan sobre una base de trabajo equivalente y se elimina la ventaja o desventaja introducida por la elección de OST. La elección de r fija el nodo respecto al cual se expresa el tiempo ponderado.

5.3.5. Aceleración y eficiencia

Para cuantificar el efecto del paralelismo de almacenamiento al escalar el número de OST se calcularon la aceleración y la eficiencia para cada aplicación evaluada respecto a una configuración de referencia con un único OST. El análisis se restringió, en cada aplicación evaluada, a la serie de configuraciones de uno, dos, tres y cuatro OST sobre el sistema de archivos de base ldiskfs y a la configuración de cuatro OST que obtuvo el mejor tiempo de ejecución entre los sistemas de archivos de base evaluados. La aceleración se definió como $S = T_{1\,\mathrm{OST}}^\star / T_p^\star$ y la eficiencia como $E = S/N$. $T^\star$ es el tiempo ponderado por heterogeneidad de OST de cada configuración, calculado como se explica en la Sección 5.3.4, $T_p^\star$ el tiempo ponderado de la configuración con N OST y $T_{1\,\mathrm{OST}}^\star$ el tiempo ponderado de la configuración base con un único OST.

La base utilizada para calcular ambas métricas fue, en todos los casos, el tiempo de la configuración de un OST. Dado que la ponderación de la configuración con un OST se realiza contra sí misma, el costo de referencia empleado en la ponderación coincidió con el costo del único OST utilizado en las ejecuciones de un OST, por lo que el tiempo ponderado de la base resultó igual a su tiempo sin ponderar. La configuración de base no corresponde a una ejecución secuencial de la aplicación, sino a su ejecución sobre el sistema

de archivos paralelo utilizando un único OST y un único MDT. Entonces, la aceleración y la eficiencia calculadas no cuantifican la ganancia respecto de una ejecución no paralela, sino la ganancia relativa obtenida al escalar el almacenamiento del sistema de archivos paralelo de 1 a N OST.

5.4. Caracterización de E/S y uso de recursos

Para evaluar el desempeño de las aplicaciones sobre Lustre se combinaron dos tipos de medición complementarios. Por un lado, se caracterizó el patrón de E/S de las aplicaciones a nivel de llamadas al sistema, de forma independiente del sistema de archivos de base. Por otro, durante una ejecución de cada configuración del sistema de archivos se monitoreó la utilización de los recursos de disco y de CPU en los servidores del sistema de archivos paralelo y en el cliente. La caracterización de E/S describe qué hace la aplicación en E/S; el monitoreo de recursos, cómo responde la infraestructura ante la carga de E/S.

5.4.1. Caracterización de E/S de las aplicaciones

Realizar la caracterización de E/S de una aplicación consiste en medir el volumen lógico de datos leídos y escritos, el tamaño de las operaciones de E/S, la proporción entre lectura y escritura, la cantidad de archivos sobre los que se opera y las operaciones sobre metadatos. Al ser mediciones lógicas, la caracterización de cada aplicación depende únicamente de la lógica de cada programa y no del sistema de archivos utilizado. Como paso previo a la ejecución de las aplicaciones, se caracterizó su comportamiento de E/S.

Para realizar la caracterización se utilizó la herramienta bpftrace (bpftrace developers, 2026), un programa de rastreo dinámico basado en extended Berkeley Packet Filter (eBPF), una tecnología del núcleo Linux que permite ejecutar programas en un entorno aislado y controlado. Las métricas se obtuvieron interceptando las llamadas al sistema de E/S open, openat, dup, dup2, dup3, read, write, unlink, unlinkat y close sobre el punto de montaje del sistema de archivos, por lo que toda operación de E/S se registró independientemente del lenguaje, biblioteca o capa de abstracción que la originara. Además, la medición no interfiere con la lógica del programa a medir y no requiere modificar ni recompilar las aplicaciones.

Para la caracterización de las aplicaciones se desarrolló un script que midió la cantidad de aperturas exitosas sobre el punto de montaje del sistema de archivos paralelo, cuántos descriptores de archivo se utilizaron efectivamente para leer o escribir, el volumen de bytes lógicos leídos y escritos desglosado por archivo y la cantidad de archivos únicos leídos y escritos. La medición se ejecutó junto a una ejecución de cada aplicación y se incluyeron los hilos de procesamiento de datos en la medición en caso de que la aplicación creara hilos.

5.4.2. Utilización de recursos durante la ejecución

Durante una ejecución de cada aplicación se midió la utilización de los recursos disco y CPU en el MDS, los OSS y el cliente. El objetivo fue analizar la respuesta del subsistema de almacenamiento ante la carga generada, contrastar el comportamiento de los dos sistemas de archivos de base de los OST evaluados e identificar posibles cuellos de botella. La medición en los servidores se realizó con un script que toma muestras una vez por segundo y registra, para los dispositivos físicos que conforman cada MDT u OST, métricas de bloque obtenidas con la herramienta iostat (datos transferidos, tamaño medio de solicitud, latencias y utilización del dispositivo) y los contadores crudos del subsistema de bloques expuestos en /sys/block. Para los OSS también se registró su utilización de CPU, el porcentaje de uso en modo usuario, en modo kernel, en espera de E/S y ocioso. En el caso del sistema de archivos de base ZFS, se agregaron además los datos transferidos por pool y por vdev mediante la herramienta zpool.

La medición se orquestó desde el cliente, que lanzó el script de monitoreo en todos los servidores de forma sincronizada con la ejecución de la aplicación. En el cliente se registró además la utilización de CPU y el tiempo de ejecución de la aplicación. Al inicio y al final de cada ejecución se registraron marcas de tiempo que delimitaron la ventana de medición y permitieron alinear temporalmente las series de los distintos nodos.

5.5. Configuraciones de Lustre evaluadas en las aplicaciones

Los experimentos para evaluar el sistema de archivos paralelo mediante el uso de aplicaciones se organizaron en dos etapas. Primero se realizaron ejecuciones sin sistema de archivos paralelo con el sistema de archivos distribuido NFS. Luego, dentro del sistema de archivos Lustre, se evaluó el rendimiento de las aplicaciones a través de una serie de configuraciones en las que se varió un parámetro por vez para aislar el efecto en el rendimiento de cada uno. La métrica utilizada para medir el rendimiento de las aplicaciones fue el tiempo de ejecución total. Se realizaron tres ejecuciones de cada caso.

Para las ejecuciones sin sistema de archivos paralelo se evaluaron dos configuraciones de un solo nodo, con los dispositivos de bloque locales del OSS compartidos con el cliente a través de NFS. Como Lustre confirma las escrituras de manera asíncrona, los servidores NFS se exportaron con la opción async para replicar el comportamiento asíncrono. A partir de los resultados del benchmark fio (Sección 6.1), se seleccionó el OSS de cuatro discos de mejor rendimiento y el OSS de seis discos de peor rendimiento para cubrir dos puntos de operación representativos del rango de rendimiento de los OSS. Por lo tanto, las ejecuciones de referencia de cuatro discos se ejecutaron sobre el nodo benny y las ejecuciones de referencia de seis discos se ejecutaron sobre el nodo annifrid.

Las ejecuciones sin sistema de archivos paralelo miden el tiempo de ejecución de la aplicación sin el paralelismo introducido por Lustre. Las configuraciones sin sistema de archivos paralelo se evaluaron con los sistemas de archivos de base ext4 (ldiskfs está basado en ext4) y ZFS con compresión LZ4. Al igual que en la Sección 5.2.2, para la evaluación con el sistema de archivos paralelo se partió de una condición de referencia (striping entre cuatro OST con tamaño de stripe de 4 MiB y sistema de archivos de base ldiskfs) y se varió un parámetro específico por caso para evaluar su impacto en el rendimiento de la aplicación. Los parámetros que se variaron fueron:

- *Cantidad de OST*. Para evaluar la contribución al rendimiento de agregar OST al sistema de archivos se utilizaron cuatro configuraciones de Lustre con uno, dos, tres o cuatro OST activos. El conjunto de OST se hizo crecer de forma incremental para mantener las configuraciones

comparables. La configuración de un OST utilizó el OST de annifrid; la configuración de dos OST agregó el OST de agnetha; la configuración de tres OST agregó el OST de benny; y la configuración de cuatro OST incorporó el OST de bjorn. Los stripes de los archivos se distribuyeron equitativamente entre los OST participantes.

- *Sistema de archivos de base.* La configuración con cuatro OST se evaluó también con los sistemas de archivos de base ZFS y ZFS con compresión LZ4.
- *Cantidad de stripes.* Sobre la configuración con cuatro OST se fijó la cantidad de stripes por archivo en 1, 2 y 3, para que cada archivo se divida en stripes distribuidos entre uno, dos o tres OST. Los OST asignados a los stripes pueden variar de un archivo a otro. La configuración de cuatro OST con ldiskfs corresponde a un stripe count de 4.
- *Progressive File Layout (PFL).* Se evaluó una distribución de datos progresiva en la que los primeros 256 MiB de cada archivo se almacenaron en un único OST, los bytes entre 256 MiB y 1 GiB se distribuyeron entre dos OST y el resto se distribuyó entre los cuatro OST. PFL se evaluó sobre los sistemas de archivos de base ldiskfs, ZFS y ZFS+LZ4.
- *RPC de gran tamaño (large I/O).* El cliente de Lustre se configuró para emitir RPC de 16 MiB en lugar de los 4 MiB por defecto.
- *Data on MDT (DoM).* Se evaluó una distribución de datos que combina DoM y PFL: el primer MiB de cada archivo se almacenó en el MDT (almacenado en un SSD) y el resto utilizó la misma distribución de datos progresiva de la configuración PFL.

Capítulo 6

Evaluación experimental con benchmarks

En este capítulo se presentan y analizan los resultados de los experimentos de evaluación del rendimiento del sistema de archivos paralelo a través de la ejecución de benchmarks. En primer lugar se analizan los resultados del benchmark fio, que caracteriza el rendimiento por OST sobre el dispositivo en crudo, sobre el OST montado localmente y sobre el OST a través de la red. A continuación se presentan los resultados del benchmark IOR, que miden el ancho de banda agregado del sistema de archivos paralelo bajo acceso concurrente a través de la red. Luego se presentan los resultados del benchmark MDTest, que evalúa el rendimiento del subsistema de metadatos.

6.1. Flexible I/O Tester

En esta sección se presentan los resultados de los diferentes experimentos del benchmark fio, que se realizó para medir el rendimiento por OST a través de la red y localmente. Se buscó identificar qué parte del rendimiento es atribuible al dispositivo de almacenamiento, qué parte al sistema de archivos de base del OST (por ejemplo, ldiskfs o ZFS) y qué parte al pasaje de los datos a través de la red. El objetivo fue medir el rendimiento de un flujo de datos secuencial, no el máximo agregado entre varios procesos concurrentes, para comparar los distintos OST de los OSS bajo condiciones idénticas. El benchmark correspondiente para varios procesos concurrentes es IOR y se presenta en la Sección 6.2.

6.1.1. Resultados

Las Tablas 6.1, 6.2 y 6.3 presentan la media y desviación estándar (σ) de la velocidad de lectura y escritura obtenida en los experimentos de las familias rawdev, localmount y client, por sistema de archivos de base (sist. arch. base) cuando corresponda. Para simplificar la notación, en adelante los nombres de los OSS se utilizan para identificar el OST alojado en cada uno de ellos.

OST	*lectura (MiB/s)*		*escritura (MiB/s)*	
	media	σ	*media*	σ
agnetha	684,0	1,0	416,0	38,4
annifrid	678,0	5,6	280,7	38,9
benny	369,7	48,5	260,7	6,8
bjorn	448,7	4,2	232,7	6,4

Tabla 6.1: Resultados de los experimentos de fio en la familia rawdev.

OST	*sistema de archivos de base*	*lectura (MiB/s)*		*escritura (MiB/s)*	
		media	σ	*media*	σ
agnetha	ldiskfs	653,7	0,6	389,0	27,1
	ZFS	851,0	26,1	412,3	7,6
	ZFS+LZ4	708,7	44,1	384,7	15,1
annifrid	ldiskfs	653,0	8,7	100,6	7,6
	ZFS	546,0	7,9	123,3	16,6
	ZFS+LZ4	532,0	54,2	136,3	19,4
benny	ldiskfs	260,3	24,9	160,7	0,6
	ZFS	522,3	22,0	361,0	8,0
	ZFS+LZ4	502,3	0,6	303,0	3,5
bjorn	ldiskfs	402,3	2,5	217,3	11,2
	ZFS	356,7	1,5	128,7	20,6
	ZFS+LZ4	341,3	6,7	137,3	13,7

Tabla 6.2: Resultados de los experimentos de fio en la familia localmount.

La escritura fue más lenta que la lectura en todos los experimentos y sistemas de archivos de base de las Tablas 6.2 y 6.3. La diferencia en rendimiento entre la lectura y la escritura es consistente con el uso de espejado en el almacenamiento, la distribución de las lecturas entre los espejos de los dispositivos de almacenamiento (RAID 1+0 o vdevs ZFS espejados) permitió un mayor

OST	*sistema de archivos de base*	*lectura (MiB/s)*		*escritura (MiB/s)*	
		media	σ	*media*	σ
agnetha	ldiskfs	535,3	1,5	148,3	1,5
	ZFS	701,7	15,4	9,0	0,1
	ZFS+LZ4	703,7	7,0	8,8	0,2
annifrid	ldiskfs	522,3	0,6	91,7	0,4
	ZFS	418,7	8,1	6,2	0,3
	ZFS+LZ4	421,3	3,5	6,5	0,2
benny	ldiskfs	313,0	86,6	94,8	1,4
	ZFS	603,7	5,8	20,9	7,5
	ZFS+LZ4	604,3	9,1	25,4	0,6
bjorn	ldiskfs	270,3	5,1	63,7	0,7
	ZFS	391,0	4,0	8,2	0,2
	ZFS+LZ4	385,7	3,2	8,0	0,4

Tabla 6.3: Resultados de los experimentos de fio en la familia client.

paralelismo. En cambio, la replicación de cada escritura en las dos copias limitó el rendimiento de escritura. La diferencia entre lectura y escritura varió fuertemente según el experimento: fue moderada en rawdev y localmount y extrema en la familia client sobre ZFS. La activación de la compresión en ZFS no tuvo un efecto relevante sobre la escritura, consistente con el hecho de que la carga de fio se generó con datos pseudoaleatorios poco compresibles; en la lectura, en cambio, en la familia localmount introdujo una moderada reducción de rendimiento, más marcada en agnetha (de 851,0 a 708,7 MiB/s).

En la fase de lectura de los experimentos de la familia rawdev los dos nodos con seis discos obtuvieron resultados en promedio 66 % mayores que los de los nodos con cuatro discos. Los tres nodos con controladoras de disco sin caché obtuvieron resultados similares en escritura; el nodo con seis discos y sin caché obtuvo un resultado ligeramente mayor que el de los otros dos nodos con cuatro discos. El nodo con seis discos y caché obtuvo un resultado 48,2 % mayor que el del nodo con seis discos y sin caché, consistente con el uso de caché de escritura en la controladora RAID, que puede almacenar temporalmente parte de las operaciones y aumentar el rendimiento percibido. La desviación estándar en las escrituras fue mayor en los nodos con seis discos que en los nodos de cuatro discos.

Los resultados de la familia rawdev, al medir el dispositivo de bloques sin la intervención de un sistema de archivos, constituyen la comparación más directa contra las cotas teóricas de la Sección 4.6. Los cuatro OST alcanzaron entre 46 % y 70 % de su respectiva cota $R_{\mathrm{OST}}^{\mathrm{lect}}$ en lectura. Las cotas $R_{\mathrm{OST}}^{\mathrm{lect}}$ corresponden al mejor caso de lectura espejada, en el que un balanceo ideal reparte las solicitudes entre ambas copias del espejo y duplica el ancho de banda de lectura respecto de un único disco; la posible ausencia de un reparto ideal en las condiciones medidas puede explicar, en parte, la distancia respecto al máximo teórico. En escritura, agnetha obtuvo 416,0 MiB/s frente a la cota ajustada por espejado $R_{\mathrm{OST}}^{\mathrm{esc}}$ de 491,1 MiB/s (85 %) y los tres OST restantes se ubicaron entre 58 % y 65 % de sus respectivas cotas.

En los experimentos de la familia localmount en el nodo benny, que permitió acceder a los discos individualmente en modo pass-through, ZFS obtuvo en las escrituras un rendimiento 2,25 veces mayor que ldiskfs (361,0 frente a 160,7 MiB/s). En los nodos annifrid y bjorn, donde ZFS operó sobre un único arreglo RAID 1+0, el rendimiento de la escritura de ZFS fue muy inferior a la de los nodos con acceso individual a los discos (123,3 y 128,7 frente a los 412,3 y 361,0 MiB/s de agnetha y benny); respecto a ldiskfs en el mismo nodo, en bjorn la escritura de ZFS quedó por debajo (128,7 frente a 217,3 MiB/s). La diferencia más significativa en lecturas se dio en el nodo benny, donde ZFS obtuvo un rendimiento de aproximadamente el doble que ldiskfs (522,3 frente a 260,3 MiB/s), independientemente de si la compresión estuvo activada. En el resto de los nodos las diferencias entre sistemas de archivos de base en lectura fueron menores; en annifrid y bjorn ZFS tuvo un menor rendimiento que ldiskfs y en agnetha ZFS sin compresión superó a ldiskfs.

