Universidad Carlos III de Madrid

Ingeniería Técnica en Informática de Gestión

PROYECTO FIN DE CARRERA:

ANÁLISIS, DISEÑO E IMPLANTACIÓN DE UNA SOLUCIÓN DE ALMACENAMIENTO DISTRIBUIDO PARA UN GRUPO DE INVESTIGACIÓN

Autor: David Ventas Sierra

Tutor: Alejandro Calderón Mateos

Para mi madre.

We make our own choices,

but in the end, our choices make us.

ANDREW RYAN

Agradecimientos:

Llegado a este punto, me gustaría agradecer a todas aquellas personas que de un modo u otro han intervenido y me han ayudado directa o indirectamente a terminar el proyecto fin de carrera.

Les doy las gracias a todos los compañeros que me han acompañado durante la carrera, en especial a todos mis compañeros de prácticas recurrentes. También a mis familiares y amigos, por su apoyo y por su ayuda en los momentos más críticos de estos años.

Por último, quiero agradecer especialmente a mi tutor, Alejandro Calderón, que aceptase hacer este proyecto conmigo: gracias por tu ayuda y por la paciencia que has tenido conmigo.

Gracias a todos. Sin vosotros no habría podido llegar hasta aquí.

RESUMEN

El objetivo de este proyecto es estudiar las distintas posibilidades de almacenamiento distribuido que se ofrecen hoy en día y encontrar la más adecuada para el buen funcionamiento de un grupo de investigación universitario.

Para esto, se recopilarán las necesidades y requisitos del grupo y posteriormente se creará un entorno virtual de pruebas y se instalarán y probarán exhaustivamente en él una serie de sistemas de almacenamiento candidatos, empleando para ello una colección de herramientas de medición especializadas escogidas en función tanto de los parámetros que miden como de la manera en que lo hacen.

Una vez extraídos los datos de rendimiento se tratarán y mostrarán de una forma accesible para que se puedan comparar fácilmente y seleccionar, dentro de las tecnologías que cumplan los requisitos, la que ofrezca unas mejores prestaciones.

TABLA DE CONTENIDO

íNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

NOTA DEL AUTOR

La memoria de este proyecto es un texto técnico, del ámbito de la informática, y como tal presenta ciertas peculiaridades. Se manejan constantemente conceptos nuevos que no siempre tienen una buena traducción al español o no tienen traducción posible. Para mantener el texto lo más correcto posible se han seguido los criterios de los siguientes libros y entidades:
-*Ortografía de la lengua española.* RAE, Madrid, Espasa, 2010. 978-84-3426-4670
-*Manual de estilo de la lengua española (MELE4)*. José Martínez de Sousa. Editorial Trea. ISBN 9 788497 046060.
-Fundéu BBVA: http://www.fundeu.es/recomendacion/web-plural-webs-604/

Sobre las citas y referencias bibliográficas, se van a tratar con un número arábigo a modo de superíndice, y el formato de la cita va a ser el estilo Harvard, según el documento «Cómo citar bibliografía: UNE-ISO 690» disponible en la biblioteca de la UC3M (http://portal.uc3m.es/portal/page/portal/biblioteca/aprende_usar/como_citar_bibliografia). Estas recomendaciones también se pueden encontrar en el mismo libro citado anteriormente, en la primera parte, capítulo 4 «Notas», apartado 4.3 «La llamada de nota», página 80.

En este estudio se han extraído numerosos diagramas de la documentación oficial de las tecnologías de almacenamiento. Al ser simplemente imágenes en una web, las citas no incluyen una persona como autor, sino la propia empresa.

Me he permitido incluir neologismos que aunque no están aún aceptados, no hay traducciones apropiadas y dentro del ámbito de la informática son comprensibles por todos. Algunas de estas palabras son: *escalabilidad*, *incremental*, *funcionalidad*, *soportar* o *montar*.

1. Introducción

El volumen de datos que se maneja hoy en día, ya sea por usuarios particulares, empresas multinacionales, o grupos de investigación, va en constante aumento. Tanto la necesidad de espacio de almacenamiento como la de disponibilidad de los datos en cualquier momento y lugar son cada vez mayores. Para poder dar respuesta a estas necesidades se emplean sistemas de almacenamiento distribuidos.

Este tipo de sistemas, además de ser transparentes para el usuario final, cuenta con una serie de ventajas sobre los sistemas centralizados, como por ejemplo una mayor disponibilidad de los datos por el hecho de encontrarse separados, una mayor flexibilidad que permite adaptar el sistema exactamente a las necesidades del grupo de usuarios y un coste notablemente menor que el de una estructura centralizada.

Dentro de los sistemas de almacenamiento distribuidos encontramos dos tipos de solución: los sistemas de ficheros distribuidos y las bases de datos distribuidas.

1.1. Motivación

Las necesidades de almacenamiento de un grupo de investigación en el entorno universitario no son inmensas en cuanto a cantidad, pero sí son variadas. Se deben tener en cuenta tanto las necesidades directas de los usuarios como las necesidades indirectas a través de las aplicaciones y servicios que utilizan.

Además de la gestión directa del *cluster* de almacenamiento, también existe la opción de la externalización del almacenamiento a través de algún servicio de computación en la nube. Actualmente, numerosas empresas ofrecen servicios de virtualización, como Google Cloud, Microsoft Azure o Amazon EC2.

1.2. Objetivos

El objetivo de este estudio es determinar cuál es la mejor solución de almacenamiento distribuido para un grupo de investigación universitario, desde el punto de vista del rendimiento y del coste.

Para ello, se analizarán las necesidades de un grupo de investigación, se seleccionarán y estudiarán varios de los productos existentes más prometedores, y posteriormente se diseñarán soluciones basadas en los que a priori parezcan más adecuados.
Una vez diseñadas estas soluciones de almacenamiento distribuido, se implantarán en una plataforma de pruebas virtualizada. El principal beneficio del uso de una plataforma virtual, tanto ajena como propia, es el poder independizar la solución de almacenamiento del *hardware* real que la alberga y el poder uniformar el sistema operativo sobre el que se ejecuta de una manera sencilla.

Una vez desplegados, se extraerán datos de rendimiento a partir de pruebas que simulen situaciones reales de carga de trabajo. Estas mediciones se realizarán con herramientas de *software* específicas para dicha tarea (*benchmarks*).

Además, el hecho de utilizar una plataforma virtual sobre una máquina real potente proporciona dos beneficios adicionales:

- El rendimiento que se obtenga de un servicio externalizado como los mencionados anteriormente nunca será inferior al obtenido en estas mediciones, ya que se está estudiando el peor escenario posible, esto es, que todas las máquinas virtuales del *cluster* estén ejecutándose en la misma máquina real.
- Es una buena simulación del rendimiento que se obtendría al desplegar el entorno virtualizado en las máquinas de las que dispone un grupo de investigación, por ejemplo en aulas docentes, ya que es menos potente que el *hardware* real empleado en este estudio.

1.3. Estructura del documento

Para obtener una visión de conjunto del trabajo que se pretende realizar, a continuación se explica brevemente en qué consiste cada capítulo de la presente memoria:

- Estado de la cuestión: se explica más detalladamente cuáles son los tipos de tecnologías existentes y se presentan algunas de las soluciones que ofrece el mercado para dar respuesta a las necesidades de almacenamiento. Se explica además en qué consiste el almacenamiento distribuido y las principales diferencias entre sistemas de ficheros y bases de datos. Se presentan también los *benchmarks* y la información útil que se obtiene de ellos.

- Análisis y propuestas de diseño de una solución de almacenamiento para un grupo de investigación: se analizan formalmente las necesidades del grupo, con casos de uso y un pliego de requisitos. Se analizan y comparan las tecnologías candidatas y se diseñan las soluciones para dar respuesta a los requisitos.

- Evaluación de las soluciones propuestas: se describe el entorno y se explica el despliegue del mismo, en el que se probarán las soluciones candidatas. Se presentan las características que van a ser evaluadas, se recogen los datos de los *benchmarks*, se comparan e interpretan.

- Planificación y presupuesto: se describe un plan para implementar la mejor solución y los costes asociados a dicho despliegue.

- Conclusiones y trabajos futuros: se recogen las conclusiones de todo el trabajo, tanto generales como personales. Además, se añaden nuevas líneas de trabajo para continuar con lo iniciado en este proyecto.

2. Estado de la cuestión

2.1. Tipos de soluciones de almacenamiento distribuido

Para dar respuesta a las necesidades de grandes capacidades de almacenamiento, tolerancia a fallos y rendimiento de almacenes de datos, se utilizan sistemas de almacenamiento distribuido. Un sistema distribuido es aquel que emplea varios nodos o servidores para llevar a cabo su tarea.

Dentro de los sistemas de almacenamiento distribuidos, existen dos tipos fundamentales: los sistemas de ficheros distribuidos y las bases de datos distribuidas. A su vez, dentro de las bases de datos, existen otros dos tipos: las bases de datos SQL y las NoSQL.

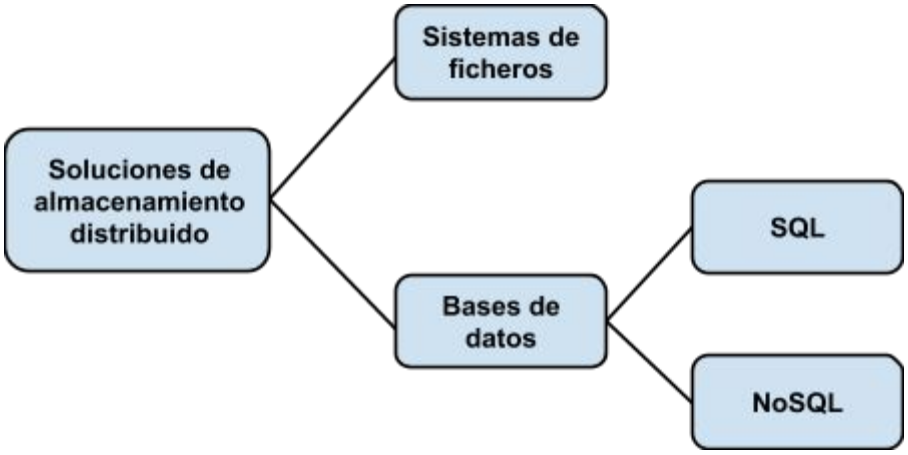


Figura 2.1: Esquema de los tipos de sistemas de almacenamiento distribuidos.