Los nodos agnetha y annifrid fueron equipados con la misma controladora de almacenamiento y la misma cantidad de discos y en ldiskfs su rendimiento de lectura fue muy similar; sin embargo, en ZFS sin compresión agnetha superó a annifrid en 234 % en la fase de escritura y en 56 % en la fase de lectura. La diferencia entre ambos nodos fue la forma en que los discos se expusieron a ZFS: en agnetha se expusieron como discos individuales, que ZFS manejó de forma individual y en annifrid como un único arreglo RAID 1+0.

Cuando ZFS accede a los discos individualmente (en los nodos agnetha y benny), ZFS gestiona internamente el espejado y el striping y distribuye las operaciones en paralelo sobre los distintos vdevs. Si en cambio recibe un único dispositivo (por ejemplo, el RAID 1+0 de annifrid y bjorn), opera sobre un

solo vdev y no puede planificar ni paralelizar las operaciones entre discos, ya que la controladora le oculta la geometría física del arreglo. Además, la gestión interna de los discos explica que agnetha (en lectura) y benny (en lectura y escritura) superaran sus respectivos resultados de rawdev, ya que la utilización de los discos de ZFS mediante una disposición distinta a la del arreglo RAID 1+0 produjo un mayor rendimiento sobre el mismo hardware.

La comparación entre las familias localmount y client sobre ldiskfs permitió aislar el costo de atravesar la red. La penalización en lectura fue moderada, de entre 18 % y 33 % en agnetha, annifrid y bjorn; en benny, en cambio, la lectura fue superior a la obtenida en la familia localmount, pero con una desviación estándar muy alta. En escritura, la penalización fue mayor y más dispar; se ubicó entre 9 % de annifrid y 71 % de bjorn y la penalización fue especialmente pronunciada en agnetha y bjorn, cuyos rendimientos de escritura local no se sostuvieron al atravesar la red.

6.1.2. Anomalías detectadas

Se utilizó la herramienta bpftrace para analizar el comportamiento de los dispositivos físicos durante las mediciones y explicar los resultados anómalos. Las mediciones se centraron en el punto en el que el kernel emite una solicitud de E/S hacia el dispositivo físico. Se contabilizó la distribución del tamaño en bytes de las solicitudes emitidas hacia el dispositivo (métrica block_rq_issue) y el número de segmentos físicos discontiguos que componen cada solicitud (métrica nr_phys_segments). También se midieron, en cada servidor de almacenamiento, el valor de tres parámetros expuestos por cada dispositivo: max_segments, que determina el número máximo de segmentos físicos no contiguos que pueden estar presentes en una misma solicitud de E/S; max_sectors_kb, que determina el tamaño máximo en KiB de una solicitud individual; y max_hw_sectors_kb, que es el límite máximo que el hardware permite fijar para max_sectors_kb.

Los segmentos físicos que componen una solicitud de E/S corresponden a regiones contiguas de memoria principal asociadas al buffer de la operación de E/S; aunque desde el punto de vista de la aplicación un buffer aparece como una región continua de memoria virtual, internamente puede estar compuesto por múltiples marcos de memoria física no contigua. Durante la preparación de una solicitud de E/S, el kernel le indica al dispositivo las direcciones físicas que

contienen los datos. Marcos físicos de memoria contiguos pueden agruparse en un único segmento y marcos separados requieren segmentos distintos, por lo que el tamaño de una solicitud es la suma de los tamaños de sus segmentos. Un mayor número de segmentos implica que la solicitud debe contener más regiones de memoria no contiguas, por lo que valores reducidos de max_segments pueden limitar el tamaño de las solicitudes.

En cada solicitud de E/S al dispositivo se cumple necesariamente que nr_phys_segments $\leq$ max_segments y que el tamaño de la solicitud no supera max_sectors_kb, ya que no se pueden emitir más segmentos físicos discontiguos ni solicitudes más grandes de lo que el dispositivo puede procesar. El valor de max_segments está determinado por la controladora y no es configurable; max_sectors_kb sí es configurable, pero no puede superar el límite de hardware expuesto por max_hw_sectors_kb. El valor de max_segments por nodo se presenta en la Tabla 6.4.

nodo	*max_segments*
agnetha	543
benny	60
annifrid	31
bjorn	31

Tabla 6.4: Valor de max_segments por nodo.

El valor más bajo corresponde a las controladoras de almacenamiento HPE Smart Array sin caché. La controladora Dell PERC H310 expone un valor más alto (60) y el valor más alto es el expuesto por la controladora Smart Array con caché (543), 17,5 veces mayor al valor más bajo. Los nodos agnetha y annifrid fueron equipados con el mismo modelo de controladora y su única diferencia fue la presencia del módulo de caché, por lo que la diferencia de valores de max_segments se explica por la presencia del módulo. En todos los nodos, max_sectors_kb coincidió con max_hw_sectors_kb y fue 1024 KiB, excepto en benny, donde ambos valores fueron 128 KiB; que max_sectors_kb iguale a max_hw_sectors_kb indica que en todos los casos el parámetro estuvo fijado en su máximo posible.

En todos los experimentos con fio se utilizó un tamaño de bloque de 1 MiB. Un segmento físico puede abarcar una o varias páginas físicamente contiguas, de modo que su tamaño no es fijo: en el mejor caso, un buffer de 1 MiB

completamente contiguo cabe en un único segmento y en el peor caso, con el buffer completamente fragmentado a nivel físico y páginas de 4 KiB, la misma operación requiere 256 segmentos. Por lo tanto, max_segments solo limita el tamaño de una solicitud cuando el buffer está fragmentado; en el peor caso, si cada segmento contiene una única página de memoria, la solicitud queda acotada a max_segments×4 KiB. Para los nodos con max_segments = 31, como annifrid y bjorn, el peor caso es 31 segmentos×4 KiB= 124 KiB por petición. La Tabla 6.5 presenta el número de segmentos (cant. segmentos) y los tamaños de petición dominantes por experimento (tam. dominante peticiones) por sistema de archivos de base medido en el nodo agnetha. Para las familias rawdev y localmount el patrón fue equivalente en todos los nodos, por lo que se reporta a agnetha como caso representativo.

familia	*sistema de archivos de base*	*cantidad de segmentos*	*tamaño dominante de peticiones*
rawdev	—	16–17 (91 %)	1 MiB (100 %)
localmount	ldiskfs	16–17 (94 %)	1 MiB (100 %)
	ZFS	4 (48 %)	256 KiB (99 %)
	ZFS+LZ4	4–5 (83 %)	256 KiB (99 %)
client	ldiskfs	37–38 (34 %)	1 MiB (93 %)
	ZFS	1 (51 %)	4 KiB (51 %)
	ZFS+LZ4	1 (52 %)	4 KiB (52 %)

Tabla 6.5: Número de segmentos y tamaño de petición dominantes (en escritura) en agnetha, para cada una de las tres familias de experimentos.

La Tabla 6.6 presenta los valores dominantes de segmentos y de tamaño de petición en cada nodo para la escritura de los experimentos de la familia client.

En el sistema de archivos de base ZFS, con y sin compresión, los cuatro nodos presentaron el mismo patrón dominante: un único segmento y peticiones de 4 KiB. En el sistema de archivos de base ldiskfs, en cambio, el tamaño de petición estuvo condicionado por los límites del subsistema de bloque de cada nodo. En agnetha, con max_segments = 543 y max_sectors_kb = 1024 KiB, el tamaño de las peticiones dominantes fue 1 MiB. En benny, con max_segments = 60 pero max_sectors_kb = 128 KiB, el tamaño de petición quedó limitado a 128 KiB por max_sectors_kb.

nodo	*sistema de archivos de base*	*cantidad de segmentos*	*tamaño dominante de peticiones*
agnetha	ldiskfs	37–38 (34 %)	1 MiB (93 %)
	ZFS	1 (51 %)	4 KiB (51 %)
	ZFS+LZ4	1 (52 %)	4 KiB (52 %)
annifrid	ldiskfs	31 (55 %)	1 MiB (8 %)
	ZFS	1 (39 %)	4 KiB (38 %)
	ZFS+LZ4	1 (36 %)	4 KiB (36 %)
benny	ldiskfs	5 (21 %)	128 KiB (94 %)
	ZFS	1 (39 %)	4 KiB (39 %)
	ZFS+LZ4	1 (42 %)	4 KiB (41 %)
bjorn	ldiskfs	31 (79 %)	124 KiB (20 %)
	ZFS	1 (44 %)	4 KiB (38 %)
	ZFS+LZ4	1 (38 %)	4 KiB (33 %)

Tabla 6.6: Cantidad de segmentos físicos y tamaño de petición dominantes (en escritura) de los experimentos de la familia client por nodo y sistema de archivos de base.

Los nodos annifrid y bjorn tuvieron el mismo valor de max_segments (31) pero produjeron distribuciones de tamaño de petición distintas. En bjorn, 79 % de las escrituras alcanzó el techo de 31 segmentos y 20 % de las solicitudes tuvo un tamaño de 124 KiB (31 segmentos × 4 KiB, el caso de fragmentación máxima con una sola página por segmento), con un tamaño medio de petición de aproximadamente 251 KiB. En annifrid, 55 % de las escrituras alcanzó también el techo de 31 segmentos, pero no hubo un tamaño de petición dominante: más del 82 % de las peticiones se repartieron entre 128 KiB y 1 MiB y el tamaño medio de petición fue aproximadamente 515 KiB, el doble que en bjorn. Ambos nodos saturaron el mismo límite de 31 segmentos, pero la mayor cantidad de páginas contiguas agrupadas por segmento permitió que annifrid emitiera peticiones de mayor tamaño.

La anomalía más relevante fue la caída en el rendimiento de escritura en los experimentos de la familia client sobre el sistema de archivos de base ZFS con y sin compresión: el rendimiento cayó a 6–25 MiB/s frente a los 123–412 MiB/s medidos para el mismo sistema de archivos de base en los experimentos de la familia localmount. El colapso apareció únicamente en la operación de es-

critura; en cambio, en lectura, el sistema de archivos de base ZFS en la familia client no presentó esta caída de rendimiento. Las peticiones dominantes de 4 KiB reportadas en la Tabla 6.6 correspondieron, además, a escrituras dispersas por el dispositivo. La herramienta iostat mostró que en la fase de escritura de ZFS con y sin compresión el conjunto de los seis discos del pool atendió aproximadamente 400 operaciones por segundo, con un tamaño de petición promedio de 57 KiB. Individualmente, los discos operaron con una utilización media de entre 67 % y 84 %, con picos de entre 89 % y 98 %, un patrón de utilización oscilante y una latencia media de 24 ms. Las cifras obtenidas fueron compatibles con discos mecánicos cuyo rendimiento estuvo limitado por la latencia de posicionamiento, ya que los dispositivos estuvieron saturados atendiendo un gran número de operaciones pequeñas y dispersas. En contraste, las escrituras de la familia localmount sobre el mismo dispositivo, con el mismo sistema de archivos de base y la misma carga de fio, presentaron un tamaño de petición dominante de 256 KiB, superior a los 4 KiB de la familia client.

En el nodo agnetha, para el sistema de archivos de base ldiskfs, no apareció fragmentación extrema y se obtuvo un rendimiento 16,5 veces mayor que ZFS sobre los mismos seis discos (expuestos como un único disco lógico RAID 1+0). La herramienta iostat mostró que durante la fase de escritura el disco lógico operó con un tamaño de petición cercano a 1 MiB (media 957 KiB, mínimo 936 KiB), atendió aproximadamente 157 operaciones por segundo y tuvo una utilización media del 98 %. ldiskfs atendió menos operaciones por segundo que ZFS pero de un tamaño considerablemente mayor y ZFS atendió un gran número de operaciones por segundo pero dispersas y de tamaño reducido.

La baja de rendimiento no fue atribuible a ningún modelo específico de controladora de disco, ya que el nodo agnetha, con controladora con caché y valor de max_segments alto, también experimentó el colapso de rendimiento. Tampoco se explica por la cantidad de vdevs de ZFS, ya que el colapso se dio de manera similar en pools de uno, dos y tres vdevs. Los resultados sugieren una interacción desfavorable entre la carga fio, el camino cliente-Lustre y el sistema de archivos de base ZFS, pero no permitieron aislar una única causa.

6.1.3. Síntesis

Los experimentos con el benchmark fio permitieron descomponer el rendimiento de un flujo secuencial por OST en sus tres contribuciones: el dispositivo de bloques (rawdev), el sistema de archivos de base del OST (localmount) y el pasaje por la red (client). En todos los casos la escritura fue más lenta que la lectura, un comportamiento consistente con el espejado del almacenamiento, que reparte las lecturas entre ambas copias pero no ofrece el mismo paralelismo a las escrituras. En la familia rawdev los dos nodos con seis discos tuvieron los mejores resultados de lectura y la caché de escritura de la controladora fue el factor que varió más fuertemente el rendimiento en la escritura. Frente a las cotas teóricas, los OST alcanzaron entre 46 % y 70 % del rendimiento en lectura y hasta 85 % del rendimiento en escritura.

Cuando ZFS gestionó los discos de forma individual, el rendimiento superó a la configuración sobre un único arreglo RAID 1+0 e incluso mejoró los resultados de rawdev en lectura. Sobre un único vdev, ZFS no pudo planificar ni paralelizar las operaciones entre discos y el rendimiento se vio penalizado. El pasaje por la red introdujo una degradación del rendimiento en lectura y escritura.

La anomalía más relevante fue el colapso de la escritura en la familia client sobre ZFS, que cayó a 6–25 MiB/s frente a los 123–412 MiB/s de la familia localmount sobre el mismo sistema de archivos de base. El análisis con bpftrace e iostat mostró que el colapso se debió a un patrón de escrituras pequeñas (4 KiB) y dispersas que saturó los discos con operaciones de posicionamiento, efecto ausente en la lectura y en el sistema de archivos de base ldiskfs. El fenómeno se reprodujo en pools de uno, dos y tres vdevs y en distintas controladoras, por lo que no fue atribuible al hardware sino a una interacción desfavorable entre la carga de fio, el camino cliente-Lustre y el sistema de archivos de base ZFS que no se logró aislar por completo.

6.2. Interleaved or Random

El benchmark IOR mide el ancho de banda agregado de lectura y escritura del sistema de archivos paralelo bajo cargas de E/S de procesos concurrentes. En esta sección se evalúa el impacto sobre el rendimiento de seis parámetros: el número de OST utilizados para el striping de cada archivo, el tamaño de

stripe, el modo de acceso, el número de procesos concurrentes, el número de clientes y el sistema de archivos de base de los OST. Todos los experimentos se realizaron sobre el sistema de archivos paralelo con cuatro OST.

A partir de una configuración de base, se varió un parámetro por vez y se mantuvo el resto constante. La variación de un parámetro por vez permite aislar el efecto en el rendimiento de cada factor. La configuración de base consistió en dos clientes, modo de acceso FPP, striping sobre cuatro OST con tamaño de stripe de 4 MiB, doce procesos concurrentes, total de datos de 160 GiB y escalamiento fuerte. La fila correspondiente a la configuración de base se marca con fondo gris en las tablas de resultados experimentales.

6.2.1. Efecto del número de OST utilizados para striping

La Tabla 6.7 presenta el ancho de banda de lectura y escritura al variar el número de OST sobre los que se distribuyó cada archivo. En el caso de un OST, los archivos se almacenaron íntegros en alguno de los cuatro OST del sistema.

configuración	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
1 OST	343,4	6,3	1475,4	13,0
2 OST	358,8	2,3	1323,8	41,4
3 OST	376,7	34,0	1275,4	88,6
4 OST	348,6	8,0	1314,9	26,2

Tabla 6.7: Rendimiento de IOR según el número de OST utilizados para el striping de los archivos.

El rendimiento de escritura fue prácticamente constante entre los distintos escenarios; los cuatro valores se ubicaron en un rango cercano y las diferencias fueron del orden de la desviación estándar. Aumentar el stripe count no agregó paralelismo útil ya que todos los OST estaban ocupados con carga de E/S debido a los procesos concurrentes. En lectura, en cambio, la configuración de un solo OST fue la más veloz, 12 % por encima de la configuración de base y las configuraciones de dos, tres y cuatro OST rindieron de forma similar entre sí.

Alojar cada archivo en un único OST pudo haber aprovechado la secuencialidad de los datos para la fase de lectura, que se correspondería con menores accesos a disco y menor coordinación. Con un stripe count mayor, cada archivo

debe reconstruirse a partir de fragmentos alojados en varios OST y como el resto de OST estaban cargados con las peticiones de otros archivos, se introdujo sobrecarga sin aportar paralelismo adicional.

6.2.2. Efecto del tamaño de stripe

La Tabla 6.8 muestra el rendimiento al variar el tamaño de stripe entre 256 KiB y 16 MiB.

tamaño de stripe	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
256 KiB	353,5	2,3	1220,7	3,6
1 MiB	346,1	3,6	1220,6	5,7
4 MiB	348,6	8,0	1314,9	26,2
16 MiB	331,1	1,2	1263,2	8,5

Tabla 6.8: Rendimiento de IOR según el tamaño de stripe utilizado para los archivos.

El tamaño de stripe tuvo un efecto menor en escritura; los valores se mantuvieron entre 331,1 y 353,5 MiB/s, con una leve caída en el escenario con stripes de 16 MiB. El ancho de banda de escritura estuvo limitado por la capacidad de los OST y no por el tamaño de la unidad de transferencia, por lo que ninguna de las variantes se apartó de manera significativa de la configuración de base.

En lectura, el rendimiento de los tamaños de stripe mayores (4 y 16 MiB) superó al rendimiento de los tamaños de stripe pequeños (256 KiB y 1 MiB) por 3,5 % a 8 %. El mejor resultado correspondió al tamaño de stripe de 4 MiB (la configuración de base) y el peor resultado correspondió al tamaño de stripe de 256 KiB. Los stripes pequeños fragmentan cada archivo en más objetos, generan una mayor cantidad de RPC y operaciones de E/S de menor tamaño y aumentan la sobrecarga. Los stripes más grandes favorecen transferencias contiguas.