Conviene en este punto hacer una distinción entre bases de datos operacionales y analíticas. Se trata de una distinción en el uso que se les va a dar. Las operacionales van a ser grandes bases de datos que van a soportar numerosísimas lecturas y escrituras constantemente, mientras que las analíticas son bases de datos aún más grandes y sus datos sobre todo van a ser consultados y procesados, no tanto modificados.

2.1.1. Sistemas de ficheros

Un sistema de ficheros debe ser transparente para el usuario. Lo ideal es que tanto el usuario como las aplicaciones lo vean como un sistema de ficheros normal.

Es necesario distinguir entre:

- Servicio de archivos es la especificación de los servicios que ofrece el sistema de ficheros a sus clientes (S.O., etcétera).
- Servidor de archivos es un proceso que se ejecuta en una máquina y ayuda a implementar el servicio de archivos distribuidos. Deben ser transparentes para el usuario.

Un servicio de archivos consta del «verdadero» servicio de archivos y el servicio de directorios.

El servicio de archivos:

- Proporciona primitivas para leer y escribir los atributos apropiados.
- Algunos servicios pueden implementar solo operaciones Create y Read, creando ficheros inmutables.
- Para la protección de los ficheros, el servicio de archivos se puede implementar mediante *capabilities*, que son una especie de *tickets* que guardan los usuarios con los permisos adecuados para cada fichero al que tienen acceso, o mediante tablas centralizadas (*access control list*) asociadas a cada fichero, como se hace en UNIX.
- El servicio puede seguir o bien el modelo carga/descarga o el de acceso remoto. En el primer caso cada vez que se accede a un fichero se envía completo y se recibe igual. En el segundo caso se puede a acceder a partes concretas del fichero. Se pierde rapidez y sencillez en la implementación pero se gana eficiencia para tamaños medios.

El servicio de directorios:

- Proporciona primitivas para crear y eliminar directorios, mover, nombrar y renombrar ficheros.

Existen tres métodos usuales para nombrar los archivos y directorios en un sistema distribuido:
1. Nombre máquina + ruta de acceso, como /máquina/ruta.
2. Montaje de sistemas de archivos remotos en la jerarquía local de archivos.
3. Un espacio de nombres que tenga la misma apariencia en todas las máquinas (independencia con respecto a la posición).

Los ficheros y directorios tienen un nombre binario y otro «legible» para los humanos. Existe una tabla en la que se relacionan ambos. En los servidores autocontenidos (aquellos que no tienen referencias a archivos o directorios situados en otros servidores) se puede asignar una numeración propia, como por ejemplo el número de i-nodo. Se puede solucionar haciendo que el nombre binario indique el servidor y un archivo específico en ese servidor. Este método permite que un directorio en un servidor contenga un archivo en un servidor distinto. Otra alternativa, que a veces es preferible, es utilizar un enlace simbólico, que es una entrada de directorio asociada a una cadena (servidor, nombre de archivo), la cual se puede buscar en el servidor correspondiente para encontrar el nombre binario.

Frente a los sistemas de ficheros locales, los sistemas de ficheros distribuidos presentan las siguientes ventajas:

- El beneficio más evidente es la accesibilidad. Varios clientes pueden acceder a los datos a través de una red.
- La capacidad de almacenamiento es potencialmente mucho mayor que la que pueda presentar un ordenador de sobremesa.
- Puede ser tolerante a fallos, ya que los datos se encuentran distribuidos. Además, puede contar con sistemas de redundancia que protejan mejor los datos que almacenan.

Aunque no están exentos de ciertos inconvenientes:

- El primer inconveniente es que se añade otro posible foco de problemas que es la red que media entre el cliente y los ficheros: errores de configuración, cortafuegos, desconexiones, latencia, etcétera.
- Al añadir redundancia se añade también complejidad en el sistema. Mantener todas las copias de los ficheros sincronizadas no es una tarea sencilla.
- La disponibilidad mencionada anteriormente provoca problemas de concurrencia a los ficheros. Es necesario arbitrar los accesos cuando varios clientes quieren acceder a un mismo fichero simultáneamente.

Algunos de los sistemas de ficheros distribuidos más populares son NFS, Lustre, LizardFS, GlusterFS, la interfaz POSIX de Ceph, AFS, CIFS o DCE.

2.1.2. Bases de datos SQL

El término «base de datos» se escuchó por primera vez en 1963 en California (EE.UU.), con el significado de «un conjunto de información relacionada, toda ella estructurada y agrupada» [1] (Jiménez, 2014). La constante evolución de las bases de datos hace que sea difícil mantener una definición formal.

Una base de datos es una colección o depósito de datos, donde estos se encuentran lógicamente relacionados entre sí. Las bases de datos relacionales son aquellas que representan los datos y las relaciones entre los datos mediante una colección de tablas, cada una con un nombre único, donde cada fila de una tabla representa una relación entre un conjunto de valores.

Como su propio nombre indica, las bases de datos SQL utilizan el extendido Lenguaje de consulta estructurado o *Structured Query Language* en inglés.

A continuación se enumeran las ventajas que ofrece este tipo de bases de datos:

- Se basan en un lenguaje bien definido y con un largo recorrido. Estandarizado desde hace casi 30 años, hace que muchos profesionales que conocen el lenguaje puedan tener una ventaja a la hora de familiarizarse con cualquiera de las soluciones que implementa este lenguaje.
- Se trata de una tecnología madura y asentada.
- El álgebra relacional sobre la que se asienta SQL es un sólido conjunto de operaciones que sirven para diseñar bases de datos relacionales y definir consultas sobre ellas.
- SQL permite interactuar fácilmente con los datos, definiendo consultas que permiten dar un nuevo enfoque a los datos almacenados.
- Es capaz de realizar una gran cantidad de consultas complejas por segundo.
- Estas bases de datos son adecuadas tanto para entornos que requieren transacciones de escrituras rápidas como para análisis en profundidad de los datos, que requieran lecturas intensivas.
- No hay nada inherente a SQL que impida que sea escalable. SQL aporta una abstracción sobre los datos que lo hace perfectamente viable. Los sistemas modernos proporcionan herramientas para favorecer la escalabilidad horizontal, la redundancia y la tolerancia a fallos.
- SQL es compatible con JSON (*JavaScript Object Notation*), como muchos entornos NoSQL.

- Es fácil encontrar soporte en el desarrollador del gestor de la base de datos. Incluso existen muchos problemas comunes entre fabricantes, ya que todos utilizan SQL como base.
- A fecha de la redacción de este estudio sigue siendo la opción más extendida en el mercado, incluso para problemas de *Big Data.*

Sin embargo, las bases de datos SQL también cuentan con algunos inconvenientes:

- Las licencias suelen estar pensadas para grandes servidores, por lo que si se quiere escalar, resulta un precio muy elevado.
- Las necesidades de lecturas y escrituras de algunos entornos hacen que las bases de datos relacionales SQL tengan problemas de rendimiento en determinados picos. Al tener que sincronizar muchas tablas para hacer ciertas lecturas y escrituras, y todo esto prácticamente en tiempo real, pueden aparecer los problemas mencionados a la hora de dar el servicio requerido.

2.1.3. Bases de datos NoSQL

Las bases de datos NoSQL [2,3] (Fowler y Sadalage, 2013), (Wikipedia, 2015), a menudo llamadas «no solo SQL», del inglés *Not only SQL*, no poseen un lenguaje común. Este tipo de bases de datos es NoREL, que quiere decir que no sigue un modelo relacional en el que existen tablas que se relacionan con otras.
La propia definición de la sigla NoSQL está en entredicho y algunos le dan el significado de *No SQL*, por no considerarlas bases de datos. Sin embargo, mayoritariamente se las engloba dentro de las bases de datos. Evidentemente su funcionamiento está lejos de la rigidez de una base de datos relacional, pero siguen estando más próximas a estas que a los sistemas de ficheros distribuidos. Por eso en este estudio se considera a las tecnologías NoSQL bases de datos no relacionales como un subconjunto de las bases de datos.

Las bases de datos SQL se basan en el modelo relacional. Las NoSQL siguen cuatro modelos operacionales o sistemas de funcionamiento fundamentales. Estos son:

- Clave-Valor:

Se trata de la implementación más básica de NoSQL. Funciona de manera semejante a un diccionario, relacionando una clave dada con el valor que se quiere almacenar. El valor se puede recuperar suministrando la clave al sistema.
Algunos de los productos más destacados son CouchDB, Dynamo, MemcacheDB y Redis.

- Columna:

Una columna es un *array* bidimensional donde cada clave (fila) tiene al menos un par clave-valor asociados. Este modelo permite almacenar grandes cantidades de datos no estructurados.
Este tipo de base de datos se encuentra en MonetDB o RCFile.

- Documento:

Se trata de una derivación del tipo clave-valor. La mayor diferencia se encuentra en que en este tipo de bases de datos, estos no son opacos al sistema, sino que se utiliza la estructura de lo que se almacena para obtener metadatos que son útiles para diversas optimizaciones del funcionamiento.
Los ejemplos más notables son: MongoDB, CouchDB y Cassandra.