6.2.3. Efecto del modo de acceso

La Tabla 6.9 compara los modos de acceso FPP, en el que cada proceso escribe y lee su propio archivo y SSF, en el que todos los procesos comparten un único archivo.

modo	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
FPP	348,6	8,0	1314,9	26,2
SSF	349,5	23,4	1247,3	3,1

Tabla 6.9: Rendimiento de IOR según el modo de acceso.

En escritura, ambos modos rindieron prácticamente igual, con una diferencia inferior a 1 %, aunque SSF presentó una desviación estándar mayor. Posiblemente la contención por bloqueos y el control de concurrencia sobre el archivo compartido introdujo variabilidad entre repeticiones, aunque el rendimiento promedio no se modificó. En lectura, FPP superó a SSF por 5,4 %. Sin embargo, SSF resultó mucho más estable en lectura, con una desviación estándar marcadamente menor.

6.2.4. Efecto del número de procesos

La Tabla 6.10 muestra la evolución del rendimiento de lectura y escritura al variar el número de procesos concurrentes.

número de procesos	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
1	318,8	9,1	1110,0	0,8
2	343,7	2,7	1343,4	9,0
4	341,2	2,0	1266,2	9,5
8	360,2	2,3	1318,4	39,0
12	348,6	8,0	1314,9	26,2
24	357,2	0,6	1236,8	4,6

Tabla 6.10: Rendimiento de IOR según el número de procesos.

En escritura, el menor resultado de ancho de banda se obtuvo con un único proceso. El ancho de banda de escritura fue creciendo al aumentar el número de procesos hasta llegar a su valor máximo con ocho procesos. En lectura, el valor con un único proceso fue también el más bajo y a la vez el más estable. Con un solo proceso interviene un único cliente, cuyo enlace de red de 10 Gb/s (1192 MiB/s teóricos) constituyó el cuello de botella. El valor obtenido representó 93,1 % del máximo de red, consistente con la eficiencia de TCP sobre un enlace de 10 Gb/s. A partir de dos procesos se incorporó el segundo cliente y la lectura ascendió a su máximo, para luego fluctuar sin una tendencia marcada.

6.2.5. Efecto del número de clientes

La Tabla 6.11 compara el rendimiento del sistema de archivos paralelo con uno, dos y tres clientes.

clientes	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
1	316,2	12,5	1114,9	1,5
2	348,6	8,0	1314,9	26,2
3	356,8	1,0	961,0	5,5

Tabla 6.11: Rendimiento de IOR según el número de clientes.

El rendimiento en escritura aumentó 10 % al pasar de uno a dos clientes y se mantuvo estable con tres clientes. Agregar un tercer cliente aumentó el rendimiento de escritura pero no lo hizo de manera significativa. El cuello de botella en escritura fue, en consecuencia, la capacidad agregada de los OST y no el número de clientes. En lectura, el comportamiento no fue monótono. Con un cliente el ancho de banda fue limitado por el enlace de red de 10 Gb/s del único nodo activo, al igual que en el caso de un solo proceso de la Sección 6.2.4. Con dos clientes se alcanzó el máximo valor obtenido al sumar un segundo enlace de red. Con tres clientes, sin embargo, la lectura cayó 27 % respecto al valor con dos clientes; al competir tres clientes por los cuatro OST, la contención sobre los servidores de datos probablemente degradó el rendimiento. La lectura, por lo tanto, se vio perjudicada por el aumento de concurrencia.

6.2.6. Efecto del sistema de archivos de base

La Tabla 6.12 compara el rendimiento de Lustre sobre tres sistemas de archivos de base de los OST: ldiskfs, ZFS sin compresión y ZFS con compresión LZ4 activada.

sistema de archivos de base	*escritura (MiB/s)*	σ	*lectura (MiB/s)*	σ
ldiskfs	348,6	8,0	1314,9	26,2
ZFS	681,3	13,2	1161,5	55,1
ZFS+LZ4	924,1	55,2	1575,2	56,6

Tabla 6.12: Rendimiento de IOR según el sistema de archivos de base de los OST.

ZFS sin compresión casi duplicó el rendimiento de escritura de ldiskfs (95 % más), aunque en lectura quedó 12 % por debajo. La ventaja en escritura se podría atribuir al modelo transaccional de ZFS descrito en la Sección 4.2.2,

que agrupa las escrituras en grupos de transacciones y las vuelca al disco en bloques grandes y ordenados, reduce los seeks de disco y aprovecha mejor el ancho de banda secuencial del dispositivo. En lectura, en cambio, ZFS sin compresión introdujo una sobrecarga que, sin el beneficio de leer menos datos físicos, obtuvo un rendimiento por debajo de ldiskfs.

En escritura, activar la compresión LZ4 aumentó el rendimiento 36 % respecto a ZFS sin compresión y 165 % respecto a ldiskfs. En lectura, la compresión aumentó el rendimiento 36 % respecto a ZFS sin compresión y 20 % respecto a ldiskfs. El patrón sintético que escribe IOR es redundante y compresible y la compresión alcanzó una tasa de 1,98 en todos los nodos, por lo que el volumen físico escrito y leído en disco representó 50,5 % del volumen lógico configurado. En ambas operaciones, sin embargo, la mejora en rendimiento atribuible a la compresión quedó por debajo del factor 1,98. La mejora introducida por el uso de compresión no necesariamente aplica para otro tipo de cargas de datos que tengan baja compresibilidad.

6.2.7. Síntesis

Con el sistema de archivos de base ldiskfs, el rendimiento de IOR quedó acotado por límites que las opciones de configuración de Lustre apenas afectaron. La escritura se mantuvo entre 316 y 377 MiB/s y casi no cambió al variar el número de OST utilizados para el striping, el tamaño de stripe o el modo de acceso. El techo del rendimiento de escritura no correspondió a la capacidad física agregada de los OST sino a la capa de almacenamiento ldiskfs; los mismos OST alcanzaron 681 MiB/s con ZFS y 924 MiB/s con ZFS+LZ4. El sistema de archivos paralelo sí podía rendir más del máximo obtenido para ldiskfs en escritura; en consecuencia, el cuello de botella estuvo en algún punto del camino de escritura de ldiskfs y no en el hardware.

En lectura, con un solo cliente el ancho de banda estuvo limitado por la red, con un techo impuesto por el enlace de 10 Gb/s. El límite es consistente con que los escenarios de un solo cliente (Secciones 6.2.4 y 6.2.5) resultaron en valores prácticamente idénticos (1110,0 y 1114,9 MiB/s), independientemente del número de procesos. La lectura alcanzó un máximo con dos clientes al sumar un segundo enlace de red y luego se degradó con tres clientes.

Ajustar los parámetros de striping o de acceso afectó ligeramente el rendimiento de escritura y produjo variaciones algo mayores en lectura (hasta 12 % al concentrar cada archivo en un único OST y hasta 8 % según el tamaño de stripe), sin alterar el orden de magnitud. El mayor factor de mejora del rendimiento fue utilizar ZFS como sistema de archivos de base en los OST. Cambiar de ldiskfs a ZFS casi duplicó el rendimiento de escritura, aunque penalizó levemente la lectura. La compresión LZ4 revirtió la penalización, mejoró el rendimiento de ambas operaciones y produjo el mejor resultado de todos los experimentos, al reducir la E/S física sobre datos altamente compresibles.

Por lo tanto, el parámetro que más influyó en la mejora del rendimiento no estuvo en la configuración de Lustre sino en la elección del sistema de archivos de base, con la salvedad de que la ventaja de la compresión no aplicaría en cargas con datos poco compresibles. Como limitación del diseño experimental, al variar un parámetro por vez no se exploraron las interacciones entre factores.

6.3. MDTest

El benchmark MDTest se utilizó para medir el rendimiento del sistema de metadatos del sistema de archivos paralelo. Cada escenario se ejecutó sobre tres sistemas de archivos de base: ldiskfs, ZFS sin compresión y ZFS con compresión.

6.3.1. Escalamiento débil con directorios únicos

En el escenario de escalamiento débil con directorios únicos cada proceso operó en su propio directorio sobre 20 000 archivos. La Tabla 6.13 presenta las operaciones por segundo obtenidas en el escenario experimental.

La operación de creación mostró comportamientos opuestos entre ldiskfs y los dos sistemas de archivos de base ZFS. Con ldiskfs, agregar procesos en un único cliente aumentó las operaciones por segundo hasta saturar entre ocho y dieciséis procesos y agregar un segundo cliente escaló el rendimiento. La tasa de creación creció aproximadamente tres veces entre cuatro y dieciséis procesos, por lo que con un cliente la saturación estuvo en el lado del cliente y no en el MDS. Con ZFS y ZFS con compresión, la tasa de creación con un único cliente cayó a medida que aumentó el número de procesos y al utilizar dos clientes el rendimiento empeoró.

sistema de archivos de base	*clientes*	*número de procesos*	*creación*		*stat*		*eliminación*	
			tasa (op/s)	σ	*tasa (op/s)*	σ	*tasa (op/s)*	σ
ldiskfs	1	4	11501	338	5437	97	21375	200
	1	8	17942	84	15544	547	34718	680
	1	16	17649	116	32252	193	33780	414
	2	4	11678	165	4526	127	22821	61
	2	8	22472	1247	14001	244	40521	895
	2	16	35606	2208	37035	295	66618	2018
ZFS	1	4	8918	366	3424	217	17115	42
	1	8	8175	116	5930	115	27713	238
	1	16	5223	630	12068	165	27459	68
	2	4	860	74	3292	421	15118	83
	2	8	770	59	6789	94	28310	12
	2	16	791	60	10342	93	45821	213
ZFS+LZ4	1	4	7400	597	2543	20	15874	85
	1	8	2630	52	5076	61	26574	194
	1	16	1859	29	8882	928	27037	38
	2	4	6203	54	3239	91	15003	214
	2	8	694	22	6900	147	28029	104
	2	16	902	76	10145	304	44854	75

Tabla 6.13: Resultados de MDTest en el escenario escalamiento débil con directorios únicos.

La operación stat presentó el patrón más uniforme entre los tres sistemas de archivos de base. En todos los casos el rendimiento creció de manera aproximadamente lineal con el número de procesos, comportamiento esperable al tratarse de una operación de solo lectura. La diferencia entre sistemas de archivos de base fue significativa: ldiskfs alcanzó un techo de 37035 op/s con dieciséis procesos en dos clientes, ZFS alcanzó 10342 op/s y ZFS con compresión LZ4 alcanzó 10145 op/s. La compresión no afectó significativamente la tasa de stat respecto de ZFS sin compresión.

La eliminación escaló con los procesos y con el número de clientes en los tres sistemas de archivos de base. Para ldiskfs la tasa con dos clientes y dieciséis procesos casi duplicó la de un solo cliente; para ZFS, agregar el segundo cliente aumentó el rendimiento con dieciséis procesos 1,67 veces y con ZFS y compresión el comportamiento fue similar al de ZFS sin comprimir.

6.3.2. Escalamiento débil con directorio compartido

En este escenario todos los procesos escribieron 20 000 archivos en el mismo directorio. La Tabla 6.14 presenta los resultados del escenario.

sistema de archivos de base	*clientes*	*número de procesos*	*creación*		*stat*		*eliminación*	
			tasa (op/s)	σ	*tasa (op/s)*	σ	*tasa (op/s)*	σ
ldiskfs	1	4	3942	40	5456	80	4694	66
	1	8	3939	23	15719	458	4619	109
	1	16	3891	40	30367	887	4669	78
	2	4	7457	24	4395	98	8917	105
	2	8	7312	112	13220	46	8449	132
	2	16	7320	60	36022	604	8228	176
ZFS	1	4	3660	38	3154	55	3543	56
	1	8	3638	7	5450	20	3555	43
	1	16	3145	24	12103	211	3593	14
	2	4	1466	707	3390	12	7269	134
	2	8	1037	207	6601	57	7386	61
	2	16	862	31	10104	117	7293	31
ZFS+LZ4	1	4	3569	26	2841	84	3816	32
	1	8	2030	108	5069	188	3877	28
	1	16	2144	68	9664	93	3791	22
	2	4	709	18	3255	33	7294	13
	2	8	1128	63	6998	172	7275	94
	2	16	839	199	10096	144	7313	18

Tabla 6.14: Resultados de MDTest en el escenario escalamiento débil con directorio compartido.

Utilizar un único directorio compartido generó un techo en las operaciones de creación y eliminación en los tres sistemas de archivos de base, probablemente debido a los bloqueos introducidos por la modificación concurrente del mismo directorio. Con un único cliente, ldiskfs mantuvo la creación constante entre 3891 y 3942, independientemente del número de procesos y ZFS sin comprimir varió entre 3145 y 3660 op/s según el número de procesos. La eliminación se comportó de manera análoga, con 4650 op/s para ldiskfs y 3550 op/s para ZFS. En cambio, ZFS con compresión mostró una caída en la tasa de creación al pasar de cuatro a ocho procesos, de 3569 a 2030 op/s, que se mantuvo en dieciséis procesos y su tasa de eliminación fue estable entre 3791 y 3877 op/s.

Con dos clientes, el techo de creación y eliminación se duplicó para ldiskfs, hasta 7457 op/s para creación y 8917 op/s para eliminación. La eliminación para ambos sistemas de archivos de base ZFS también se duplicó hasta 7300 op/s. La creación con dos clientes en los sistemas de archivos de base ZFS no mostró la duplicación y obtuvo valores inferiores a los de un solo cliente. La operación stat no se vio afectada por el uso de un directorio compartido: escaló con el número de procesos en las tres configuraciones y alcanzó techos similares a los del escenario de escalamiento débil con directorios únicos.

6.3.3. Escalamiento fuerte con directorios únicos

El escalamiento fuerte mantuvo constante el número total de archivos en 1 000 000 y a medida que aumentó el número de procesos se redujo la carga por proceso, pero aumentó la concurrencia sobre el MDS. La Tabla 6.15 presenta los resultados del escenario.

sistema de archivos de base	*clientes*	*número de procesos*	*creación*		*stat*		*eliminación*	
			tasa (op/s)	σ	*tasa (op/s)*	σ	*tasa (op/s)*	σ
ldiskfs	1	4	10707	130	5334	150	17510	252
	1	8	15297	677	15289	105	29222	169
	1	16	15658	1295	30233	691	29721	491
	2	4	10822	370	4469	37	17646	149
	2	8	20583	39	13755	1070	34696	773
	2	16	32029	1093	34248	798	60624	641
ZFS	1	4	2364	428	3098	134	14090	15
	1	8	2246	119	5384	99	25126	57
	1	16	2100	17	10701	181	26938	44
	2	4	754	60	3118	8	13216	35
	2	8	765	46	6373	14	25416	35
	2	16	816	41	9697	67	43868	67
ZFS+LZ4	1	4	1783	37	3009	103	13887	69
	1	8	1630	25	4801	332	24859	117
	1	16	1250	280	6500	948	26614	16
	2	4	763	15	3069	194	12879	86
	2	8	701	49	6307	54	25314	163
	2	16	831	40	9753	5	43905	85

Tabla 6.15: Resultados de MDTest en el escenario escalamiento fuerte con directorios únicos.

En ldiskfs la creación se saturó con un único cliente. La creación continuó creciendo hasta dieciséis procesos con dos clientes y en ambos ZFS la tasa de creación con un solo cliente decreció a medida que se aumentó el número de procesos. La operación stat escaló con los procesos en los tres sistemas de archivos de base y mantuvo la misma diferencia de magnitud que en el escenario de escalamiento débil. En la eliminación, agregar un segundo cliente aumentó las operaciones por segundo en los tres sistemas de archivos de base.

Los patrones de escalamiento se mantuvieron pero la tasa de creación en ZFS fue menor que en el escenario de escalamiento débil. Con un único cliente, ZFS sin compresión pasó, en promedio entre todos los números de procesos, de 7439 op/s en el escalamiento débil a 2237 op/s en el escalamiento fuerte. La diferencia se dio por el mayor volumen total de metadatos del escenario. Las operaciones stat y eliminación, en cambio, mantuvieron valores del mismo orden de magnitud de rendimiento que en el escalamiento débil.

6.3.4. Experimento de estrés

Para el experimento de estrés se operó sobre 9 600 000 archivos repartidos entre 32 procesos que trabajaron en un directorio único. El objetivo de este experimento fue medir la capacidad del sistema de metadatos ante cargas muy altas y superar la capacidad de caché que pueda tener el MDS. La Tabla 6.16 presenta el resultado del experimento de estrés.

sistema de archivos de base	*clientes*	*creación*		*stat*		*eliminación*	
		tasa (op/s)	σ	*tasa (op/s)*	σ	*tasa (op/s)*	σ
ldiskfs	1	9726	443	39248	1532	14724	158
	2	21262	627	58097	2038	24012	1881
ZFS	1	1150	342	8724	6181	14674	9468
	2	1071	132	15154	1048	26667	380
ZFS+LZ4	1	1482	—	10816	—	20196	—
	2	1216	134	14407	1539	25985	556

Tabla 6.16: Resultados de MDTest en el escenario de experimento de estrés.

Los resultados del experimento de estrés evidenciaron una gran diferencia en rendimiento entre ldiskfs y ZFS con y sin compresión. En ldiskfs la operación stat alcanzó valores superiores a los registrados en los escenarios de las

secciones 6.3.1 a 6.3.3 y la creación con dos clientes fue 2,19 veces mayor a la de un solo cliente. Por lo tanto, en este experimento el cuello de botella en ldiskfs se ubicó en los clientes y no en el servidor de metadatos.