- Grafo:

Estas bases de datos se basan en la teoría de grafos. Existen tres elementos fundamentales: los nodos, las propiedades y las aristas. Los nodos representan el grueso de la información que se quiere almacenar. Las propiedades son información relevante asociada a los nodos. Las aristas son las relaciones que existen entre nodos y entre nodos y propiedades.
Debido a la naturaleza de este modelo, estas bases de datos son muy rápidas a la hora de ejecutar ciertas operaciones matemáticas, como por ejemplo hallar el camino más corto entre dos nodos.
Al no depender de un esquema rígido, son adecuadas para usos en los que la naturaleza de los datos o su formato cambia.
Las bases de datos relacionales suelen ser más rápidas y resuelven las mismas consultas que las NoSQL de tipo grafo cuando la cantidad de datos que se manejan es muy grande.
Las bases de datos de tipo grafo más populares son: OrientDB y Neo4J.

Presentan ventajas como:

- Las Bases de datos NoSQL nacen con la idea de distribuir horizontalmente el trabajo entre un grupo de servidores llamado *cluster*.
- La escalabilidad, al formar parte de la naturaleza del NoSQL, se consigue de una manera bastante más económica que en SQL. Añadir máquinas al *cluster* resulta bastante más barato que mejorar el *hardware* de un enorme servidor SQL.
- La mayoría de las licencias son de código abierto y pensadas para funcionar como *cluster.*
- Si se quiere almacenar datos de una base de datos relacional, al no tener que procesar operaciones de JOIN, el rendimiento a la hora de mostrar los datos es muy superior.
- Aunque haya duplicación en los datos, su arquitectura diseñada para ser distribuida hace que el precio del almacenamiento no sea un factor limitante.

Entre los inconvenientes con que nos encontramos:

- No existe una definición reconocida de lo que es una tecnología NoSQL. Una de las definiciones comúnmente aceptadas es: «sistema que proporciona operaciones sencillas como el almacenamiento de clave-valor y que se concentra en la escalabilidad horizontal para llevarlas a cabo» [4] (Fowler, 2012) .
- Solo se obtienen beneficios suficientes de su utilización cuando se emplean para resolver un determinado conjunto de problemas muy concreto.
- Si se quiere almacenar datos de una base de datos relacional, lo normal es que aparezcan datos duplicados, ya que NoSQL no mantiene tablas interrelacionadas.

- Ausencia de ACID (*Atomicity, Consistency, Isolation and Durability*). Son las características que permiten clasificar las operaciones de un SGBD.
- Es difícil encontrar grandes proveedores de servicios que ofrezcan soluciones NoSQL. Probablemente esto sea debido a la ausencia de personal cualificado para poder diseñar y dar soporte a estas tecnologías.

2.1.4. Tabla comparativa de características

Las principales diferencias entre las bases de datos SQL y las NoSQL se recogen en la siguiente tabla:

	Sistemas de ficheros	**Bases de datos SQL**	**Bases de datos NoSQL**
Almacenamiento	Almacenan los datos en ficheros a los que asocian unos metadatos con información adicional.	Se organiza en tablas. Las tablas contienen filas y columnas. Cada fila contiene la información de una entidad y cada columna la de un atributo.	Generalmente hay 4 tipos básicos. Pueden estar orientadas a documentos, columnas, clave-valor y grafo.
Estructura	Cada sistema de ficheros organiza los ficheros de una manera propia, como los i-nodos en Unix.	Ha de ser fija y estática. Generalmente, cada fila tiene que tener un valor para cada columna.	El equivalente a cada fila no tiene necesariamente un valor para cada columna.
Flexibilidad	No es posible variar la estructura de un sistema de ficheros.	Cambiar el número de columnas o añadir nuevas tablas es un proceso que requiere la detención de la base de datos.	Se pueden realizar cambios al vuelo o en caliente.

Escalabilidad	Los sistemas de ficheros distribuidos modernos pueden aumentar y disminuir su capacidad de almacenamiento en caliente.	La escalabilidad de las bases de datos clásicas era vertical. Solía conseguirse con un servidor más potente. Pueden distribuirse pero es un proceso más lento y complicado que en NoSQL.	Su escalabilidad es horizontal, y eso le permite añadir nuevos servidores que no han de ser necesariamente grandes máquinas potentes y costosas. Incluso el proceso de distribución suele estar automatizado.
ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)	ACID es una característica propia de las bases de datos.	Sí lo cumple.	No lo cumple.
Estandarizado	Existe la interfaz estándar POSIX (IEEE 1003, ISO/IEC 9945) que cumplen la inmensa mayoría de los sistemas de ficheros actuales.	SQL es un lenguaje estandarizado, incluso normalizado en ISO/IEC 9075. Su primera estandarización data de 1986.	No hay un estándar.
Principales usos	Son omnipresentes. Desde un *smartphone*, tableta, PC o *cluster* para grandes cantidades de datos.	Indicado si hay necesidad de hacer constantemente consultas complejas.	Indicado para el manejo de grandes cantidades de datos (*Big Data*).
Soporte	Existen grandes comunidades dedicadas a desarrollar y dar soporte a estos productos.	Los fabricantes de los productos SQL suelen dar soporte.	En muchas tecnologías que implementan NoSQL es necesario recurrir a la comunidad de usuarios.
Licencias	Las soluciones basadas en sistemas	Orientadas a un único servidor	Código abierto, orientado a un

	de ficheros distribuidos suelen estar licenciadas según LGPL versión 2.1 o BSD.	centralizado.	*cluster*.
Ejemplos de tecnologías que lo implementan	Ceph, GlusterFS, Lustre y NFS.	MySql, Oracle, Sqlite, Postgres y MS-SQL.	MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j y CouchDb.

Tabla 2.1: Comparativa de las principales características de los tres tipos de sistemas de almacenamiento distribuidos.

2.2. *Benchmarks*

Para poder elegir con conocimiento de causa qué solución de almacenamiento distribuido se adapta mejor a las necesidades de un determinado problema, hay que analizar qué tipo de uso se le va a dar. Una vez que se sabe qué exigencias se tienen, se puede emular el uso real del sistema mediante unas herramientas llamadas *benchmarks* o bancos de pruebas.

2.2.1. *Benchmarks* para soluciones de almacenamiento

Un *benchmark* es un programa que permite medir el rendimiento de un determinado componente de un ordenador o de otro programa. El objetivo final es poder obtener unos valores objetivos que nos sirvan de referencia para poder comparar unos dispositivos con otros. En el caso concreto del almacenamiento, objeto de este estudio, va a obtener una serie de valores que ilustran el rendimiento de un disco duro. El *benchmark* se va a encargar de leer y escribir ficheros en un disco duro y de medir el tiempo que se tarda en realizar cada tarea para así obtener las velocidades reales del dispositivo bajo un determinado escenario de carga de trabajo.

La mayoría de estos *benchmarks* permiten configurar las características de la secuencia de trabajo que van a llevar a cabo, como por ejemplo el número de lecturas y escrituras, el tamaño de los ficheros que se van a leer y escribir, el número de hilos en ejecución, el tamaño de bloque, etc.

Un *benchmark* nos va a permitir conocer de antemano el comportamiento y el rendimiento del sistema antes de su despliegue. Nunca se va a poder predecir exactamente la carga de trabajo y la forma en la que esta se va a distribuir a lo largo del tiempo, pero es posible hacerse una idea aproximada del rendimiento del sistema si se hace una buena estimación del uso que se le va a dar.

La naturaleza del entorno virtual de pruebas va a influir en los resultados de los *benchmarks* que se van a utilizar. Sin embargo, esta limitación será la misma para todos los *benchmarks*, y será posible obtener información útil al comparar los datos.

2.2.2. *Benchmarks* aplicables y aspectos medibles

No existe un consenso sobre qué *benchmark* debe utilizarse para obtener datos de un sistema de ficheros. Un *benchmark* es una herramienta que trata de medir determinados aspectos en un sistema, y que tiene capas por debajo y por encima de él que pueden alterar las mediciones. Aunque no son herramientas altamente complejas, la tarea de la medición sí es más complicada de lo que parece.

Se han seleccionado los *benchmarks* más populares (en función del volumen de resultados en Google), más utilizados en análisis de productos en prensa (Linux Magazine, Storage Review) o los que tienen más documentación disponible. A continuación, se presentan los *benchmarks* utilizados.

2.2.2.1. IOzone

IOzone es un *benchmark* gratuito para sistemas de ficheros. Es capaz de generar y medir el rendimiento de una gran variedad de operaciones sobre ficheros y se puede ejecutar en los sistemas operativos más extendidos. Las pruebas que puede llevar a cabo y cuya velocidad puede medir son:

- Escritura (*Write*): es la creación y escritura de un fichero. Cuando se crea un fichero, también es necesario crear sus metadatos, lo que supone una tarea adicional a la escritura en sí.
- Reescritura (*Re-write*): se trata de escribir un fichero que ya existe, así que no es necesario reescribir muchos de los metadatos del fichero.
- Lectura (*Read*): simplemente consiste en solicitar los datos contenidos en un fichero existente.
- Relectura (*Re-read*): consiste en leer un fichero poco después de haberlo leído. Esta operación debe tardar menos que una lectura ordinaria gracias a la caché del sistema operativo.
- Lectura aleatoria (*Random Read*): es una lectura no secuencial. Se solicita un fragmento cualquiera del fichero. El rendimiento de un sistema sometido a esta prueba depende de muchos factores como el tamaño de la caché, las latencias de búsqueda, etcétera.
- Escritura aleatoria (*Random Write*): semejante al anterior, escrituras a partes aleatorias del fichero.