En los sistemas de archivos de base ZFS, la tasa de creación cayó a valores menores a 1500 op/s en ambas configuraciones de clientes, con resultados similares entre uno y dos clientes. La creación en ZFS no se benefició de mayor concurrencia y aumentó la degradación registrada al aumentar el volumen de metadatos entre el escalamiento débil y el escalamiento fuerte. La diferencia entre ZFS y ZFS con compresión no fue significativa.

Para ZFS con compresión y un único cliente solo se dispone de datos de una ejecución. Las dos repeticiones adicionales no pudieron completarse porque el nodo cliente agotaba su espacio en memoria RAM, llenado por el buffer que contenía los metadatos pendientes de escritura que no llegaban a escribirse en el sistema de archivos. Cuando el sistema de archivos de base no procesó los metadatos al ritmo en que los clientes los generaron, los buffers se acumularon en el cliente hasta saturar su memoria. Este problema es consistente con el cuello de botella en el sistema de archivos de base de almacenamiento en los escenarios anteriores, donde ZFS en ambas configuraciones tuvo un rendimiento menor a ldiskfs.

Con ldiskfs se completaron las tres ejecuciones del experimento de estrés, por lo que el fallo ocurrido en el experimento de estrés para ZFS con y sin compresión no fue originado por una limitación del cliente. En ZFS con compresión, la carga impuesta sobre el sistema de metadatos fue lo suficientemente alta para saturar al cliente, el rendimiento del sistema de archivos de base no fue suficiente para atender la carga y el experimento falló. En ZFS sin compresión se lograron realizar las tres repeticiones del experimento pero la diferencia de los resultados entre las tres operaciones fue muy alta, con desviaciones estándar de entre 30 % y 70 % de la media. La variabilidad obtenida sugiere que ZFS sin compresión también operó cerca de la capacidad del sistema, pero sin llegar al fallo presentado en ZFS con compresión. Con dos clientes, la desviación estándar fue baja para ambos sistemas de archivos de base ZFS, por lo que el problema fue consecuencia de operar al límite de la capacidad del sistema con un único cliente.

Con dos clientes, la operación stat presentó un techo entre 14407 y 15154 op/s en ambos sistemas de archivos de base ZFS, muy por debajo de las 58097 op/s de ldiskfs. La eliminación alcanzó 26667 op/s en ZFS y 25985 op/s en

ZFS con compresión, valores levemente superiores a los 24012 op/s de ldiskfs. El experimento de estrés es el único caso de todos los experimentos realizados con MDTest en el que los sistemas de archivos de base ZFS superaron a ldiskfs.

6.3.5. Síntesis

A diferencia de los resultados del benchmark IOR, donde ZFS tuvo mayor rendimiento que ldiskfs, en los experimentos con MDTest ldiskfs superó a ZFS en casi todos los escenarios y operaciones, por lo que el sistema de archivos de base volvió a ser el factor determinante del rendimiento. Además, el rendimiento dependió del tipo de operación más que de la configuración de Lustre.

Con ldiskfs, la tasa de creación creció al aumentar la concurrencia y escaló al agregar un segundo cliente, por lo que el cuello de botella estuvo en el cliente y no en el MDS. Con ZFS la tasa de creación se redujo a medida que aumentó el número de procesos y empeoró al pasar a dos clientes, por lo que la operación de creación en ZFS no escaló con el aumento de la concurrencia. Además, la creación en ZFS se degradó de forma progresiva a medida que creció el volumen total de metadatos del experimento: con un único cliente ZFS sin compresión pasó de 7439 op/s en el escalamiento débil con directorios únicos a 2237 op/s en el escalamiento fuerte y a 1150 op/s en el experimento de estrés.

La operación stat fue la más uniforme; escaló de forma aproximadamente lineal con los procesos en los tres sistemas de archivos de base, comportamiento consistente con el hecho de que stat es una operación de solo lectura. Para stat ldiskfs mantuvo un techo de entre tres y cuatro veces el de los sistemas de archivos de base ZFS, de 37035 op/s frente a un rango de 10000 a 12000 op/s. Fue también la única operación que no se vio penalizada por el uso de un directorio compartido.

La eliminación escaló con los procesos y con el número de clientes en los tres sistemas de archivos de base, nuevamente con ldiskfs en el techo más alto, con 66618 op/s frente a 45821 op/s de los sistemas de archivos de base ZFS. La creación y la eliminación son operaciones de escritura de metadatos, pero en ZFS la eliminación escaló con la concurrencia y el número de clientes y la creación colapsó.

La compresión no introdujo mejoras en ninguna operación de metadatos. Su efecto fue prácticamente nulo en stat y en eliminación y en la creación agregar compresión redujo el rendimiento respecto a ZFS sin compresión.

En los experimentos de estrés con los sistemas de archivos de base ZFS, la creación cayó por debajo de 1500 op/s sin beneficiarse de mayor concurrencia. Con ZFS con compresión, dos de las tres ejecuciones con un único cliente no pudieron completarse, ya que la memoria del cliente se saturó con metadatos pendientes de escritura que el sistema de archivos de base no procesó al ritmo en que se generaron. El único caso en todos los experimentos de MDTest en el que los sistemas de archivos de base ZFS superaron a ldiskfs fue la eliminación bajo el experimento de estrés. En general, ldiskfs superó a ambos sistemas de archivos de base ZFS en todos los escenarios y operaciones, con la excepción de la eliminación en el experimento de estrés.

Capítulo 7

Evaluación experimental con aplicaciones

En este capítulo se presentan y analizan los resultados de los experimentos de evaluación de rendimiento del sistema de archivos paralelo a través de la ejecución de dos aplicaciones de consumo masivo de datos y una aplicación intensiva en cómputo. Se reportan los resultados de la ejecución de las aplicaciones evaluadas sobre las distintas configuraciones de Lustre junto al análisis del uso de disco y CPU obtenido durante su ejecución. Para cada experimento se contrastan los tres sistemas de archivos de base evaluados (ldiskfs, ZFS sin compresión y ZFS con compresión), se identifican los cuellos de botella, se explican las anomalías de rendimiento, se comparan los tiempos ponderados por heterogeneidad de OST con los de las ejecuciones sobre NFS y se reportan la aceleración y la eficiencia obtenidas entre configuraciones de Lustre.

7.1. Procesamiento de datos para DeepONet

En esta sección se presentan los resultados de la caracterización del comportamiento de E/S de la aplicación de procesamiento de datos para DeepONet obtenidos con la metodología descrita en la Sección 5.4.1 y a continuación se evalúa su rendimiento sobre las distintas configuraciones de Lustre estudiadas.

7.1.1. Caracterización de E/S

La Tabla 7.1 presenta el número de operaciones de apertura sobre archivos realizadas durante la ejecución del programa, la cantidad de archivos únicos accedidos y el volumen lógico de datos transferidos.

métrica	*valor*
Aperturas exitosas	4490
Descriptores de archivo utilizados para lectura	1960
Descriptores de archivo utilizados para escritura	748
Aperturas sin E/S asociada	1782
Archivos leídos	1093
Archivos escritos	366
GiB leídos	358,64
GiB escritos	409,25

Tabla 7.1: Operaciones de apertura, cantidad de archivos únicos y volumen de datos leídos y escritos de la aplicación de procesamiento de datos para DeepONet.

Casi 40 % de las operaciones de apertura no correspondieron a operaciones de lectura o escritura, por lo que probablemente correspondieron a verificaciones de existencia o lecturas de metadatos, que son interacciones con el MDS y no con los OSS. El número de descriptores utilizados para lectura y escritura fue mayor a la cantidad correspondiente de archivos leídos o escritos porque los mismos archivos se leyeron o escribieron más de una vez. El volumen lógico combinado de datos leídos y escritos fue 767,89 GiB. La carga se distribuyó entre dieciséis procesos, por lo que cada uno procesó, en promedio, aproximadamente 48 GiB de datos. La Tabla 7.2 desglosa las operaciones de lectura y escritura por la extensión de los archivos.

tipo	*extensión*	*número de archivos*	*volumen (GiB)*	*tamaño medio*
lectura	gz	722	358,62	509 MiB
lectura	mat	365	0,02	64 KiB
lectura	m	6	<0,01	224 KiB
escritura	mat	365	409,25	1,12 GiB
escritura	log	1	<0,01	10,5 KiB

Tabla 7.2: Desglose de las operaciones de lectura y escritura de la aplicación de procesamiento de datos para DeepONet por tipo de archivo.

Los 722 archivos gz leídos correspondieron al 99,99 % de los bytes totales leídos y los 365 archivos mat escritos correspondieron al 99,99 % de los datos escritos. Los archivos mat también aparecieron en la lectura y en la misma cantidad, pero con un tamaño medio considerablemente menor; los archivos mat de entrada contuvieron los parámetros y descriptores de cada caso y los archivos mat de salida contuvieron los resultados. Los seis archivos .m leídos correspondieron a los scripts de MATLAB cargados una única vez durante la inicialización del programa.

A partir de la caracterización realizada, se clasificó a la aplicación de procesamiento de datos para DeepONet como una aplicación con carga fuerte de entrada y salida, con lecturas y escrituras de muchos archivos de gran tamaño (desde cientos de MiB a más de un GiB). El patrón es el favorable para Lustre, donde el rendimiento se optimiza con accesos largos y secuenciales. Si bien los archivos de entrada son comprimidos y los de salida no, los volúmenes de datos son del mismo orden de magnitud: a nivel lógico se escriben aproximadamente 1,14 veces más bytes de los que se leen. La diferencia se explica porque la descompresión del formato gzip expande los datos que luego se persisten en formato mat sin una compresión equivalente.

7.1.2. Resultados experimentales

La Tabla 7.3 reporta los tiempos de ejecución de la aplicación de procesamiento de datos para DeepONet sobre cada una de las configuraciones estudiadas. Cada configuración se ejecutó tres veces y se reportan la media y la desviación estándar. Salvo que se indique lo contrario, el tamaño de stripe de Lustre se fijó en 4 MiB y se utilizó el tamaño de RPC por defecto de 4 MiB.

El número de OST fue el parámetro más influyente en el cambio de rendimiento para la aplicación. Aumentar de uno a tres OST redujo el tiempo de ejecución de 11 265 s a 3949 s, una mejora del 65 %. Sin embargo, el cambio en rendimiento no fue uniforme al aumentar el número de OST: aumentar de dos a tres OST redujo casi a la mitad el tiempo medio de ejecución y aumentar de tres a cuatro OST aumentó 4 % el tiempo de ejecución. El cambio de sistema de archivos de base tuvo un efecto moderado en el rendimiento. ZFS sin compresión redujo el tiempo de ejecución 12,7 % respecto a ldiskfs con cuatro OST y la desviación estándar fue considerablemente menor. Habilitar la compresión LZ4 en ZFS no produjo una mejora de rendimiento. Aumentar el número de

sistema de archivos de base	*configuración*	*tiempo de ejecución (s)*	*σ (s)*
	Ejecuciones sin sistema de archivos paralelo		
NFS	4 discos, ext4	5014	205
	4 discos, ZFS+LZ4	3569	10
	6 discos, ext4	4997	454
	6 discos, ZFS+LZ4	5317	198
	Ejecuciones con sistema de archivos paralelo		
ldiskfs	1 OST	11 265	334
	2 OST	7202	168
	3 OST	3949	328
	4 OST	4107	256
ZFS	4 OST	3586	12
ZFS+LZ4	4 OST	3588	9
	Ajuste de parámetros de Lustre		
ldiskfs	4 OST, striping en 1 OST	4510	241
	4 OST, striping en 2 OST	4572	146
	4 OST, striping en 3 OST	4142	138
ldiskfs	4 OST, PFL	3883	221
ZFS	4 OST, PFL	3590	12
ZFS+LZ4	4 OST, PFL	3590	15
ldiskfs	4 OST, large I/O	3709	23
	4 OST, DoM + PFL	3710	104

Tabla 7.3: Tiempo medio de ejecución y desviación estándar de la aplicación de procesamiento de datos para DeepONet sobre las configuraciones de sistema de archivos evaluadas.

stripes en ldiskfs produjo una mejora menor al 10 % al aumentar de uno a cuatro stripes. El resultado fue consistente con la distribución de tamaños de la carga; 93 % de los archivos de salida superaron 1 GiB de tamaño, por lo que incluso con un solo stripe la cola de peticiones de E/S por OST se mantuvo saturada por los escritores concurrentes.

La configuración de striping PFL mejoró el rendimiento en ldiskfs en 5,5 % pero no causó una mejora sobre ZFS con o sin compresión e incluso redujo ligeramente el rendimiento. Los experimentos con RPC de 16 MiB y la configuración DoM+PFL produjeron tiempos de ejecución similares sobre ldiskfs: una mejora de aproximadamente 10 % respecto de la configuración base de cuatro OST con ldiskfs. Utilizar RPC de 16 MiB redujo el número de RPC en un factor de cuatro y fue más efectivo en cargas dominadas por escrituras

secuenciales grandes, que coincide con el perfil de E/S de la aplicación. Tener datos en el MDT aportó una mejora comparable mediante un mecanismo distinto: el primer MiB de cada archivo se sirvió desde el MDT (con almacenamiento en discos sólidos) y favoreció las lecturas de los archivos auxiliares de tamaño reducido.

7.1.3. Análisis de tiempos ponderados, aceleración y eficiencia

La ponderación por heterogeneidad de OST descrita en la Sección 5.3.4 se aplicó a los tiempos de la Tabla 7.3. Se midió la relación entre lecturas y escrituras físicas de los experimentos y, en promedio, la relación lectura/escritura fue 0,87 lecturas por cada escritura. Por lo tanto, la carga se modeló como $n_{\mathrm{r}} = 87$ lecturas y $n_{\mathrm{w}} = 100$ escrituras. Los nodos de referencia fueron benny para las ejecuciones con NFS de cuatro discos y annifrid para las de seis discos y con este perfil los costos de referencia resultaron $C_{\mathrm{benny}} = 118{,}0$ y $C_{\mathrm{annifrid}} = 155{,}4$. El costo agregado $C_{\mathcal{O}}$ de cada configuración se calculó sobre el conjunto de OST que efectivamente recibió la carga: annifrid en la configuración de un OST; annifrid y bjorn en la de dos OST, de acuerdo con el comportamiento anómalo descrito en la Sección 7.1.4; annifrid, benny y agnetha en la de tres OST y los cuatro OST en las restantes. La Tabla 7.4 presenta los tiempos ponderados obtenidos bajo la referencia de benny y los tiempos ponderados obtenidos bajo la referencia de annifrid.

Bajo la referencia de benny, un OST rápido, las configuraciones de cuatro OST redujeron su tiempo respecto del crudo, porque parte de su carga recayó sobre OST más lentos. Bajo la referencia de annifrid, un OST lento, los tiempos ponderados aumentaron de forma uniforme. En la configuración de un OST, ejecutada sobre annifrid, el factor frente a la referencia annifrid fue 1 y el tiempo ponderado coincidió con el crudo.

La comparación entre el sistema de archivos paralelo y NFS se realizó sobre los tiempos ponderados, mediante la comparación de la columna de referencia benny con las ejecuciones con NFS de cuatro discos y la columna de referencia annifrid con las de seis discos. Frente a la referencia de cuatro discos sobre ext4 (5014 s), todas las configuraciones de cuatro OST resultaron más rápidas: el tiempo ponderado de ldiskfs fue 3789 s y las variantes con sistema de archivos de base ZFS se ubicaron entre 3308 y 3312 s, 34 % por debajo. La referencia con

sistema de archivos de base	*configuración*	$T_p^\star$ *(benny) (s)*	$T_p^\star$ *(annifrid) (s)*
ldiskfs	1 OST	8555	11 265
	2 OST	5335	7025
	3 OST	3988	5252
	4 OST	3789	4989
ZFS	4 OST	3308	4356
ZFS+LZ4	4 OST	3310	4359
ldiskfs	4 OST, striping en 1 OST	4161	5479
	4 OST, striping en 2 OST	4218	5554
	4 OST, striping en 3 OST	3821	5032
ldiskfs	4 OST, PFL	3582	4717
ZFS	4 OST, PFL	3312	4361
ZFS+LZ4	4 OST, PFL	3312	4361
ldiskfs	4 OST, large I/O	3422	4506
	4 OST, DoM + PFL	3423	4507

Tabla 7.4: Tiempos ponderados por heterogeneidad de OST de la aplicación de procesamiento de datos para DeepONet.

mejores resultados de NFS, NFS sobre ZFS+LZ4 con cuatro discos (3569 s), solo fue superada por los sistemas de archivos de base ZFS y por las variantes de ldiskfs con large I/O y DoM + PFL. Los escenarios de ldiskfs sin ajustes y las variantes con striping entre un número definido de OST tuvieron un menor rendimiento que el mejor escenario con NFS. Frente a las referencias de seis discos (4997 s sobre ext4 y 5317 s sobre ZFS+LZ4), las configuraciones de cuatro OST basadas en ZFS resultaron entre 13 % y 18 % más rápidas y ldiskfs con cuatro OST igualó a la referencia sobre ext4 (4989 s frente a 4997 s). Las configuraciones de uno y dos OST tuvieron menor rendimiento que todas las referencias sobre NFS. El sistema de archivos paralelo obtuvo un rendimiento similar al de NFS a partir de tres OST y superó el rendimiento de NFS en la mayoría de los escenarios con cuatro OST.

Para cuantificar el efecto del paralelismo de almacenamiento se calcularon la aceleración y la eficiencia al escalar el número de OST. El costo de referencia empleado fue el del único OST utilizado en las ejecuciones de un OST (annifrid), $C_{\text{ref}} = C_{1\,\text{OST}} = 155{,}4$. El análisis se restringió a las configuraciones de uno, dos, tres y cuatro OST sobre ldiskfs y a la configuración de cuatro OST que obtuvo el mejor tiempo de ejecución, ZFS sin compresión.