- Mezcla aleatoria (*Random Mix*): se trata de una mezcla de lecturas y escrituras a zonas aleatorias de un fichero. La distribución de las lecturas y escrituras se hace según el algoritmo Round-Robin. Para poder evaluar este parámetro es necesario poder contar con varios hilos de ejecución simultáneamente. Cada hilo se encarga de una operación de lectura o de escritura.
- Lectura hacia atrás (*Backwards Read*): mide la velocidad a la que se puede leer un fichero hacia atrás. Pocas aplicaciones lo hacen y pocos sistemas operativos disponen de mecanismos para detectar y mejorar una lectura hacia atrás.
- Lectura a zancadas (*Strided Read*): las zancadas son un tipo de lectura especial empleada por algunas aplicaciones que contienen ciertas estructuras de datos en el interior de sus ficheros. Estos se manipulan con un patrón del tipo lectura, búsqueda, lectura, búsqueda, etcétera, con tamaños fijos tanto para las lecturas como para los saltos.
- Escritura con *buffer* (*Fwrite*): obtiene el rendimiento a la hora de escribir utilizando la función *fwrite*. El fichero almacenado en *buffer* está en el espacio de direcciones del usuario, así que se reducen las llamadas al sistema operativo. Se crea un nuevo fichero para hacer la medición.
- Reescritura con *buffer* (*Frewrite*): es igual que en el caso anterior pero sin necesidad de reescribir todos los metadatos.
- Lectura con *buffer* (*Fread*): el concepto es igual que en la escritura con *buffer*, pero realizando lecturas en su lugar.
- Relectura con *buffer* (*Freread*): se trata de hacer una lectura con *buffer* a un fichero que ha sido leído recientemente.

2.2.2.2. FIO (Flexible I/O)

FIO es un benchmark gratuito y multiplataforma. El objetivo de FIO es crear una carga de trabajo flexible y configurable por el usuario y generar unos datos de salida en las unidades deseadas. El diseño de la carga de trabajo es completamente independiente del sistema operativo donde se va a ejecutar. Además, FIO permite la ejecución remota y la recopilación local de datos.

Los aspectos más interesantes de la salida de FIO son los siguientes:

- Para cada operación en cada tarea (lectura, escritura...) calcula la cantidad de *kilobytes* movidos, el ancho de banda registrado, el número de operaciones, la latencia media y el tiempo de ejecución (*io*, *bw*, *iops*, *lat avg* y *runt*, respectivamente).

- El resumen final de cada tipo de operación de todas las tareas muestra la cantidad de *kilobytes* movidos, la suma del ancho de banda de todos los hilos y el ancho de banda mínimo y máximo (*io*, *aggrb*, *minb* y *maxb*).

La latencia media de fin es el tiempo que se tarda en completar la operación desde que la orden es enviada al *kernel*.

La latencia media total es el tiempo que transcurre desde que se crean las estructuras necesarias y se completa la operación, por lo que engloba a la latencia de fin. La latencia de envío es otro valor llamado *slat*, de *submission latency*.

Todas las velocidades están medidas en *kilobytes* por segundo (KB/sec).

2.2.2.3. Filebench

Filebench es un *benchmark* para sistemas de ficheros y almacenamiento. Es muy flexible y permite generar muy fácilmente situaciones de trabajo semejantes a servidores de correo y servidores web, ya que incorpora más de cuarenta personalidades o comportamientos predefinidos. Es posible además añadir nuevas personalidades creadas por el usuario para que se adapten de la manera más fiel posible a las necesidades de las pruebas.

Los datos de salida de Filebench miden la velocidad en *megabits* por segundo (Mb/s) de las mismas operaciones que realizan otros benchmarks, y además calcula el tiempo que se tarda en llevar a cabo otras operaciones como la apertura o el cierre de los ficheros. Adicionalmente, facilita el número de operaciones de cada tipo que se han ejecutado y el número de operaciones por segundo.

Cada tipo de operación se mide de la siguiente manera:
- Nº de ops: número de operaciones.
- Ops/s: número de operaciones realizadas por segundo.
- Mb/s: tasa de transferencia medida en *megabits* por segundo.
- ms/op: cuántos milisegundos tarda cada operación de media.
- us/op-cpu: microsegundos que tarda cada operación de CPU.
- R/W: número de lecturas y escrituras, respectivamente.
- Latencia: tiempo medio de retardo.

2.2.2.4. Postmark

Postmark es un *benchmark* especializado, útil para medir el rendimiento de un sistema de almacenamiento orientado a albergar un servidor de correo electrónico o a recibir exigencias semejantes. El *benchmark* en cuestión está diseñado para comportarse como haría un servidor de correo, haciendo el mismo tipo de peticiones al sistema operativo. Una ejecución de este *benchmark* se divide en tres fases:

- Creación: se crea un grupo de ficheros y directorios cuyo número es configurable por el usuario.
- Transacciones: durante esta fase se simula el funcionamiento del servidor llevando a cabo operaciones como crear ficheros, borrarlos, leerlos y anexarles datos.
- Borrado: finalmente se borran las estructuras resultantes de las fases anteriores.

Los valores que arroja la salida de Postmark son:

- Transacciones por segundo (*transactions per second*): Número de operaciones por segundo.
- Datos leídos (*Data Read*): medido en *kilobytes* por segundo (Kbytes/s).
- Datos escritos (*Data Written*): medido en *kilobytes* por segundo (Kbytes/s).

2.2.2.5. Fdtree

Fdtree es un *benchmark* especializado en medir el rendimiento de operaciones que supongan una alta carga de trabajo sobre los metadatos del sistema de ficheros. Funciona en cuatro pasadas recursivas; la primera crea tantos niveles de directorios y número de directorios dentro de estos como se le haya indicado; la segunda crea un número de ficheros también configurable dentro de cada directorio; la tercera elimina los ficheros y la cuarta los directorios.

La salida de Fdtree ofrece los siguientes valores:

- creación de directorios por segundo (*Directory creates per second*)
- creación de ficheros por segundo (*File creates per second*)
- *kilobytes* por segundo en la operación de creación de ficheros (*KiB per second*)
- eliminación de directorios por segundo (*Directory removals per second*)
- eliminación de ficheros por segundo (*File removals per second*)

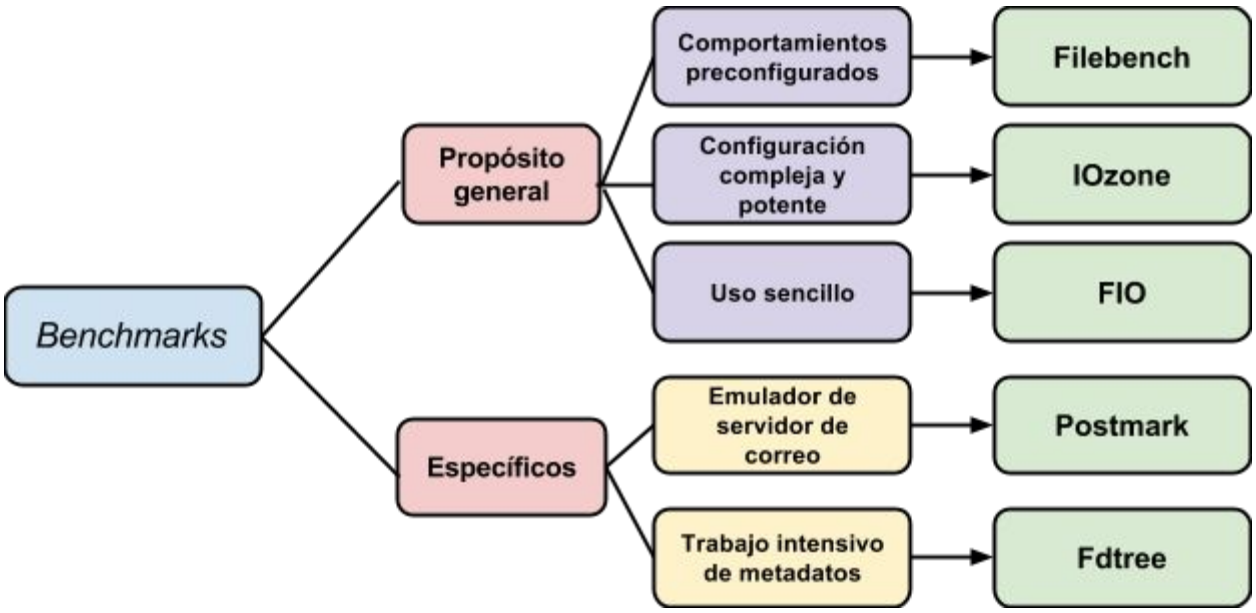


Figura 2.2: Esquema de los tipos de *benchmarks* y las características de los seleccionados.

3. Análisis y propuestas de diseño para la solución de almacenamiento

En este capítulo se va a ahondar en el estudio de las necesidades concretas de almacenamiento del grupo de investigación y de los requisitos que le imponen a un sistema de almacenamiento distribuido. Una vez que se tenga toda la información, se decidirá qué tecnologías son aptas y se diseñará un entorno para la solución.

3.1. Análisis de las necesidades

Un grupo de investigación universitario puede tener un abanico de necesidades de almacenamiento muy diversas. Tras mantener una serie de reuniones con integrantes de grupos de investigación, se ha ido extrayendo un conjunto de requisitos. La mayoría de ellos tiene que ver con el rendimiento, ya que la capacidad de almacenar datos se le presupone a cualquier sistema de almacenamiento distribuido. De lo que este estudio va a tratar es de decidir qué sistema puede hacer **mejor** lo que el grupo necesita.

3.1.1. Identificación de los interesados en el proyecto (*stakeholders*)

- Cliente/tutor del proyecto: es la persona que acepta llevar a cabo el proyecto y que asesorará durante la realización del mismo.
- Usuarios: los usuarios finales serán los beneficiarios de los resultados de este estudio, es decir, cualquier grupo de investigación de la universidad que desee optimizar su infraestructura de almacenamiento.
- Autor del proyecto: David Ventas Sierra, autor y máximo responsable de la correcta ejecución del proyecto.

3.1.2. Casos de uso

El propósito de los casos de uso es ilustrar mediante diagramas sencillos las principales interacciones de los diversos actores con el sistema. Muchos de los casos de uso que

aparecen a continuación parten de los requisitos *software*, que han sido depurados a partir de los requisitos de usuario.

3.1.2.1. Principales actores

Los actores que intervienen en el sistema son los siguientes:

- Administrador: es la persona encargada de instalar, configurar y mantener el *cluster*.
- Usuarios: hay dos tipos de usuarios, las personas y las aplicaciones que estas usan. Los programas que los miembros del grupo han identificado durante las reuniones como aplicaciones importantes son un servidor de correo electrónico, un servidor *web*, un repositorio de código en un sistema de control de versiones y un programa de virtualización.

3.1.2.2. Diagramas de casos de uso

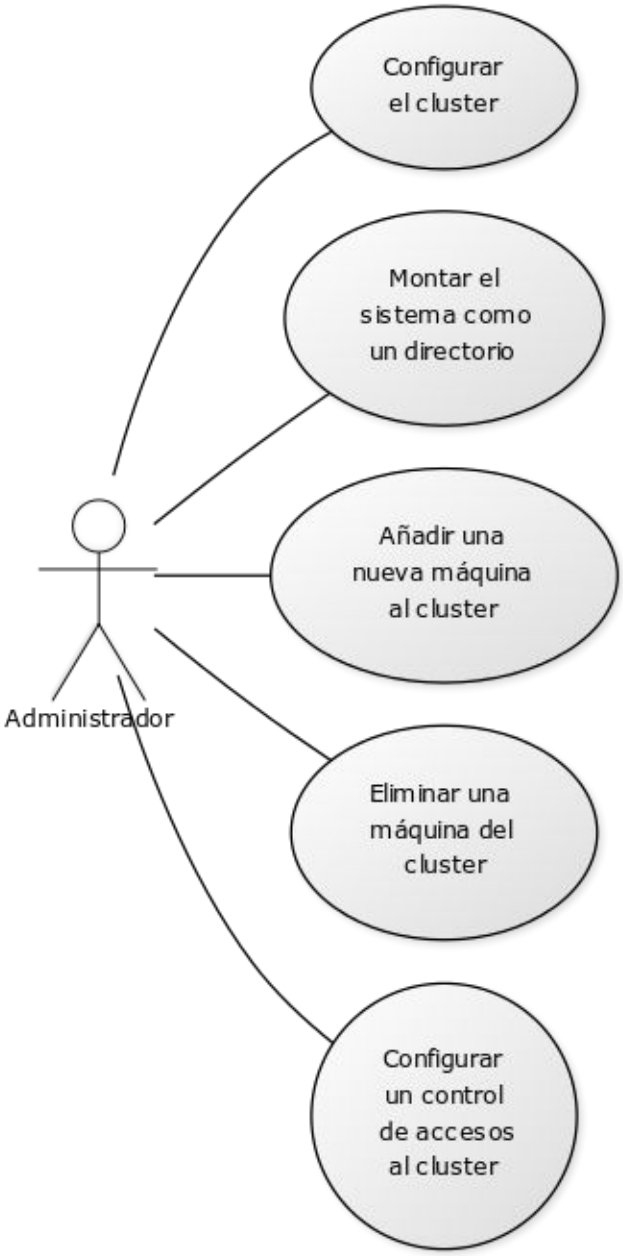


Figura 3.1: Diagrama de casos de uso del actor Administrador.

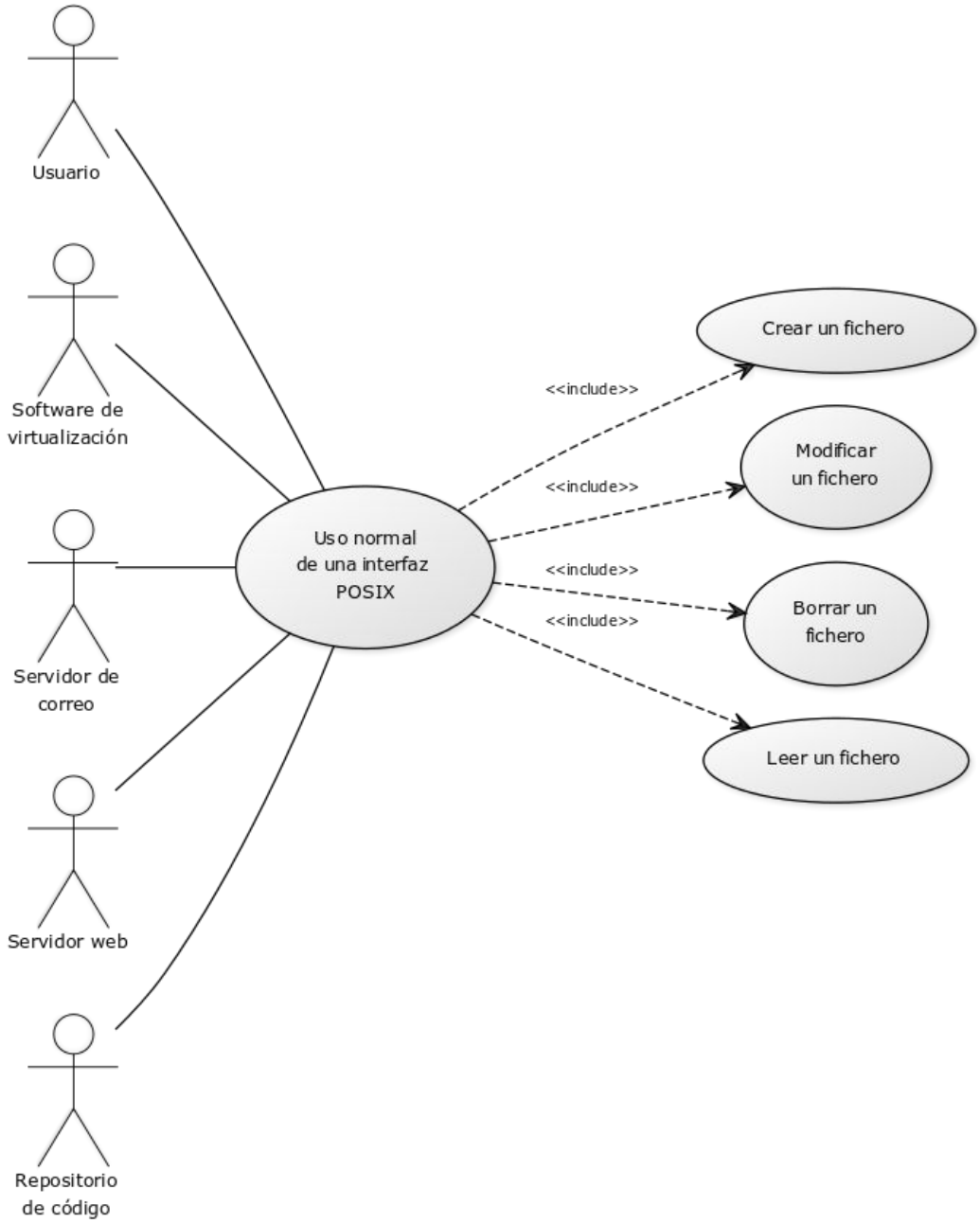


Figura 3.2: Diagrama de casos de uso del actor Usuario y de los actores *software*.

3.1.2.3. Especificación de los casos de uso

Es necesario completar los diagramas con una especificación que detalle toda la información de la interacción que describe el caso de uso. Para ello, se van a emplear tablas resumen compuestas por los siguientes campos:

- Identificador: es un código del tipo CU-XX que identifica a un único caso de uso. El término «XX» corresponde a un número natural de dos cifras.
- Nombre: el nombre del caso de uso. Es el mismo nombre del diagrama.
- Actores: el actor o actores que pueden intervenir en el caso de uso.
- Objetivo: describe la finalidad que busca el actor cuando realiza el caso de uso.
- Precondiciones: las circunstancias que deben darse para que el caso de uso se lleve a cabo correctamente.
- Poscondiciones: consecuencias de la ejecución del caso de uso.
- Escenario: descripción paso a paso de cómo se realiza el caso de uso.
- Condiciones de fallo: recogen los posibles fallos y las respuestas del sistema.

CU-XX	**Nombre**
Actores	
Objetivo	
Precondiciones	
Poscondiciones	
Escenario	
Condiciones de fallo	

Tabla 3.1: Plantilla para las tablas con las que describir los casos de uso.