La Tabla 7.5 presenta la aceleración y la eficiencia de los tiempos ponderados respecto a la configuración de un OST.

sistema de archivos de base	*configuración*	*aceleración*	*eficiencia*
ldiskfs	1 OST	1,00	1,00
	2 OST	1,60	0,80
	3 OST	2,15	0,72
	4 OST	2,26	0,56
ZFS	4 OST	2,59	0,65

Tabla 7.5: Aceleración y eficiencia ponderadas de la aplicación de procesamiento de datos para DeepONet respecto a la configuración de un OST.

La aceleración creció de forma monótona con el número de OST sobre ldiskfs, pero el crecimiento no fue lineal. La mayor parte de la ganancia se obtuvo al pasar de uno a tres OST y el cuarto OST aportó una mejora marginal, consistente con el traslado del cuello de botella hacia el OST de menor rendimiento que se analiza en la Sección 7.1.4. La mejor configuración de cuatro OST, ZFS sin compresión, alcanzó una aceleración de 2,59 y una eficiencia de 0,65, por encima de la configuración de cuatro OST sobre ldiskfs. La eficiencia no fue lineal en todas las configuraciones y decreció de forma monótona al aumentar la cantidad de OST, por lo que aun en la mejor configuración de cuatro OST se llegó a apenas 65 % del escalamiento ideal.

Los resultados mostraron que el sistema de archivos paralelo superó a la referencia NFS sobre ext4 en todas las configuraciones de cuatro OST y superó a la mejor referencia NFS solo en las variantes de base ZFS y en large I/O y DoM+PFL. Los mejores resultados se obtuvieron con las configuraciones basadas en ZFS y con los ajustes PFL y large I/O sobre ldiskfs, aunque con un margen de mejora limitado al agregar OST.

7.1.4. Análisis de uso de disco y CPU

Las diferencias de rendimiento entre configuraciones pueden explicarse en parte a partir del comportamiento de E/S durante la ejecución. En esta sección se analiza la utilización del disco y la CPU en varias configuraciones con el objetivo de identificar los cuellos de botella y contrastar el comportamiento de ldiskfs y ZFS.

La Figura 7.1 muestra la tasa agregada de lectura y escritura entre todos los OSS a lo largo del tiempo de ejecución en ldiskfs con cuatro OST.

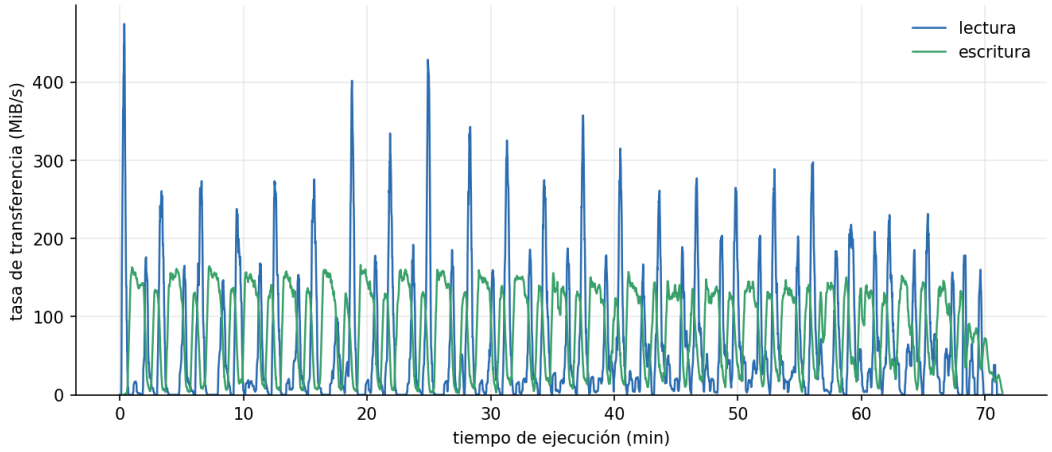


Figura 7.1: Tasa de transferencia agregada de lectura y escritura en los OSS para la aplicación de procesamiento de datos para DeepONet a través del tiempo.

El patrón de E/S no fue uniforme. La diferencia de tiempo entre lecturas y escrituras se explica porque, si bien se leen y se escriben volúmenes de datos similares, la lectura fue más rápida que la escritura. El comportamiento de ráfagas de lectura y posterior escritura refleja el procesamiento: cada trabajador lee un archivo gz, procesa el archivo y escribe el resultado.

La Figura 7.2 muestra el porcentaje de utilización del disco lógico RAID 1+0 del OST bjorn a lo largo de la ejecución de la aplicación para las configuraciones de cuatro OST con ldiskfs, ZFS y ZFS+LZ4.

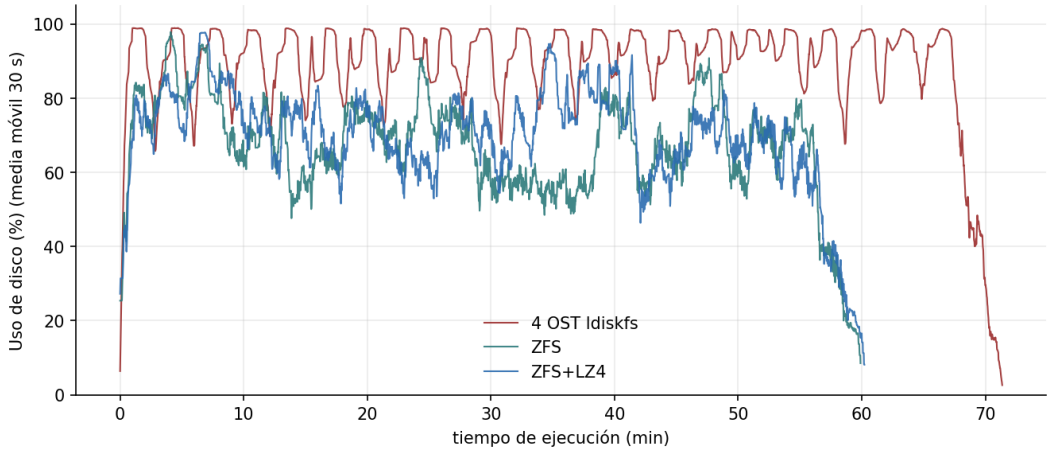


Figura 7.2: Porcentaje de utilización del disco lógico RAID 1+0 del OST bjorn a lo largo de la ejecución de la aplicación (media móvil 30 segundos).

La ejecución con el sistema de archivos de base ldiskfs mantuvo una utilización cercana al 100 % durante casi todas las ráfagas de escritura, por lo que el almacenamiento fue un cuello de botella en la configuración. ZFS y ZFS+LZ4 presentaron una utilización considerablemente menor y más variable, por lo que el sistema de archivos de base ZFS logró mantener el disco menos saturado y con mayor capacidad de respuesta ante nuevas solicitudes. Mantener el dispositivo menos tiempo saturado puede reducir las colas de E/S, mejorar la latencia efectiva de las operaciones y posiblemente contribuya a los menores tiempos de ejecución con ZFS y ZFS+LZ4 frente a ldiskfs.

La Tabla 7.6 presenta el tamaño físico medio de operación en los OSS en las configuraciones con ldiskfs, ZFS y ZFS+LZ4 con cuatro OST.

sistema de archivos de base	*lectura (KiB)*	*escritura (KiB)*
ldiskfs	3991	745
ZFS	1019	426
ZFS+LZ4	1018	376

Tabla 7.6: Tamaño medio de operación de lectura y escritura en los OST para ldiskfs, ZFS y ZFS+LZ4 (cuatro OST).

El tamaño medio de operación de lectura en ldiskfs fue 3991 KiB, muy cercano al límite de RPC (4 MiB), en ZFS fue 1019 KiB y en ZFS+LZ4 fue 1018 KiB, muy cercano al tamaño de bloque lógico configurado en ZFS (1 MiB). El tamaño medio de operación de escritura siguió el mismo patrón: 745 KiB en ldiskfs frente a 426 KiB en ZFS y 376 KiB en ZFS+LZ4.

La Figura 7.3 muestra la utilización media del disco de cada OST al escalar de uno a cuatro OST con ldiskfs y ayuda a explicar el escalado del tiempo de ejecución de la aplicación.

Con un OST, el único dispositivo activo (annifrid) se mantuvo saturado con una utilización media de 92 %, por lo que el dispositivo se convirtió en el cuello de botella de la configuración y es consistente con el tiempo más alto y la mayor variabilidad. Al pasar a dos OST se presentó un comportamiento anómalo en el sistema de archivos: si bien el striping se definió entre annifrid y agnetha, por un error de causas desconocidas de Lustre las escrituras se dirigieron a annifrid y bjorn. En consecuencia, agnetha permaneció inactivo, annifrid redujo su utilización al 50 % y bjorn se saturó con 95 % de uso medio de disco, convirtiéndose en el cuello de botella.

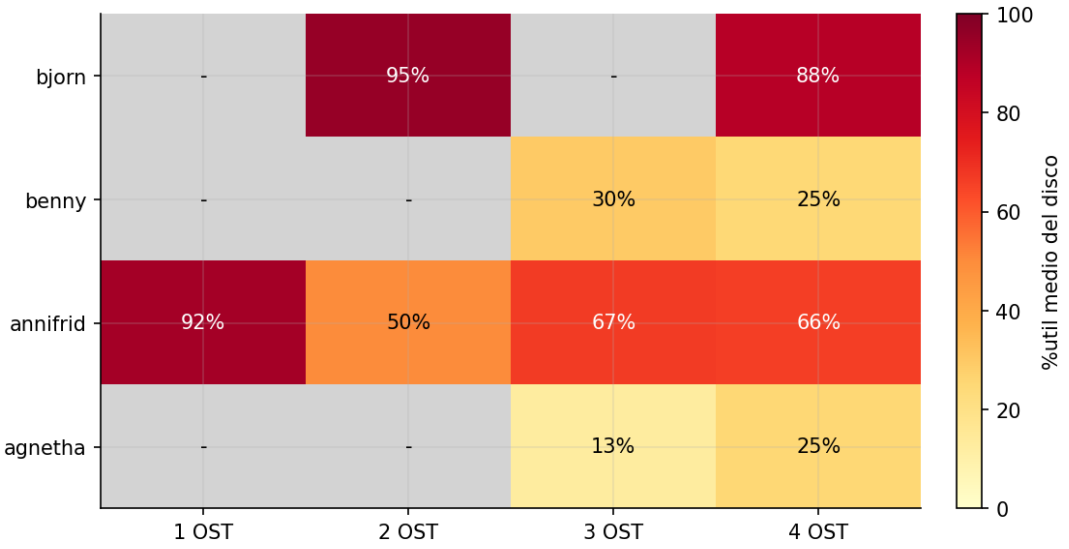


Figura 7.3: Utilización media del disco de cada OST al escalar de uno a cuatro OST con ldiskfs.

Aun al recibir la mitad de la carga, bjorn saturó su disco y limitó el rendimiento de todo el sistema de archivos paralelo. En el sistema de archivos se verificó la configuración de striping para este escenario y resultó correctamente definida, por lo que no se dio por un error de configuración. La anomalía solo se produjo en el caso de dos OST; en las configuraciones de tres y cuatro OST los datos se distribuyeron entre los OST especificados. A partir de tres OST la utilización media no superó 67 % y el cuello de botella volvió a ser annifrid, seguido por benny y agnetha. La reducción de la saturación de los discos se correspondió con la mejora del tiempo de ejecución ocurrida al pasar de dos a tres OST. Al incorporar el cuarto OST el cuello de botella se trasladó nuevamente a bjorn, que alcanzó 88 % de utilización media de disco, en línea con ser el nodo con el almacenamiento de menor rendimiento según los benchmarks. El resto de los OST quedaron con sus discos no sobrecargados.

La Figura 7.4 muestra el porcentaje de uso de CPU en el OST bjorn a lo largo de la ejecución de la aplicación para el caso ZFS y ZFS+LZ4.

La tendencia de uso de CPU en ambos escenarios fue similar, pero en ZFS+LZ4 fue igual o mayor en la mayor parte de los casos. El resultado es esperable, dado que aplicar compresión a los datos genera un uso adicional de CPU comparado a no comprimir. En ningún momento la CPU se convirtió en un cuello de botella: el porcentaje de uso promedio de CPU fue 8,25 % en el escenario con ZFS y 9 % en el escenario con ZFS+LZ4.

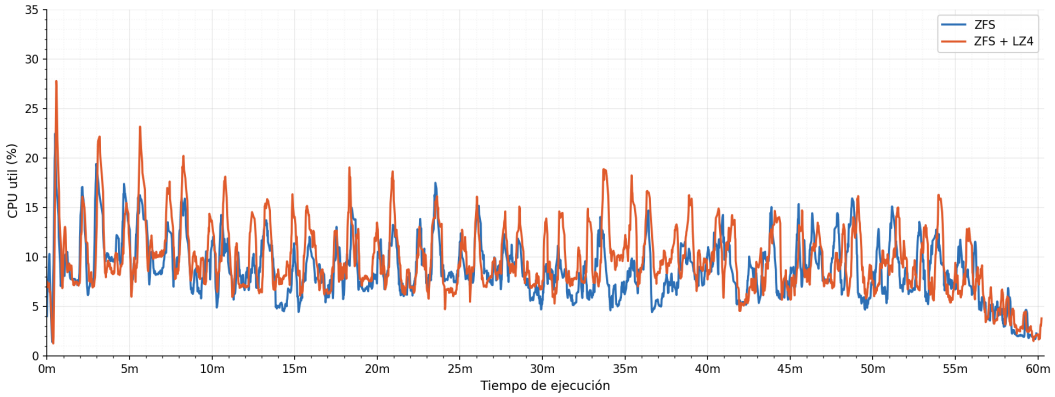


Figura 7.4: Porcentaje de uso de CPU en el OST bjorn a lo largo de la ejecución de la aplicación, ZFS y ZFS+LZ4.

La Figura 7.5 presenta la distribución promedio del uso de CPU del cliente para las distintas configuraciones.

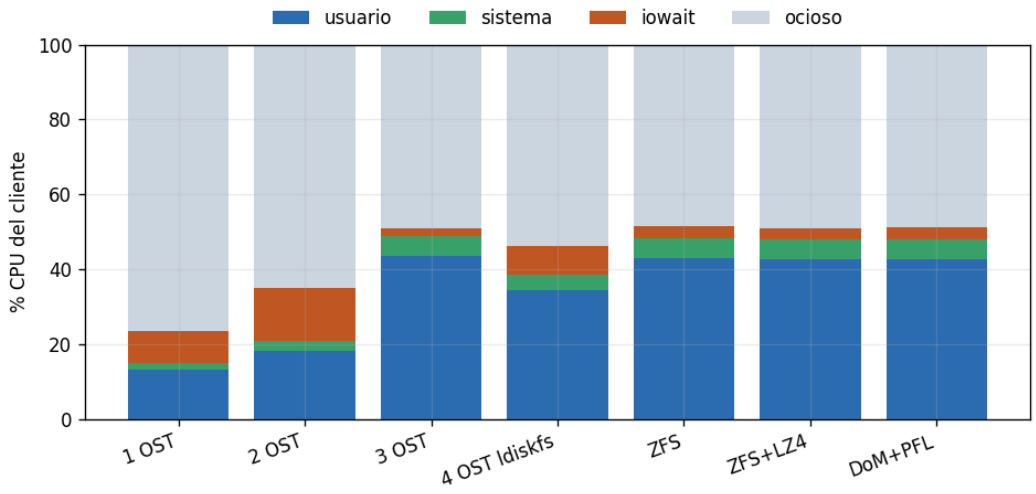


Figura 7.5: Desglose promedio de uso de CPU en el cliente a lo largo de la ejecución de la aplicación.

Con uno y dos OST el porcentaje de espera por E/S representó una fracción significativa del tiempo de CPU y el porcentaje de uso en modo usuario fue bajo, por lo que los procesos trabajadores pasaron un tiempo considerable bloqueados a la espera de que se completaran las operaciones de E/S. Con tres OST el porcentaje de espera por E/S se redujo y el porcentaje de tiempo de uso en modo usuario aumentó, por lo que se redujo el cuello de botella causado por el almacenamiento. Con cuatro OST con ldiskfs el porcentaje de uso en modo usuario se redujo y el porcentaje de espera por E/S aumentó

(en comparación con tres OST), por lo que agregar un recurso adicional de almacenamiento tuvo un efecto contraproducente. El OST agregado al pasar de tres OST a cuatro OST fue el de menor rendimiento, por lo que escribir las partes de los archivos en el nuevo OST resultó más costoso que utilizar solo tres OST. Las configuraciones ZFS, ZFS+LZ4 y DoM+PFL presentaron un perfil de uso de CPU similar al de tres OST, con una utilización total de aproximadamente 50 % y una fracción de espera por E/S reducida, consistente con los tiempos de ejecución similares obtenidos.

7.2. Procesamiento de datos de viajes de transporte público

En esta sección se presentan los resultados de la caracterización del comportamiento de E/S de la aplicación de procesamiento de datos de transporte público urbano obtenidos con la metodología descrita en la Sección 5.4.1 y a continuación se evalúa su rendimiento sobre las distintas configuraciones de Lustre estudiadas.

7.2.1. Caracterización de E/S

La aplicación procesa un conjunto de registros de viajes contenido en un único archivo de entrada de 50,87 GiB. El procesamiento consiste en dos etapas que se ejecutan de forma secuencial. La primera, de particionado, distribuye los registros del archivo de entrada en cien archivos según la tarjeta de transporte utilizada en cada viaje. La segunda, de ordenamiento, reordena cada una de las particiones agrupando los registros de una misma tarjeta y produce cien archivos de salida.