CU-01	**Configurar el *cluster***
Actores	Administrador
Objetivo	Configurar el sistema de almacenamiento en todas las máquinas para que ofrezcan un volumen virtual de almacenamiento distribuido con el nivel de redundancia deseada.
Precondiciones	- Las máquinas en las que se instale el sistema de almacenamiento deben estar conectadas y debidamente configuradas. - El sistema de almacenamiento debe estar instalado.
Poscondiciones	- Se dispondrá de un *cluster* de almacenamiento distribuido.
Escenario	1. El administrador toma el manual de instalación y configuración. 2. Sigue los pasos para tener el *cluster* configurado.
Condiciones de fallo	n/a

Tabla 3.2: Caso de uso CU-01

CU-02	**Montar el sistema como un directorio**
Actores	Administrador
Objetivo	Tener un directorio en el que todo lo que se guarde sea repartido y replicado en el *cluster*.
Precondiciones	- Debe existir un *cluster* de almacenamiento en funcionamiento.
Poscondiciones	- El *cluster* está disponible para usarse desde un directorio en una máquina cliente.
Escenario	1. El administrador toma el manual de instalación y configuración. 2. Sigue los pasos para tener el *cluster* configurado.
Condiciones de fallo	n/a

Tabla 3.3: Caso de uso CU-02

CU-03	**Añadir una nueva máquina al *cluster***
Actores	Administrador
Objetivo	Aumentar la capacidad de almacenamiento o el rendimiento del *cluster* añadiendo nuevo *hardware*.
Precondiciones	- Debe existir un *cluster* de almacenamiento en funcionamiento.
Poscondiciones	- El *cluster* aumenta su capacidad de almacenamiento.
Escenario	1. El administrador toma el manual de instalación y configuración. 2. Sigue los pasos para tener el *cluster* configurado.
Condiciones de fallo	n/a

Tabla 3.4: Caso de uso CU-03

CU-04	**Eliminar una máquina del *cluster***
Actores	Administrador
Objetivo	Retirar una máquina del *cluster* por cualquier motivo, sin que ello conlleve la pérdida de los datos que están almacenados en ella.
Precondiciones	- Debe existir un *cluster* de almacenamiento en funcionamiento con más de dos máquinas.
Poscondiciones	- El *cluster* disminuye su capacidad de almacenamiento.
Escenario	1. El administrador toma el manual de instalación y configuración. 2. Sigue los pasos para tener el *cluster* configurado.
Condiciones de fallo	n/a

Tabla 3.5: Caso de uso CU-04

CU-05	**Configurar un control de acceso al *cluster***
Actores	Administrador
Objetivo	Configurar el mecanismo de seguridad de la tecnología elegida que permita seleccionar qué clientes están autorizados a montar el volumen.
Precondiciones	- Debe existir un *cluster* de almacenamiento en funcionamiento.
Poscondiciones	- Una vez configurado, solo ciertos clientes pueden acceder al cluster.
Escenario	1. El administrador toma el manual de instalación y configuración. 2. Sigue los pasos para tener el *cluster* configurado.
Condiciones de fallo	n/a

Tabla 3.6: Caso de uso CU-05

CU-06	**Uso normal de una interfaz POSIX**
Actores	Usuario, servidor de correo, servidor web, *software* de virtualización, repositorio de código.
Objetivo	El objetivo es interactuar con la carpeta del sistema de almacenamiento distribuido montada en el cliente como si se tratara de almacenamiento local, con todas las operaciones de manipulación de ficheros que admite el estándar POSIX.
Precondiciones	- El sistema de almacenamiento debe estar montado en una carpeta en el cliente. - El *software* que quiere interactuar con ella debe tener configurada esa carpeta como su directorio de trabajo.
Poscondiciones	- Los actores crean, leen, borran o modifican ficheros almacenados en el *cluster*.
Escenario	1. El actor realiza la operación que desee sobre un fichero en la carpeta donde se ha montado el sistema de ficheros distribuido.
Condiciones de fallo	n/a

Tabla 3.7: Caso de uso CU-06

Como se ha visto, no se puede ser muy específico en el escenario, ya que este depende de la tecnología escogida. Tampoco se desea serlo, porque lo importante es que el sistema sea capaz de responder a la necesidad planteada, no cómo lo haga. En cualquier caso, todas las tecnologías candidatas dispondrán de su manual en el siguiente capítulo, en el apartado de despliegue.

3.1.3. Requisitos

Existen dos tipos de requisitos:

- Requisitos de capacidad o funcionales: son aquellos que aportan información sobre las capacidades del sistema, lo que el sistema debe ser capaz de hacer. Recogen un servicio requerido por el cliente o el usuario.
- Requisitos de restricción o no funcionales: son los que imponen una limitación a la forma en que hay que prestar los servicios al usuario.

3.1.3.1. Requisitos del usuario

Para plasmar los requisitos del usuario obtenidos se va a utilizar una tabla con los siguientes campos:

- Identificador: es un código del tipo RU-XX, que identifica a un único requisito del usuario. Las letras «XX» corresponden a un número natural de dos cifras.
- Nombre: nombre del requisito.
- Descripción: se trata de una descripción del requisito tal y como ha sido expresada.
- Fuente: de quién se ha extraído este requisito.
- Necesidad: es el grado de necesidad que se tiene de que este requisito se cumpla. Puede ser «esencial» u «opcional».

La plantilla de la tabla es:

RU-XX	**Nombre**
Descripción	
Fuente	
Necesidad	Esencial/Opcional

Tabla 3.8: Plantilla para describir los requisitos del usuario.

RU-01	**Entorno operacional**
Descripción	El sistema de almacenamiento debe poder ejecutarse en el sistema operativo Linux, tanto los nodos donde se almacenen los datos como los clientes para acceder a ellos.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.9: Requisito del usuario RU-01

RU-02	**Seguridad**
Descripción	El sistema de almacenamiento debe poseer herramientas para poder determinar qué clientes están autorizados para conectarse al *cluster*.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.10: Requisito del usuario RU-02

RU-03	**Requisitos *hardware***
Descripción	El sistema de almacenamiento debe poder ejecutarse en *hardware* convencional y heterogéneo, para que no sea necesaria la adquisición de nuevos equipos si no se desea. Los recursos necesarios para la ejecución no deben ser muy elevados.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.11: Requisito del usuario RU-03

RU-04	**Escalabilidad**
Descripción	El sistema debe poder añadir o quitar *hardware* al *cluster* en cualquier momento.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.12: Requisito del usuario RU-04

RU-05	**Transparencia**
Descripción	El sistema de almacenamiento distribuido debe poder montarse en un directorio y comportarse como almacenamiento local.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.13: Requisito del usuario RU-05

RU-06	**Perfiles de almacenamiento**
Descripción	El sistema debe funcionar lo mejor posible almacenando discos duros virtuales, un servidor web, un servidor de correo, un repositorio de código y los ficheros que un miembro del grupo quiera guardar.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.14: Requisito del usuario RU-06

RU-07	**Entorno virtualizado**
Descripción	El sistema de almacenamiento debe funcionar correctamente en un entorno virtualizado, para garantizar un rápido despliegue en cualquier máquina.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.15: Requisito del usuario RU-07

RU-08	**Redundancia**
Descripción	El sistema de almacenamiento debe poder mantener réplicas coherentes de la información.
Fuente	Cliente
Necesidad	Esencial

Tabla 3.16: Requisito del usuario RU-08

3.1.3.2. Requisitos del *software*

Los requisitos del software se obtienen de la depuración de los requisitos del usuario. Se van a describir usando tablas con los siguientes campos:

- Identificador: es un código del tipo RS-XX, que identifica a un único requisito del *software*. Las letras «XX» corresponden a un número natural de dos cifras.
- Nombre: nombre del requisito.
- Descripción: se trata de una descripción más precisa del requisito.
- Fuente: de quién se ha extraído este requisito.

- Necesidad: es el grado de necesidad que se tiene de que este requisito se cumpla. Puede ser «esencial» u «opcional».
- Tipo: el requisito puede ser de capacidad o de restricción.
- Verificabilidad: indica la sencillez con la que se puede verificar que el requisito se cumple. Puede tomar los valores «alta», «media» o «baja».
- Origen: relaciona el requisito del *software* con el requisito del usuario del que proviene mediante su identificador.

No tiene sentido contemplar un campo «prioridad» como se hace habitualmente, porque no se trata de un desarrollo de *software* donde sea necesario establecer un orden para la implementación de los requisitos.

La plantilla de la tabla que se va a utilizar es:

RS-XX	**Nombre**
Descripción	
Fuente	
Necesidad	Esencial/Opcional
Tipo	Capacidad/Restricción
Verificabilidad	Alta/Media/Baja
Origen	RU-XX

Tabla 3.17: Plantilla para describir los requisitos del *software*.

RS-01	**Sistema operativo Linux Ubuntu**
Descripción	El sistema operativo donde se va a ejecutar el sistema de almacenamiento distribuido debe ser Ubuntu 14.04.2 LTS.
Fuente	Cliente y analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-01

Tabla 3.18: Requisito del *software* RS-01

RS-02	**Control de acceso**
Descripción	El sistema de almacenamiento debe incluir algún mecanismo que permita al administrador decidir qué clientes tienen permiso para conectarse y acceder a los datos.
Fuente	Cliente
Necesidad	Esencial
Tipo	Capacidad
Verificabilidad	Alta
Origen	RU-02

Tabla 3.19: Requisito del *software* RS-02

RS-03	**Ejecución en *commodity hardware***
Descripción	El sistema de almacenamiento no debe exigir más de 2 GB de RAM por máquina, y en ningún caso debe necesitar más de un procesador o un procesador con más de cuatro núcleos.
Fuente	Cliente y analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Media
Origen	RU-03

Tabla 3.20: Requisito del *software* RS-03

RS-04	**Añadir nuevas máquinas al sistema de almacenamiento**
Descripción	Se debe poder añadir nuevo *hardware* que aumente la capacidad de almacenamiento total sin necesidad de tener que dejar de prestar servicio.
Fuente	Analista
Necesidad	Esencial
Tipo	Capacidad
Verificabilidad	Alta
Origen	RU-04

Tabla 3.21: Requisito del *software* RS-04

RS-05	**Eliminar máquinas del sistema de almacenamiento**
Descripción	Se debe poder retirar del sistema una máquina sin dejar de prestar servicio y sin que esto provoque pérdidas de datos o se reduzca el número de copias que se tienen de los datos.
Fuente	Analista
Necesidad	Esencial
Tipo	Capacidad
Verificabilidad	Alta
Origen	RU-04

Tabla 3.22: Requisito del *software* RS-05

RS-06	**Montaje como directorio**
Descripción	El sistema de almacenamiento distribuido debe ser transparente para los usuarios o programas que lo utilicen y ser completamente compatible con el estándar POSIX.
Fuente	Cliente
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-05

Tabla 3.23: Requisito del *software* RS-06

RS-07	**Perfil de servidor de ficheros**
Descripción	El sistema de almacenamiento debe poder funcionar como un servidor de ficheros. El rendimiento mínimo exigible, así como las condiciones de carga de trabajo en las que se debe medir se encuentran descritas en los perfiles de almacenamiento.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-06

Tabla 3.24: Requisito del *software* RS-07

RS-08	**Perfil de servidor web**
Descripción	El sistema de almacenamiento debe poder guardar la estructura de una página web que un servidor web instalado ofrezca al exterior. El rendimiento mínimo exigible, así como las condiciones de carga de trabajo en las que se debe medir se encuentran descritas en los perfiles de almacenamiento.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-06

Tabla 3.25: Requisito del *software* RS-08

RS-09	**Perfil de programa de control de versiones**
Descripción	El sistema de almacenamiento debe poder guardar un repositorio de código gestionado por un sistema de control de versiones. El rendimiento mínimo exigible, así como las condiciones de carga de trabajo en las que se debe medir se encuentran descritas en los perfiles de almacenamiento.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-06

Tabla 3.26: Requisito del *software* RS-09

RS-10	**Perfil de servidor de correo electrónico**
Descripción	El sistema de almacenamiento debe poder guardar todos los datos del servidor de correo electrónico del grupo. El rendimiento mínimo exigible, así como las condiciones de carga de trabajo en las que se debe medir se encuentran descritas en los perfiles de almacenamiento.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-06

Tabla 3.27: Requisito del *software* RS-10

RS-11	**Perfil de almacén de discos duros virtuales**
Descripción	El sistema de almacenamiento debe poder alojar los discos duros de las máquinas virtuales del grupo de investigación. El rendimiento mínimo exigible, así como las condiciones de carga de trabajo en las que se debe medir se encuentran descritas en los perfiles de almacenamiento.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-06

Tabla 3.28: Requisito del *software* RS-11

RS-12	**Funcionamiento sobre VMware**
Descripción	El sistema de almacenamiento debe poder funcionar correctamente en un entorno virtual generado con VMware Workstation 11 y ejecutado en VMware Player.
Fuente	Analista
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-07

Tabla 3.29: Requisito del *software* RS-12

RS-13	**Tolerancia a fallos**
Descripción	El sistema de almacenamiento debe poder mantener un número configurable de copias actualizadas de los datos y gracias a ellas, mantener los datos disponibles de manera transparente.
Fuente	Cliente
Necesidad	Esencial
Tipo	Restricción
Verificabilidad	Alta
Origen	RU-07

Tabla 3.30: Requisito del *software* RS-13

3.1.3.3. Estudio de los perfiles de almacenamiento

Una parte fundamental de los requisitos en este proyecto son los perfiles de almacenamiento. En las reuniones de extracción de requisitos se ha hablado de las necesidades de almacenamiento que tiene un grupo de investigación. Estas necesidades se han ilustrado mediante ejemplos del tipo de programas que utiliza el grupo y que requieren más almacenamiento.

El objetivo de trazar un perfil de cada tipo de uso es poder reproducir estos escenarios mediante *benchmarks* que generen una utilización del almacenamiento disponible de manera análoga a como lo haría la aplicación real.

A continuación se presentan los resultados de un pequeño estudio que se ha hecho de estos tipos de programas para averiguar qué características distintivas tiene cada uno a la hora de usar el almacenamiento del sistema.

Todas las cifras requeridas en estos perfiles de almacenamiento son para un despliegue con el mínimo número de nodos posible y con el mínimo nivel de redundancia. Todos los sistemas distribuidos analizados mejoran su rendimiento cuantos más nodos conforman su *cluster*.

- **RS-07. Un servidor de ficheros:** este caso es completamente arbitrario. Consiste en que un usuario almacena los ficheros que cree conveniente en el sistema. Depende por completo de la finalidad de los proyectos que el grupo de investigación lleve a cabo. Pueden ser desde pequeños archivos de texto hasta vídeos de varios *gigabytes*. Para reproducir este tipo de uso, se utilizará una distribución de ficheros aleatoria con tamaños desde los 10 KB hasta los 100 MB, que son los más probables para un usuario de estas características. El número de lecturas y escrituras estará equilibrado y se pide un ancho de banda mínimo de 200 KB/s para cada usuario si hay 4 usuarios escribiendo y leyendo intensivamente al mismo tiempo. Para un pico de trabajo de 50 usuarios accediendo simultáneamente se requiere un mínimo de 80 KB/s.

- **RS-08. Un servidor web:** el uso que le puede dar un grupo de investigación a su página web es sobre todo informativo, un lugar donde darse a conocer y exponer los diferentes proyectos a la comunidad universitaria. Un uso así produce casi de forma exclusiva lecturas en el sistema. El servidor maneja sobre todo archivos pequeños tipo html y hojas de estilo como archivos css. También alberga un pequeño porcentaje de ficheros más grandes como recursos adicionales que puedan resultar de utilidad, (archivos multimedia como fotos, vídeos, etcétera). El tamaño medio de los ficheros almacenados en los servidores web de las mil primeras páginas más visitadas, es aproximadamente de 1700 KB si ha seguido la progresión que mostraba en 2014 [5] (Website Optimization, 2014). Se espera un máximo de 50 consultas simultáneas con picos de hasta 80 en el peor de los casos, y se requiere un rendimiento de al menos 150 KB/s y 70 KB/s por usuario en cada caso respectivamente. Siempre se manipulan ficheros con un tamaño comprendido en un rango de 1536 KB a 2048 KB.

- **RS-09. Un repositorio de código:** este perfil representa un repositorio donde poder almacenar el código fuente de los programas que estén en desarrollo. Este uso también se caracteriza por un elevado número de ficheros de escaso tamaño. En este entorno las lecturas son más numerosas que las escrituras. En cuanto al tamaño, depende mucho del tipo de proyecto y del lenguaje utilizado. No es lo mismo un programa convencional que un juego con todos los elementos multimedia incluidos en el repositorio, ni el número de líneas de un código en C o en Python. Por ejemplo, en Java se recomienda no exceder las 2000 líneas de código [6, 7] (Sun Microsystems, 1997),

(Ayuntamiento de Madrid, 2015), aunque lo recomendable son unas 200. En la práctica, las métricas de los proyectos consultados fijan la media entre 67 y 83. Un fichero de texto de entre 70 y 200 líneas ocupa entre 1,80 y 5,17 KB. Habitualmente un desarrollador no hace más que unas pocas lecturas o escrituras al día. Las pruebas se realizarán suponiendo que 3 desarrolladores quieran descargar de una vez o hacer una subida a un repositorio al mismo tiempo. Se requiere que ningún sistema tenga una media de lectura inferior a 1 MB/s ni una media de escritura inferior a 400 KB/s con las condiciones explicadas.

- **RS-10. Un servidor de correo electrónico:** una aplicación de estas características requiere un directorio por cada buzón de correo, con sus correspondientes subdirectorios configurados por el usuario. Cada correo se almacena en un fichero. Suele haber más lecturas que escrituras en el sistema. Un mensaje de correo electrónico normal suele situarse entre 10 y 30 KB. Los archivos adjuntos pueden disparar la media hasta los 75 KB aproximadamente [8] (Tschabitscher, 2012). El número de buzones que puede haber en un grupo de investigación tampoco es una cifra muy elevada. Puede situarse entre 10 y 50. Suponiendo el peor escenario, en el que todos los clientes arrancan y se conectan a la vez para descargar su correo, se exige un ancho de banda de 25 KB/s por cada cliente, para que ningún correo tarde más de 3 segundos en ser servido.

- **RS-11. Discos duros virtuales:** si el grupo de investigación emplea máquinas virtuales para sus pruebas, compilaciones, etcétera, pueden necesitar almacenar estas máquinas junto con sus discos duros virtuales en la infraestructura de almacenamiento distribuido. Los accesos a los discos duros virtuales son iguales que los de los discos físicos; constantes lecturas y escrituras, *swapping*,... Lo habitual es que haya más lecturas que escrituras en este escenario. Aunque no es necesario solicitar al *hardware* real toda la capacidad del disco duro virtual en el momento de su creación, el tamaño mínimo es el del sistema operativo instalado, más el espacio libre que este requiera. Los discos duros virtuales se pueden almacenar en un solo fichero, aunque es posible y más habitual fraccionarlos en ficheros de 4 GB. De no fraccionarlos, se pueden originar ficheros de decenas y hasta miles de *gigabytes*. Se requiere que se utilicen ficheros de 4 GB con accesos secuenciales y aleatorios, como accedería un sistema de virtualización. La proporción de lecturas y escrituras en el disco duro virtual vendrá determinada por el tipo de operación que se lleve a cabo en la máquina virtual. Se solicitan tres tipos de pruebas con sus respectivos rendimientos mínimos:

- dos procesos de lectura y dos procesos de escritura que realizan operaciones secuenciales. Para este escenario se requiere que el ancho de banda mínimo sea de 5 MB/s por proceso de lectura y de 2,5 MB/s por cada proceso de escritura.
- dos procesos que realizan simultáneamente lecturas y escrituras totalmente aleatorias. Este es un caso de altísima exigencia, una situación que es poco probable que se dé durante mucho tiempo. Se necesita un ancho de banda mínimo de 125 KB/s por cada proceso.
- cuatro procesos, de los cuales tres son de lectura y uno de escritura. Se trata, como la anterior, de un caso excepcional, pero se requiere que si esto ocurre se consiga un ancho de banda de al menos 125 KB/s por proceso.

3.1.3.4. Relación entre requisitos

La siguiente tabla muestra de qué requisito del usuario ha salido cada requisito del *software*:

Requisitos del usuario	Requisitos del *software*
RU-01	RS-01
RU-02	RS-02
RU-03	RS-03
RU-04	RS-04, RS-05
RU-05	RS-06
RU-06	RS-07, RS-08, RS-09, RS-10, RS-11
RU-07	RS-12
RU-08	RS-13

Tabla 3.31: Relación entre requisitos del *software* y requisitos del usuario.