La Tabla 7.7 contiene información de las operaciones de apertura de archivos, la cantidad de archivos únicos y el volumen lógico de datos leídos y escritos durante la ejecución del programa.

Casi la totalidad de las aperturas (96 %) correspondieron a descriptores utilizados para escritura, aunque solo se escribieron 304 archivos distintos. La diferencia se explica porque en la etapa de particionado se reabre cada archivo de partición para agregar registros en vez de abrir cada uno una única vez durante toda la ejecución.

métrica	*valor*
Aperturas exitosas	856397
Descriptores de archivo utilizados para lectura	10976
Descriptores de archivo utilizados para escritura	825402
Aperturas sin E/S asociada	20019
Archivos de datos leídos	201
Archivos de datos escritos	304
Datos leídos	2,03 TiB
Datos escritos	89,51 GiB

Tabla 7.7: Operaciones de apertura, cantidad de archivos únicos y volumen de datos leídos y escritos de la aplicación de procesamiento de datos de transporte público.

La Tabla 7.8 desglosa los volúmenes lógicos de lectura y escritura por etapa y conjunto de archivos.

etapa	*conjunto de archivos*	*archivos*	*leído*	*escrito*
particionado	archivo de entrada	1	50,87 GiB	—
particionado	particiones por tarjeta	100	—	43,57 GiB
ordenamiento	particiones por tarjeta	100	1,98 TiB	—
ordenamiento	índices	100	2,36 GiB	2,36 GiB
ordenamiento	archivos de salida	100	—	43,57 GiB

Tabla 7.8: Desglose de los volúmenes de lectura y escritura de la aplicación de procesamiento de datos de transporte público por etapa y conjunto de archivos.

La etapa de particionado leyó el archivo de entrada de forma secuencial y escribió las cien particiones, cada una de 446 MiB en promedio. La cantidad de datos leída y escrita fue del mismo orden de magnitud y la diferencia se dio porque el particionado descarta los registros sin tarjeta asociada. La etapa de ordenamiento presentó una gran diferencia en la cantidad de datos leída y escrita: se leyeron 1,98 TiB aunque las particiones en total contenían solo 43,57 GiB de datos. La diferencia se debe al patrón de lectura de los datos: como los registros de una misma tarjeta están repartidos a través de todo el archivo, el ordenamiento accede a los registros no secuencialmente, saltando entre posiciones lejanas del archivo. Por el comportamiento de las funciones de Python seek y readline y la implementación del programa, cada lectura en cada salto transfiere un bloque de tamaño fijo, mayor que el registro que se busca leer; al saltar repetidamente sobre un mismo archivo, las regiones se solapan y los mismos datos se leen varias veces. Como se leen constantemente

los mismos datos y el cliente y/o los OSS mantienen en caché los bloques accedidos recientemente, la lectura real sobre los OSS probablemente no sea la misma. A partir de la caracterización realizada, se clasificó a la aplicación de procesamiento de datos de transporte público como una aplicación intensiva en E/S.

7.2.2. Resultados experimentales

Para evaluar el rendimiento de la aplicación de procesamiento de datos de transporte público sobre las distintas configuraciones de Lustre se siguió la misma metodología empleada en la aplicación de procesamiento de datos para DeepONet; cada configuración se ejecutó tres veces y se reportan la media y la desviación estándar del tiempo de ejecución. Salvo que se indique lo contrario, el tamaño de stripe se fijó en 4 MiB y se utilizó el tamaño de RPC por defecto (4 MiB). La Tabla 7.9 reporta los tiempos de ejecución de la aplicación sobre cada una de las configuraciones estudiadas.

El número de OST fue el parámetro más influyente sobre el rendimiento. Con el sistema de archivos de base ldiskfs, el tiempo de ejecución se redujo de forma monótona al aumentar el número de OST, de 1675 s con un OST a 633 s con cuatro OST. La mayor parte de la ganancia se concentró al pasar de uno a tres OST (59,7 %) y agregar el cuarto OST aportó una mejora de 6,2 %. Además, el tiempo de las ejecuciones con tres y cuatro OST fue más estable en comparación con uno y dos OST.

Con cuatro OST, ZFS sin compresión redujo el tiempo de ejecución 9,0 % respecto a ldiskfs y habilitar la compresión en ZFS produjo una reducción de tiempo de 12,2 % respecto a ldiskfs. ZFS con compresión fue la configuración más rápida y la de menor variación de todas las configuraciones paralelas evaluadas. A diferencia de los resultados de la aplicación de procesamiento de datos para DeepONet (Sección 7.1), donde la compresión no aportó mejoras por la baja compresibilidad de los datos, en esta aplicación la compresión sí mejoró el tiempo de ejecución, ya que los datos de los viajes son compresibles. Sin embargo, la mejora de tiempo fue baja en comparación con la reducción de la carga de disco (se presenta en la Sección 7.2.3) que produjo la compresión.

Ninguna de las configuraciones de ajuste de parámetros de Lustre mejoró sobre la configuración base de cuatro OST con ldiskfs. Variar el número de stripes y aplicar una configuración escalonada de striping resultó en ejecucio-

sistema de archivos de base	*configuración*	*tiempo de ejecución (s)*	*σ (s)*
	Ejecuciones sin sistema de archivos paralelo		
NFS	4 discos, ext4	654	12
	4 discos, ZFS+LZ4	467	4
	6 discos, ext4	524	5
	6 discos, ZFS+LZ4	541	2
	Ejecuciones con sistema de archivos paralelo		
ldiskfs	1 OST	1675	400
	2 OST	937	152
	3 OST	675	40
	4 OST	633	38
ZFS	4 OST	576	31
ZFS+LZ4	4 OST	556	6
	Ajuste de parámetros de Lustre		
ldiskfs	4 OST, striping en 1 OST	1240	47
	4 OST, striping en 2 OST	1453	358
	4 OST, striping en 3 OST	1328	284
ldiskfs	4 OST, PFL	1118	381
ZFS	4 OST, PFL	563	10
ZFS+LZ4	4 OST, PFL	580	27
ldiskfs	4 OST, large I/O	939	19
	4 OST, DoM + PFL	847	15

Tabla 7.9: Tiempo medio de ejecución y desviación estándar de la aplicación de procesamiento de datos de transporte público sobre las configuraciones del sistema de archivos evaluadas.

nes más lentas que la ejecución con ldiskfs en cuatro OST y sus desviaciones estándar fueron elevadas. Las configuraciones large I/O y la configuración DoM+PFL también resultaron más lentas que la configuración de base. Los mejores tiempos de ejecución se obtuvieron con las configuraciones basadas en ZFS. El mejor tiempo correspondió a ZFS+LZ4 con cuatro OST, 12,2 % menor que el tiempo de ldiskfs con cuatro OST.

Durante los experimentos se detectó un comportamiento anómalo de Lustre: al ejecutar la etapa de particionado, el cliente de Lustre no respetaba la configuración de striping definida en el directorio para los archivos de particiones creados, sino que aplicaba una distribución distinta, en varias ocasiones asignando los objetos de los archivos a un solo OST o asignando objetos de los archivos a otros OST sin seguir un patrón específico, independientemente del

stripe count, el stripe size y el conjunto de OST configurados sobre el directorio. En los experimentos de uno, dos y tres OST, donde la configuración de striping del directorio era el factor que restringía los OST a utilizar, este comportamiento impedía obtener resultados correctos. Luego de consultar con la comunidad de Lustre a través de la lista de correos lustre-discuss, se descubrió que el comportamiento dependía de cómo se creaba el archivo y no del modo en que se escribía sobre él. Un archivo creado en modo de escritura y reabierto luego en modo append conservaba la configuración de striping heredada del directorio; en cambio, un archivo creado directamente en append no seguía la configuración de striping heredada.

En la etapa de particionado del programa, cada archivo de partición se abría con la función open de Python en modo append y como la primera apertura crea el archivo en modo append, todas las particiones quedaban con una distribución de stripes incorrecta. El problema se resolvió mediante la creación inicial de cada archivo en modo de escritura, seguida del cierre y la reapertura del archivo en modo append. Al existir el archivo previamente, se conservaba la distribución de stripes heredada del directorio. La versión 2.17 de Lustre resolvió el comportamiento anómalo y eliminó la necesidad de la solución manual, pero como la versión instalada en los nodos fue la 2.15.7 se debió recurrir a la solución alternativa descrita. Todos los experimentos reportados en esta sección se ejecutaron con el programa corregido, por lo que los archivos de partición respetaron la configuración de striping configurada y los resultados reflejaron las configuraciones previstas.

7.2.3. Análisis de tiempos ponderados, aceleración y eficiencia

La misma ponderación por heterogeneidad de OST descrita en la Sección 5.3.4 se aplicó a los tiempos de la Tabla 7.9. Se midió la relación entre lecturas y escrituras físicas de los experimentos y, en promedio, la relación lectura/escritura fue 0,71 lecturas por cada escritura. Por lo tanto, la carga se modeló como $n_{\mathrm{r}} = 71$ lecturas y $n_{\mathrm{w}} = 100$ escrituras. El costo de referencia resultó $C_{\mathrm{benny}} = 104{,}3$ y $C_{\mathrm{annifrid}} = 145{,}2$. El costo agregado $C_{\mathcal{O}}$ de cada configuración se calculó sobre el conjunto de OST que recibió la carga: annifrid en la configuración de un OST, annifrid y agnetha en la de dos OST, annifrid, agnetha y benny en la de tres OST y los cuatro OST en las restantes.

La Tabla 7.10 presenta los tiempos ponderados obtenidos bajo la referencia de benny y la referencia de annifrid.

sistema de archivos de base	*configuración*	$T_p^\star$ *(benny) (s)*	$T_p^\star$ *(annifrid) (s)*
ldiskfs	1 OST	1204	1675
	2 OST	922	1283
	3 OST	659	917
	4 OST	570	793
ZFS	4 OST	518	721
ZFS+LZ4	4 OST	500	696
ldiskfs	4 OST, striping en 1 OST	1116	1553
	4 OST, striping en 2 OST	1307	1819
	4 OST, striping en 3 OST	1195	1663
ldiskfs	4 OST, PFL	1006	1400
ZFS	4 OST, PFL	507	705
ZFS+LZ4	4 OST, PFL	522	726
ldiskfs	4 OST, large I/O	845	1176
	4 OST, DoM + PFL	762	1061

Tabla 7.10: Tiempos ponderados por heterogeneidad de OST de la aplicación de procesamiento de datos de transporte público.

Las ejecuciones sobre NFS fueron muy rápidas. Frente a la ejecución con NFS en benny sobre ext4 (654 s), las configuraciones de cuatro OST resultaron más rápidas: 570 s con ldiskfs y entre 500 y 522 s con los sistemas de archivos de base ZFS. Ningún tiempo de una configuración paralela alcanzó a la ejecución con NFS en benny sobre ZFS+LZ4, que fue especialmente veloz (467 s); el tiempo de la configuración paralela que más se acercó fue ZFS+LZ4 con cuatro OST (500 s), 7 % más lenta. El mejor tiempo ponderado, de 696 s con ZFS+LZ4 y cuatro OST, también resultó 33 % mayor que el tiempo de NFS sobre ext4 y 29 % mayor que el de NFS sobre ZFS+LZ4 en annifrid. Los tiempos de las configuraciones de uno y dos OST y todas las variantes de ajuste de parámetros sobre ldiskfs tuvieron un menor rendimiento que todas las ejecuciones con NFS. Todos los escenarios de ajuste de parámetros sobre ldiskfs presentaron tiempos ponderados muy superiores al de la configuración de cuatro OST sin ajustes y los sistemas de archivos de base ZFS mantuvieron su rendimiento con y sin PFL.

Para cuantificar el efecto del paralelismo del sistema de almacenamiento se calcularon la aceleración y la eficiencia al escalar el número de OST, como se describe en la Sección 5.3.5. El costo de referencia empleado fue el del único OST utilizado (annifrid), $C_{\text{ref}} = C_{1\,\text{OST}} = 145{,}2$. El análisis se restringió a la serie de configuraciones de uno, dos, tres y cuatro OST sobre ldiskfs y a la configuración de cuatro OST que obtuvo el mejor tiempo de ejecución, ZFS con compresión. La Tabla 7.11 presenta la aceleración y la eficiencia resultantes.

sistema de archivos de base	*configuración*	*aceleración*	*eficiencia*
ldiskfs	1 OST	1,00	1,00
	2 OST	1,31	0,65
	3 OST	1,83	0,61
	4 OST	2,11	0,53
ZFS+LZ4	4 OST	2,41	0,60

Tabla 7.11: Aceleración y eficiencia ponderadas de la aplicación de procesamiento de datos de transporte público respecto a la configuración de un OST.

La aceleración creció de forma monótona y la eficiencia decreció al aumentar el número de OST. La mejor configuración de cuatro OST, ZFS con compresión, superó en aceleración y eficiencia a la configuración de cuatro OST sobre ldiskfs y se alcanzó una aceleración de 2,41× y 60 % del escalamiento ideal. La diferencia obtenida al introducir la compresión es consistente con la alta compresibilidad de los datos de viajes.

Solo las configuraciones de cuatro OST superaron a la ejecución más lenta sobre NFS y ninguna de las configuraciones paralelas alcanzó el rendimiento de las otras tres ejecuciones sobre NFS. En la mayoría de los escenarios, el sistema de archivos paralelo no obtuvo mejoras significativas o tuvo un menor rendimiento respecto al acceso vía NFS.

7.2.4. Análisis de uso de disco y CPU

Las diferencias de rendimiento entre configuraciones pueden explicarse en parte a partir del patrón de E/S presentado durante las ejecuciones. La caracterización de la Sección 7.2.1 clasificó a la aplicación de procesamiento de datos de transporte público como una aplicación intensiva en E/S, compuesta por dos etapas, un particionado de accesos secuenciales grandes y un ordenamiento de pequeñas lecturas dispersas.

La Figura 7.6 muestra la tasa de transferencia agregada de lectura y escritura entre los OSS a lo largo de la ejecución para la configuración ldiskfs con cuatro OST. Ocurrió un pico inicial que superó los 1200 MiB/s correspondiente a la lectura secuencial del archivo de entrada al comienzo de la etapa de particionado y un segundo pico de magnitud similar más adelante en la ejecución. Fuera de los picos de 1200 MiB/s, la lectura y la escritura oscilaron principalmente entre 100 y 200 MiB/s, con picos intermedios que alcanzaron valores cercanos a 600 MiB/s (lectura) y 300 MiB/s (escritura).

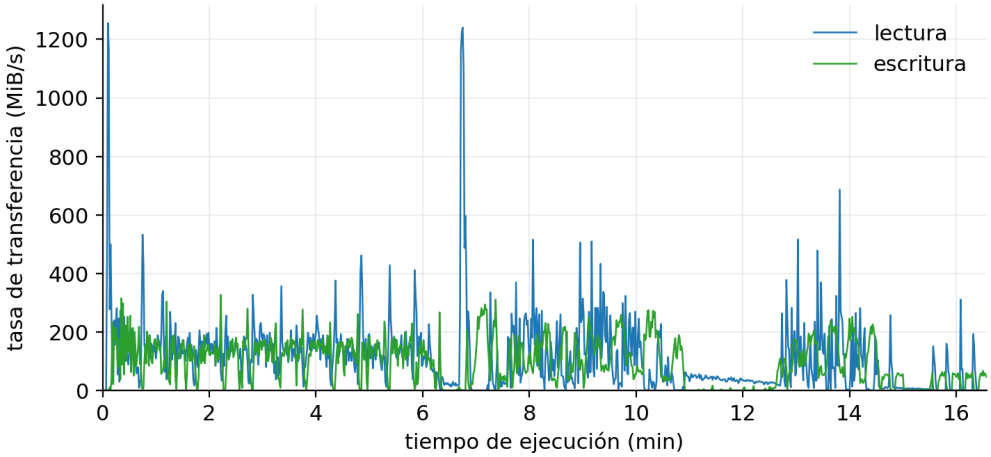


Figura 7.6: Tasa de transferencia agregada de lectura y escritura en los OSS para la aplicación de procesamiento de datos de transporte público a través del tiempo.

La Figura 7.7 muestra la utilización media del disco de cada OST al escalar de uno a cuatro OST con ldiskfs.

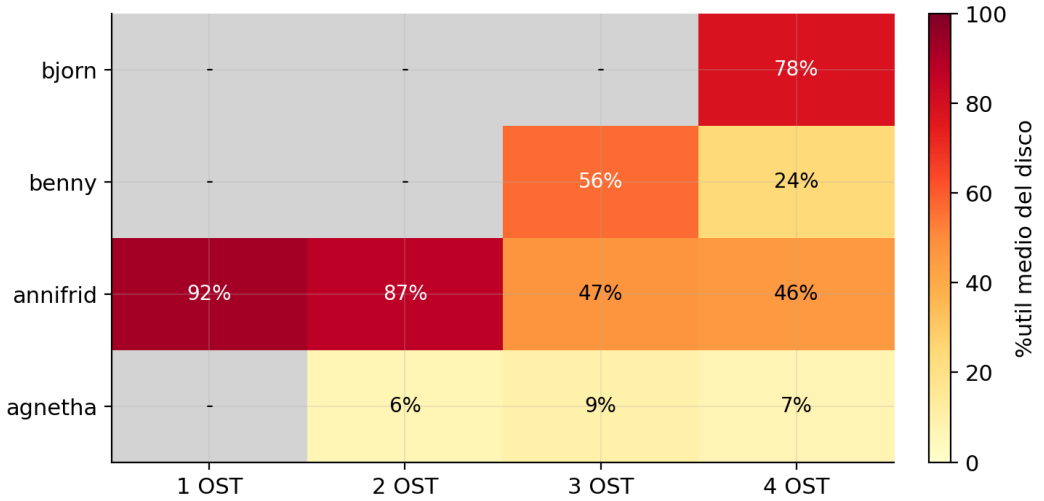


Figura 7.7: Utilización media del disco al escalar de uno a cuatro OST con ldiskfs.

Con un OST, el único dispositivo activo (annifrid) se mantuvo saturado con una utilización media de 92 %, por lo que el dispositivo se convirtió en el cuello de botella de la configuración y es consistente con el tiempo más alto de ejecución. Al pasar a dos OST la situación apenas cambió. La carga se repartió de forma equitativa y annifrid permaneció saturado y el OST agregado se mantuvo con una muy baja utilización. La diferencia se explica porque, como los discos de un nodo tienen menor rendimiento que los del otro, repartir la carga equitativamente entre ambos nodos causó que el nodo más lento se volviera un cuello de botella y limitara el rendimiento de todo el sistema de archivos paralelo.

A partir de tres OST la utilización media no superó 60 % y el cuello de botella pasó de annifrid a benny. La reducción de la saturación de los discos se correspondió con la mejora de tiempo de ejecución ocurrida al pasar de dos a tres OST. Al incorporar el cuarto OST el cuello de botella se trasladó a bjorn, que resulta consistente con que bjorn era el nodo más lento de toda la configuración según los resultados de los benchmarks. El resto de OST quedaron con sus discos no sobrecargados. A diferencia del comportamiento de la aplicación de procesamiento de datos para DeepONet, agregar un cuarto OST redujo el tiempo de ejecución de la aplicación, aunque marginalmente.

La Figura 7.8 muestra el desglose del uso promedio de CPU del cliente para las distintas configuraciones.

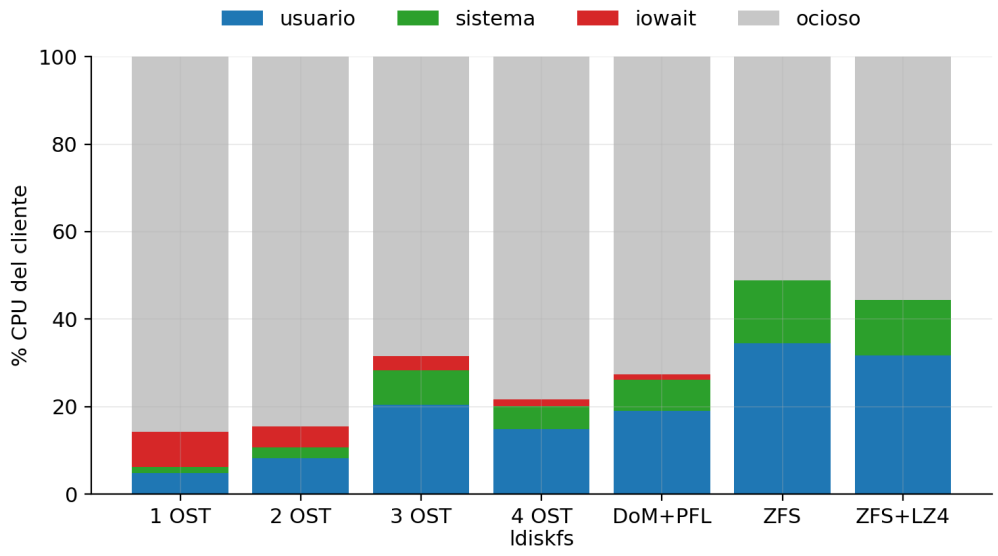


Figura 7.8: Desglose del uso promedio de CPU del cliente para las distintas configuraciones evaluadas.

Con uno y dos OST la utilización total fue baja y el tiempo de espera por E/S representó una fracción considerable del tiempo de CPU. El tiempo en modo usuario fue reducido. Los procesos pasaron una parte considerable del tiempo bloqueados a la espera de que se completaran las operaciones de E/S, coherente con la saturación del único OST de la Figura 7.7 y con los altos tiempos de ejecución. Con tres OST el tiempo de espera por E/S cayó y el tiempo en modo usuario aumentó hasta superar 20 %, en línea con la reducción del cuello de botella de almacenamiento. Las configuraciones basadas en ZFS presentaron el mayor uso total de CPU del cliente y la menor fracción de tiempo de espera por E/S.

La Tabla 7.12 presenta el tamaño medio de operaciones físicas de lectura y escritura en el OST bjorn para las configuraciones de cuatro OST con ldiskfs, ZFS y ZFS+LZ4.

sistema de archivos de base	*lectura (KiB)*	*escritura (KiB)*
ldiskfs	85	160
ZFS	1006	284
ZFS+LZ4	200	252

Tabla 7.12: Tamaño medio de operaciones físicas de lectura y escritura en el OST bjorn para ldiskfs, ZFS y ZFS+LZ4 (cuatro OST).

En ldiskfs las operaciones fueron pequeñas: 85 KiB en lectura y 160 KiB en escritura, muy por debajo del límite de RPC (4 MiB). La diferencia con las lecturas cercanas a 4 MiB de la aplicación de procesamiento de datos para DeepONet se explica porque las lecturas pequeñas y dispersas de la etapa de ordenamiento son en proporción mucho más numerosas que los accesos secuenciales grandes de la etapa de partición, por lo que el promedio es reducido. En ZFS la lectura media fue 1006 KiB, valor muy cercano al tamaño de bloque lógico configurado (1 MiB). ZFS, al realizar una lectura, trae del disco el bloque lógico completo aun cuando la lectura lógica solicitada sea menor. La lectura del bloque lógico completo explica el resultado obtenido. Con compresión el tamaño medio de lectura fue cinco veces menor, posiblemente porque los bloques comprimidos ocupan menos espacio físico en disco y al traer datos del disco se trae el bloque lógico comprimido. La tendencia en escritura fue análoga, aunque con una menor diferencia.

La Figura 7.9 muestra el porcentaje de utilización del disco de bjorn a lo largo de la ejecución para ZFS y ZFS+LZ4.

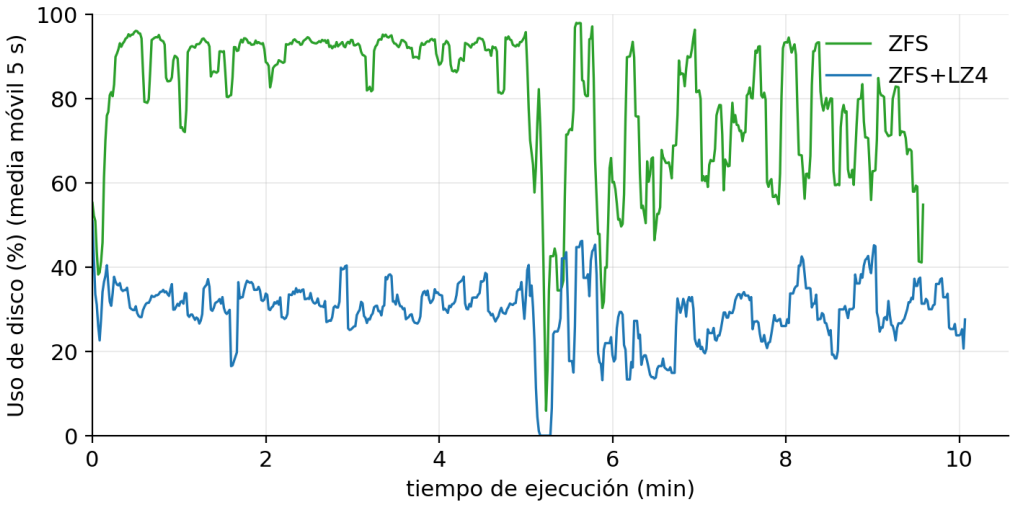


Figura 7.9: Porcentaje de utilización del disco lógico del OST bjorn a lo largo de la ejecución para ZFS y ZFS+LZ4 (media móvil 5 segundos).

ZFS mantuvo el disco mayormente saturado, con una utilización generalmente entre 60 % y 100 % y en ZFS+LZ4 la utilización del disco osciló generalmente entre 20 % y 40 %. La compresión redujo el volumen de datos que se leyeron y escribieron, por lo que el dispositivo pasó menos tiempo saturado. En ldiskfs con cuatro OST, el disco de bjorn también mantuvo una utilización media elevada, cercana a 78 % (Figura 7.7).

La Tabla 7.13 muestra la latencia media de E/S del disco de bjorn por sistema de archivos de base.

sistema de archivos de base	*lectura (ms)*	*escritura (ms)*
ldiskfs	19,3	26,6
ZFS	28,1	19,6
ZFS+LZ4	6,8	16,8

Tabla 7.13: Latencia media de E/S del disco del OST bjorn por sistema de archivos de base.

ZFS+LZ4 logró la menor latencia de lectura y la menor latencia de escritura de los tres sistemas de archivos de base evaluados. La baja latencia de lectura de ZFS+LZ4 resultó coherente con el menor volumen físico leído y con la menor saturación del disco. ZFS sin compresión presentó la mayor latencia de

lectura de los tres sistemas de archivos de base evaluados. La mayor latencia de lectura de ZFS sin compresión puede ser atribuible a que trajo del disco bloques lógicos enteros de 1 MiB con el disco saturado.

La Tabla 7.14 compara el volumen físico de datos escritos y leídos del pool ZFS entre ZFS sin y con compresión.

sistema de archivos de base	*lectura física (GiB)*	*escritura física (GiB)*
ZFS	62	140
ZFS+LZ4	9	46

Tabla 7.14: Volumen físico de datos escritos y leídos en disco para ZFS y ZFS+LZ4.

La compresión LZ4 redujo el volumen físico leído de 62 GiB a 9 GiB. Además, redujo el volumen físico escrito de 140 GiB a 46 GiB. La reducción del volumen de datos leído y escrito se da por la alta compresibilidad de los datos de viajes utilizados. La reducción del tráfico físico de disco explica la menor utilización y la menor latencia de lectura en la configuración ZFS+LZ4. La escritura física de 140 GiB fue mayor a la escritura lógica del programa; una explicación posible es que como ZFS utiliza copy-on-write cualquier modificación parcial de un bloque lógico de 1 MiB implica reescribir el bloque completo en una nueva ubicación y causa el aumento de escrituras físicas.

La Figura 7.10 muestra el tiempo de CPU en modo núcleo del OSS bjorn a lo largo de la ejecución para ZFS y ZFS+LZ4.

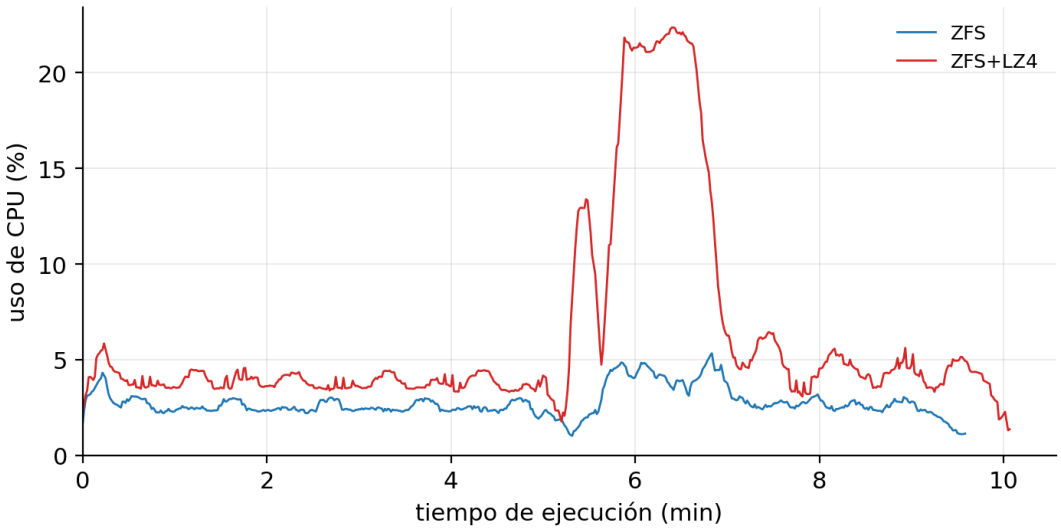


Figura 7.10: Tiempo de CPU en modo núcleo del OSS bjorn a lo largo de la ejecución de la aplicación para ZFS y ZFS+LZ4.

El tiempo de núcleo fue mayor con el sistema de archivos de base ZFS+LZ4, con un pico de aproximadamente 22 % y en ZFS sin compresión se mantuvo en valores bajos, menores a 5 % con la excepción de un pico que superó levemente ese valor. La diferencia se explica por el costo de CPU ocasionado por la compresión. Las operaciones de compresión ejecutan en modo núcleo y en consecuencia cuentan en el tiempo de núcleo. La CPU del OSS no se convirtió en cuello de botella en ningún momento, por lo que la reducción del tráfico de disco compensó el costo de CPU ocasionado por comprimir y descomprimir los datos.

El patrón de E/S de la aplicación de procesamiento de datos de transporte público hizo que el comportamiento del almacenamiento dominara el rendimiento, sobre todo en las configuraciones con pocos OST. Escalar el número de OST en ldiskfs repartió la carga y redujo el tiempo de espera por E/S, con la mayor parte de la ganancia entre uno y tres OST. El sistema de archivos de base ZFS redujo la saturación del disco y, gracias a la alta compresibilidad de los datos de viajes, ZFS+LZ4 redujo el tráfico físico de disco y alcanzó la menor latencia de lectura y la menor utilización de disco de todas las configuraciones evaluadas, a un costo de CPU del OSS que nunca llegó a ser limitante. La menor utilización de disco y la menor latencia resultaron consistentes con los tiempos de ejecución de la Tabla 7.9, donde las configuraciones basadas en ZFS obtuvieron los mejores resultados.

7.3. Calibración del modelo DayCent

En esta sección se presentan los resultados de la evaluación de rendimiento de la aplicación de calibración del modelo DayCent sobre las distintas configuraciones de Lustre estudiadas. Al ejecutarse el modelo DayCent mediante la capa de compatibilidad Wine, la metodología de caracterización de E/S no resultó aplicable. Wine implementa las interfaces de Windows utilizadas por la aplicación y traduce las operaciones de E/S a llamadas al sistema de Linux; sin embargo, el patrón de apertura y acceso a los archivos no coincidió con los criterios de captura y filtrado del script utilizado para la caracterización. Una posible causa es que parte de las lecturas y escrituras se efectuaran mediante llamadas al sistema no contempladas por el script. Por lo tanto, los resultados de la caracterización no fueron consistentes con el comportamiento esperado de la aplicación y se optó por no incluir la caracterización de E/S.

7.3.1. Resultados experimentales

Para evaluar el rendimiento de la aplicación de calibración del modelo DayCent sobre las distintas configuraciones de Lustre se siguió la misma metodología empleada en las aplicaciones anteriores. Cada configuración se ejecutó tres veces y se reportan la media y la desviación estándar del tiempo de ejecución. Salvo que se indique lo contrario, el tamaño de stripe se fijó en 4 MiB y se utilizó el tamaño de RPC por defecto (4 MiB). La Tabla 7.15 presenta los tiempos de ejecución sobre cada una de las configuraciones estudiadas.

sistema de archivos de base	*configuración*	*tiempo de ejecución (s)*	*σ (s)*
	Ejecuciones sin sistema de archivos paralelo		
NFS	4 discos, ext4	2030	237
	4 discos, ZFS+LZ4	996	2
	6 discos, ext4	1389	100
	6 discos, ZFS+LZ4	1377	1
	Ejecuciones con sistema de archivos paralelo		
ldiskfs	1 OST	1112	2
	2 OST	1169	9
	3 OST	1122	2
	4 OST	1130	2
ZFS	4 OST	1196	2
ZFS+LZ4	4 OST	1194	1
	Ajuste de parámetros de Lustre		
ldiskfs	4 OST, striping en 1 OST	1104	4
	4 OST, striping en 2 OST	1108	2
	4 OST, striping en 3 OST	1121	6
ldiskfs	4 OST, PFL	1126	1
ZFS	4 OST, PFL	1197	2
ZFS+LZ4	4 OST, PFL	1198	1
ldiskfs	4 OST, large I/O	1156	2
	4 OST, DoM+PFL	1199	8

Tabla 7.15: Tiempo medio de ejecución y desviación estándar de la aplicación de calibración del modelo DayCent sobre las configuraciones del sistema de archivos evaluadas.

El número de OST no tuvo un efecto apreciable sobre el tiempo de ejecución. El aumento del número de OST no redujo el tiempo de ejecución e incluso la configuración de dos OST resultó marginalmente más lenta que la

de un OST. El cambio de sistema de archivos de base tuvo un efecto reducido. Con cuatro OST, los tiempos de ZFS y ZFS+LZ4 fueron levemente superiores al de ldiskfs, con diferencias inferiores a 6 %. Ninguna de las configuraciones de ajuste de parámetros de Lustre modificó de forma significativa el tiempo de ejecución. La desviación estándar fue muy baja en todas las configuraciones paralelas. El comportamiento fue consistente con una aplicación cuyo tiempo estuvo dominado por el cómputo y no por el almacenamiento. Agregar recursos de E/S no aportó una mejora en el rendimiento.

Las ejecuciones sobre NFS presentaron una desviación estándar mayor que las configuraciones paralelas. La ejecución con NFS sobre ext4 en benny registró el mayor tiempo de todas las ejecuciones y también la mayor variabilidad. La ejecución con NFS sobre ZFS+LZ4 en benny obtuvo el menor tiempo de todas las configuraciones evaluadas. Las ejecuciones sobre NFS en annifrid fueron más lentas que las ejecuciones sobre el sistema de archivos paralelo.

7.3.2. Análisis de tiempos ponderados, aceleración y eficiencia

La ponderación por heterogeneidad de OST descrita en la Sección 5.3.4 se aplicó a los tiempos de la Tabla 7.15. Se midió la relación entre lecturas y escrituras físicas de los experimentos y resultó 0,000867 lecturas por cada escritura, un volumen de lecturas físicas despreciable frente al de escrituras. En consecuencia, la carga se modeló como dominada por escrituras, con $n_{\mathrm{w}} = 100$ escrituras y un número de lecturas despreciable ($n_{\mathrm{r}} \approx 0$), por lo que los costos de referencia y agregados se calcularon sobre el trabajo de escritura. Los costos de referencia resultaron $C_{\mathrm{benny}} = 43{,}7$ y $C_{\mathrm{annifrid}} = 100{,}0$. El costo agregado $C_{\mathcal{O}}$ de cada configuración se calculó sobre el conjunto de OST que efectivamente recibió la carga: annifrid en la configuración de un OST; annifrid y agnetha en la de dos OST; annifrid, agnetha y benny en la de tres OST y los cuatro OST en las configuraciones restantes. La Tabla 7.16 presenta los tiempos ponderados obtenidos bajo la referencia de benny y la referencia de annifrid.

Bajo la referencia de benny, un escritor rápido, las configuraciones paralelas redujeron su tiempo respecto del tiempo no ponderado, porque parte de su carga de escritura recayó sobre annifrid, el OST de menor velocidad de escritura. Bajo la referencia de annifrid, el OST más lento en escrituras, los tiempos ponderados aumentaron.

sistema de archivos de base	*configuración*	$T_p^\star$ *(benny) (s)*	$T_p^\star$ *(annifrid) (s)*
ldiskfs	1 OST	486	1112
	2 OST	783	1793
	3 OST	828	1896
	4 OST	794	1818
ZFS	4 OST	840	1924
ZFS+LZ4	4 OST	839	1920
ldiskfs	4 OST, striping en 1 OST	775	1776
	4 OST, striping en 2 OST	778	1782
	4 OST, striping en 3 OST	787	1803
ldiskfs	4 OST, PFL	791	1811
ZFS	4 OST, PFL	841	1925
ZFS+LZ4	4 OST, PFL	841	1927
ldiskfs	4 OST, large I/O	812	1859
	4 OST, DoM+PFL	842	1929

Tabla 7.16: Tiempos ponderados por heterogeneidad de OST de la aplicación de calibración del modelo DayCent.

La comparación entre el sistema de archivos paralelo y NFS se realizó sobre los tiempos ponderados, mediante la comparación de la columna de referencia benny con las ejecuciones con NFS de cuatro discos y la columna de referencia annifrid con las de seis discos. Bajo la referencia benny, todas las configuraciones paralelas resultaron más rápidas que ambas ejecuciones con NFS de cuatro discos, incluida la más veloz de todas (996 s sobre ZFS+LZ4). Bajo la referencia annifrid, únicamente la configuración de un OST resultó más veloz que las ejecuciones con NFS de seis discos (1389 s sobre ext4 y 1377 s sobre ZFS+LZ4). La comparación resultó fuertemente dependiente de la referencia. Dado que la aplicación es intensiva en cómputo y no en almacenamiento, ninguna configuración produjo una mejora significativa de rendimiento sobre el acceso vía NFS más allá de superar ampliamente al peor escenario (NFS sobre ext4 de cuatro discos).

Para cuantificar el efecto del paralelismo de almacenamiento se calcularon la aceleración y la eficiencia al escalar el número de OST. El costo de referencia empleado fue el del único OST utilizado en las ejecuciones de un OST (annifrid), $C_{\text{ref}} = C_{1\,\text{OST}} = 100{,}0$. Se analizan las configuraciones de uno, dos, tres y cuatro OST sobre ldiskfs y la configuración de cuatro OST con ZFS.

La Tabla 7.17 presenta la aceleración y la eficiencia ponderadas respecto a la configuración de un OST.

sistema de archivos de base	*configuración*	*aceleración*	*eficiencia*
ldiskfs	1 OST	1,00	1,00
	2 OST	0,62	0,31
	3 OST	0,59	0,20
	4 OST	0,61	0,15
ZFS	4 OST	0,58	0,14

Tabla 7.17: Aceleración y eficiencia ponderadas de la aplicación de calibración del modelo DayCent respecto a la configuración de un OST.

La aceleración resultó inferior a uno en todas las configuraciones paralelas y se mantuvo en torno a 0,6, sin una tendencia creciente, por lo que escalar el número de OST no aceleró la ejecución respecto de la configuración de un OST. La eficiencia decreció al aumentar la cantidad de OST y la configuración de cuatro OST con ZFS sin compresión alcanzó valores similares a los de ldiskfs. Al estar una aplicación intensiva en cómputo con un volumen de E/S reducido, el paralelismo de almacenamiento no dio una mejora de rendimiento.

La calibración del modelo DayCent se comportó como una aplicación intensiva en cómputo en la que el sistema de archivos de base, el número de OST y el ajuste de parámetros de Lustre tuvieron un efecto reducido sobre el tiempo de ejecución. El sistema de archivos paralelo obtuvo un rendimiento comparable al del acceso vía NFS y no hubo una aceleración al aumentar el número de OST, a diferencia de los resultados obtenidos para las aplicaciones intensivas en E/S de las secciones anteriores.

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones del proyecto y se identifican posibles líneas de trabajo futuro.

8.1. Conclusiones

Se abordó el problema del almacenamiento como factor limitante del rendimiento en entornos de computación de alto desempeño. Se desplegó y evaluó experimentalmente el sistema de archivos paralelo Lustre sobre una infraestructura de hardware heterogéneo, representativa de la coexistencia de distintas generaciones de equipamiento en los centros de supercómputo. El despliegue se completó satisfactoriamente, se realizaron experimentos sobre los dos sistemas de archivos de base soportados por Lustre (ldiskfs y ZFS) y se comparó el comportamiento de ambos sistemas de archivos de base bajo idénticas condiciones de hardware y de carga.

La evaluación con aplicaciones reales mostró que el paralelismo introducido por el sistema de archivos redujo los tiempos de ejecución al escalar el número de OST, aunque la magnitud de la mejora del rendimiento dependió del perfil de cada aplicación. En la aplicación de procesamiento de datos de transporte público, dominada por E/S, la mejor configuración (ZFS con compresión, cuatro OST) alcanzó una aceleración de 2,41× y una eficiencia de 60 % respecto a la configuración de un OST y superó a la ejecución más lenta sobre NFS, aunque no alcanzó el rendimiento de las otras tres ejecuciones sobre NFS evaluadas. En la aplicación de procesamiento para DeepONet, la mejor configuración (ZFS sin compresión, cuatro OST) alcanzó una aceleración de

2,59× y una eficiencia de 65 % respecto a la configuración de un OST. Las configuraciones de cuatro OST superaron a la referencia NFS sobre ext4. Solo se superó a la mejor referencia NFS en las configuraciones con ZFS y con los ajustes large I/O y DoM+PFL sobre ldiskfs. En la aplicación de calibración del modelo DayCent, el tiempo de ejecución estuvo dominado por el cómputo y no por el almacenamiento. El sistema de archivos paralelo obtuvo un rendimiento comparable al del acceso vía NFS y no hubo una aceleración al aumentar el número de OST.

El número de OST fue el parámetro más influyente en el rendimiento de las aplicaciones dominadas por E/S. La mayor parte de la ganancia se concentró al escalar de uno a tres OST y la incorporación del cuarto OST aportó mejoras marginales e incluso, en algunos casos, una leve degradación del rendimiento. Se concluyó que cuando los datos se repartieron equitativamente entre OST de rendimiento distinto, el OST más lento se convirtió en un cuello de botella y limitó el rendimiento del sistema de archivos.

De los sistemas de archivos de base, ZFS (particularmente con compresión LZ4) superó el rendimiento de ldiskfs en las aplicaciones y ofreció tiempos de ejecución más estables. El beneficio de la compresión, sin embargo, dependió de la compresibilidad de los datos: fue significativo en los registros del transporte público y prácticamente nulo en los datos de DeepONet. En las operaciones de metadatos, ldiskfs mostró un rendimiento mayor al de ambas configuraciones de ZFS, especialmente en las operaciones de creación. En el escenario de experimento de estrés con un único cliente, la configuración ZFS con compresión no logró sostener el ritmo de generación de metadatos, provocó el agotamiento de la memoria RAM del cliente y la falla del experimento en dos de tres repeticiones.

La caracterización con bpftrace permitió identificar una anomalía de rendimiento en la escritura de ZFS en los experimentos de fio de la familia client, asociada a la emisión de escrituras pequeñas y dispersas. El análisis del parámetro max_segments de las controladoras y de la fragmentación de las solicitudes de E/S aportó evidencia sobre las causas de la degradación.

El trabajo confirmó la viabilidad de desplegar un sistema de archivos paralelo para computación de alto desempeño sobre hardware heterogéneo y de relativo bajo costo y mostró el beneficio de aplicar opciones de configuración según el tipo de carga de las aplicaciones a ejecutar, así como los límites del beneficio introducido por el paralelismo en cargas dominadas por cómputo.

8.2. Trabajo futuro

A partir de los resultados obtenidos en este proyecto se identificaron diversas líneas de investigación que permitirían extender y profundizar el estudio realizado. En primer lugar, los resultados evidenciaron que la distribución equitativa de stripes entre OST de rendimiento heterogéneo constituyó un cuello de botella, ya que el OST de menor rendimiento limitó la velocidad del conjunto. Por lo tanto, se considera relevante evaluar políticas de distribución no equitativa, en las que la proporción de datos asignada a cada OST sea proporcional a su rendimiento medido, para así equilibrar la carga y mitigar el impacto del nodo más lento. Otra línea de interés corresponde a la evaluación de la tolerancia a fallos y la alta disponibilidad del sistema. El despliegue efectuado no contempló la replicación ni las configuraciones de recuperación ante fallos de los servicios MDS y OSS, por lo que se propone analizar el comportamiento del sistema frente a fallos de discos y de nodos y evaluar el costo en rendimiento asociado.

Resulta relevante comparar el desempeño de Lustre con el de otros sistemas de archivos paralelos. Sobre la misma infraestructura podrían desplegarse y evaluarse pNFS, CephFS o IBM Storage Scale y comparar el rendimiento de los sistemas de archivos bajo cargas de trabajo idénticas. Dado que las pruebas se llevaron a cabo con un máximo de tres clientes sobre una red de 10 Gb/s, se propone estudiar la escalabilidad del sistema ante un mayor número de clientes y sobre interconexiones de mayor ancho de banda y menor latencia, por ejemplo InfiniBand. Finalmente, dado que la aplicación de calibración del modelo DayCent no se benefició del paralelismo de almacenamiento por estar dominada por cómputo, se propone ampliar el conjunto de aplicaciones evaluadas con cargas de perfil mixto, para caracterizar con mayor precisión las condiciones bajo las cuales el paralelismo de almacenamiento deja de aportar mejoras de rendimiento.

Bibliografía

Arpaci-Dusseau, R. y Arpaci-Dusseau, A. (2018). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, LLC, edición 1.00.

Axboe, J. (2025). fio - flexible I/O tester. `https://fio.readthedocs.io`. Versión 3.41-49-gde3d. Consultado el 03/06/2026.

bpftrace developers (2026). bpftrace: High-level tracing language for Linux. `https://github.com/bpftrace/bpftrace`. Consultado el 11/06/2026.

Ceph authors and contributors (2026). Ceph file system documentation. `https://docs.ceph.com/en/latest/cephfs`. Consultado el 20/06/2026.

Chowdhury, F., Zhu, Y., Heer, T., Paredes, S., Moody, A., Goldstone, R., Mohror, K., y Yu, W. (2019). I/O characterization and performance evaluation of BeeGFS for deep learning. En *Proceedings of the 48th International Conference on Parallel Processing*. Association for Computing Machinery.

Dell Inc. (2012). *Dell PowerEdge RAID Controller H310*. `https://i.dell.com/sites/doccontent/shared-content/data-sheets/Documents/dell-perc-h310-spec-sheet.pdf`. Consultado el 16/06/2026.

Dell Inc. (2013). *Dell PowerEdge R720 and R720xd Technical Guide*. `https://dl.dell.com/manuals/all-products/esuprt_ser_stor_net/esuprt_poweredge/poweredge-r720_reference-guide_en-us.pdf`. Consultado el 16/06/2026.

Haynes, T. (2016). Network File System (NFS) Version 4 Minor Version 2 Protocol. Request for Comments 7862, Internet Engineering Task Force.

Haynes, T. y Noveck, D. (2015). Network File System (NFS) Version 4 Protocol. Request for Comments 7530, Internet Engineering Task Force.

Hewlett Packard (2013a). *HP ProLiant DL385 G7 Server QuickSpecs*. https://www.hpe.com/psnow/doc/c04284499. Consultado el 16/06/2026.

Hewlett Packard (2013b). *QuickSpecs: HP Smart Array P410 Controller*. https://www.hpe.com/psnow/doc/c04111713. Consultado el 16/06/2026.

Hewlett Packard Enterprise (2016). *QuickSpecs: HPE Smart Array P420/P420i Controller*. https://www.hpe.com/psnow/doc/c04111534. Consultado el 16/06/2026.

Hewlett Packard Enterprise (2017). *HPE ProLiant DL385p Gen8 Server QuickSpecs*. https://www.hpe.com/psnow/doc/c04222529. Consultado el 16/06/2026.

IBM (2025). *IBM Storage Scale 5.2.3: Concepts, Planning, and Installation Guide*. https://www.ibm.com/docs/en/storage-scale. Consultado el 20/06/2026.

IOR developers (2026). IOR: Interleaved or Random parallel I/O benchmark. https://ior.readthedocs.io/en/latest. Consultado el 11/06/2026.

Lustre developers (2017). Lustre* software release 2.x: Operations manual. https://doc.lustre.org/lustre_manual.pdf. Consultado el 21/06/2026.

MDTest project (2010). *mdtest(1): test file system metadata performance*. Man page distribuida con mdtest 1.8.3.

Nesmachnow, S. y Iturriaga, S. (2019). Cluster-UY: Collaborative scientific high performance computing in Uruguay. En Torres, M. y Klapp, J. (eds.), *Supercomputing*, págs. 188–202, Cham. Springer International Publishing.

Noveck, D. y Lever, C. (2020). Network File System (NFS) Version 4 Minor Version 1 Protocol. Request for Comments 8881, Internet Engineering Task Force.

OpenSFS (2026). Lustre File System. https://www.opensfs.org/lustre. Consultado el 16/06/2026.

OpenZFS (2020). OpenZFS documentation. https://openzfs.github.io/openzfs-docs. Consultado el 17/06/2026.

Pan, X., Luo, Z., y Zhou, L. (2023). Navigating the landscape of distributed file systems: Architectures, implementations, and considerations. *Innovations in Applied Engineering and Technology*, págs. 1–12.

Saini, S., Rappleye, J., Chang, J., Barker, D., Mehrotra, P., y Biswas, R. (2012). I/O performance characterization of Lustre and NASA applications on Pleiades. En *19th International Conference on High Performance Computing*, págs. 1–10. IEEE.

Seagate Technology LLC (2017). *IronWolf 3.5 HDD Data Sheet: The Power of Agility for Home, SOHO and SMB NAS Enclosures*. Seagate Technology LLC. Modelo ST10000VN0004; documento DS1904.9-1707US.

Silberschatz, A., Galvin, P., y Gagne, G. (2018). *Operating System Concepts*. Wiley, Hoboken, NJ, edición 10.

Sterling, T., Brodowicz, M., y Anderson, M. (2017). *High Performance Computing: Modern Systems and Practices*. Morgan Kaufmann, edición 1.

The Linux Kernel Developers (2026). ext4 data structures and algorithms. `https://www.kernel.org/doc/html/v7.1/filesystems/ext4`. Consultado el 17/06/2026.

The Lustre Developers (2025). Understanding Lustre internals. `https://wiki.lustre.org/Understanding_Lustre_Internals`. Consultado el 17/06/2026.

Wang, F., Nelson, M., Oral, S., Atchley, S., Weil, S., Settlemyer, B., Caldwell, B., y Hill, J. (2013). Performance and scalability evaluation of the Ceph parallel file system. En *Proceedings of the 8th Parallel Data Storage Workshop*, págs. 14–19, New York, NY, USA. Association for Computing Machinery.

Wang, Y., Lu, Y., Qiu, C., Gao, P., y Wang, J. (2011). Performance evaluation of a Infiniband-based Lustre parallel file system. *Procedia Environmental Sciences*, 11:316–321. 2nd International Conference on Challenges in Environmental Science and Computer Engineering.

Weil, S., Brandt, S., Miller, E., Long, D., y Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. En *7th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association.

Western Digital Corporation (2018a). *WD Blue PC Hard Drives Data Sheet*. Western Digital Corporation. Modelo WD60EZRZ; documento 2879-771436-A11.

Western Digital Corporation (2018b). *WD Red NAS Hard Drives Data Sheet*. Western Digital Corporation. Modelo WD60EFRX; documento 2879-800002-A11.

Western Digital Corporation (2025). *WD Red Plus 3.5-inch NAS HDD Data Sheet*. Western Digital Corporation. Modelo WD60EFPX.

Zhao, T. y Hu, J. (2010). Performance evaluation of parallel file system based on Lustre and grey theory. En *9th International Conference on Grid and Cloud Computing*, págs. 118–123. IEEE.