3.1.4. Matriz de trazabilidad de casos de uso a requisitos del *software*

La matriz de trazabilidad de casos de uso a requisitos del *software* es una tabla que relaciona los requisitos del *software* obtenidos con los casos de uso que los originan, y su principal uso es poder hacer un seguimiento de cómo han evolucionado y de dónde parten.

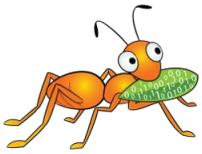
	CU-01	CU-02	CU-03	CU-04	CU-05	CU-06
RS-01	X	X	X	X	X	
RS-02					X	
RS-03			X			
RS-04			X			
RS-05				X		
RS-06		X				
RS-07						X
RS-08						X
RS-09						X
RS-10						X
RS-11						X
RS-12						X
RS-13	X					

Tabla 3.32: Matriz de trazabilidad de casos de uso a requisitos del *software*.

3.2. Tecnologías propuestas y diseño de las posibles soluciones

A continuación se presentan las principales soluciones independientes de almacenamiento distribuido que por sí solas pueden ser candidatas a dar la respuesta más adecuada a los requisitos anteriormente presentados.
El funcionamiento detallado de cada una de las soluciones se recoge en su propio capítulo anexo.

3.2.1. Solución basada en GlusterFS



GlusterFS es un sistema de ficheros distribuido. Está indicado para todo tipo de usos, desde *streaming* de multimedia, hasta análisis de grandes cantidades de datos. Gluster puede utilizarse con Swift (almacenamiento de objetos) empleando un traductor.

Existe una numerosa comunidad en torno a Gluster y tiene una gran empresa detrás, ya que fue adquirido en 2011 por Red Hat, Inc.

Para poner a prueba a GlusterFS como posible solución, se propone crear un entorno virtual tal y como los requisitos determinan, para realizar las mediciones. El entorno constará de una máquina que actuará como cliente y dos máquinas que serán los nodos de almacenamiento. Es el número mínimo de nodos que admite GlusterFS para funcionar ofreciendo las características requeridas.

3.2.2. Solución basada en Ceph

Ceph es un sistema de almacenamiento distribuido de objetos que cuenta con interfaces para poder comportarse como un dispositivo de bloques y como un sistema de ficheros POSIX.

Es un sistema asentado, popular y en constante desarrollo y mejora. Cuenta con una enorme comunidad con más de cincuenta desarrolladores habituales y más de quinientos contribuidores esporádicos. Sus casi diez millones de descargas ilustran hasta qué punto está extendido [9] (Inktank Storage Inc. 2014, Metrics).

Su versatilidad gracias a sus distintas interfaces, le hace ser un candidato perfecto para su estudio.

Cuando Ceph empezó a tener una cierta expansión, se fundó la empresa Inktank Storage para el mantenimiento y soporte de Ceph. Al igual que GlusterFS, Inktank fue adquirida por Red Hat, Inc.

A la hora de construir un entorno que cubra las necesidades expresadas en los requisitos, como ser un sistema de ficheros montable que haga las veces de sistema de ficheros local, el entorno de pruebas mínimo necesario consta de dos nodos de almacenamiento, un nodo que haga de monitor del mapa del *cluster*, otro nodo que haga de servidor de metadatos y por último el nodo cliente, donde se montará Ceph como sistema de ficheros local y desde donde se ejecutarán los *benchmarks*. Esto hace un total de cinco máquinas virtuales.

3.2.3. Solución basada en LizardFS

LizardFS es un sistema de ficheros distribuido de alta disponibilidad montable como un sistema POSIX. Se trata de un *fork* del sistema de ficheros MooseFS, desde el año 2012.

Es un sistema joven, aunque el original del que deriva tiene un considerable tiempo de rodaje. LizardFS ya se emplea en numerosas empresas, según el fabricante. Está suficientemente asentado como para considerarlo en este estudio aunque su documentación es realmente escasa.

Destaca por tener un diseño tan complejo como el de otras soluciones pero muy sencillo de utilizar. Dispone de una vista web para monitorizar el estado del sistema en todo momento.

Toda la infraestructura se instala en Linux y además dispone de un cliente para Windows.

Siguiendo las recomendaciones del fabricante, el entorno de pruebas constará de cinco máquinas virtuales.

3.2.4. Solución basada en Hadoop (HDFS)

Hadoop es un *framework* de Apache Software Foundation para gestionar almacenamiento distribuido. Está basado en las las especificaciones de Google de su MapReduce para el manejo de los datos y del sistema de ficheros de Google (GFS) implementado como HDFS (Hadoop *Distributed File System*).

Hadoop está especialmente indicado para el manejo de ficheros de grandes tamaños, desde gigabytes a terabytes. Sin embargo, puede presentar un rendimiento peor si su tarea consiste en almacenar grandes cantidades de ficheros pequeños.

HDFS presenta el máximo rendimiento cuando almacena ficheros que no se van a ver constantemente modificados.

Hadoop se ha convertido en una opción muy popular entre los sistemas de ficheros distribuidos desde su creación en 2005.

3.2.5. Comparación de las principales características

	GlusterFS	Ceph	LizardFS	HDFS
Tipo de almacenamiento	Ficheros	Objetos	Ficheros	Ficheros
Utiliza un servidor de metadatos	No	Sí	Sí	Sí
Número mínimo de nodos recomendado para funcionar	2	4	4	2
Posibilidades de uso	Ficheros, objetos	Ficheros, objetos, dispositivo de bloques	Ficheros	Ficheros, aplicaciones Hadoop
Capacidad máxima	Cerca de los 72 *brontobytes* o *hellabytes* (10^{27})	Cerca del *exabyte* (10^{18})	-	Ficheros de *terabytes*
Tipo de licencia	GNU/GPL v2, v3 LGPL	LGPL	GPLv3	Apache License 2.0
Sistema operativo	Linux	Linux	Linux. Cliente opcional Windows.	Multiplataforma
Propietario	RedHat	Inktank (RedHat)	Skytechnology	Apache Software Foundation

Tabla 3.33: Tabla resumen de las principales características de GlusterFS, Ceph, LizardFS y HDFS.

3.2.6. Comparación de las características más relevantes para los requisitos

	GlusterFS	Ceph	LizardFS	HDFS
Funciona sobre Ubuntu 14.04.2 LTS				
Puede implementar controles de acceso				
Ejecución en *commodity hardware*				
Añadir máquinas al sistema de almacenamiento en caliente				
Eliminar máquinas del sistema de almacenamiento en caliente				
Se puede montar en una carpeta y sigue el estándar POSIX				
Se puede configurar número de copias redundantes				

Tabla 3.34: Tabla resumen de las características relevantes para los requisitos de GlusterFS, Ceph, LizardFS y HDFS.

Para seguir siendo parte de este estudio, todas las tecnologías deben incluir las características recogidas en la tabla 3.34.

Toda la información ha sido obtenida de la documentación oficial de los sistemas de almacenamiento. Aunque HDFS se puede montar a través de FUSE, no es completamente compatible con el estándar POSIX, ya que han recortado algunas de las operaciones. Esto invalida a Hadoop y a su HDFS como sistema candidato para seguir siendo analizando en este estudio, por no cumplir el RS-06.

4. Evaluación de las soluciones propuestas

4.1. Descripción del entorno

Para poder evaluar las soluciones propuestas en el capítulo tercero es necesario contar con una infraestructura sobre la que poder instalarlas y ponerlas a prueba. Como se ha explicado anteriormente, se ha optado por analizar el rendimiento de estas soluciones en un entorno virtual, con la mayor cantidad de discos duros independientes posibles. Es bien sabido que esto puede conllevar alguna limitación, pero al tratarse de las mismas limitaciones para todas las soluciones, todas ellas sufrirán penalizaciones semejantes y los datos que se obtengan seguirán siendo de interés.

Por otro lado, en ciertos servicios de computación en la nube, se ofrecen alquileres de máquinas virtuales para lo que el cliente estime oportuno, y es muy habitual el montaje de *clusters* de almacenamiento distribuido sobre estas máquinas virtuales. Al no saber con certeza en cuántas máquinas físicas se ejecutan esas máquinas virtuales, este estudio puede servir también como análisis del peor escenario posible.

A continuación se describen todos los elementos que componen el entorno de pruebas de las diversas tecnologías.

4.1.1. Entorno físico

La máquina real sobre la que se ejecutan las pruebas tiene las siguientes características:

- CPU: Intel Core i7 950, 3.07 Ghz. 4 núcleos, Hyper Threading.
- RAM: 10 GB, 1066 Mhz, Triple Channel.
- Disco duro: Western Digital WD Caviar Black, 1 TB, SATA 6 Gb/s, 7200 RPM, buffer de 64 MB.
- Disco duro: Western Digital WD Caviar Green, 2 TB, SATA 6 Gb/s, 7200 RPM, buffer de 64 MB.
- Disco duro: Seagate Barracuda, 2 TB, SATA 6 Gb/s, 7200 RPM, buffer de 64 MB.
- S.O.: Windows 7 Professional, Service Pack 1, 64 bits.
- Software de virtualización: VMware Workstation 11.0.0 build-2305329.
- Ofimática: Memoria íntegramente escrita en Google Drive y Google Docs.

4.1.2. Entorno virtual

Todos los entornos de pruebas son virtuales. Los nodos que componen el *cluster* son máquinas virtuales con las siguientes características:

- Procesador dual core.
- 1 GB de RAM.
- Disco duro SCSI de 20 GB para el S.O.
- Disco duro adicional de 10 GB.
- Interfaz de red conectada a red virtual con el resto de máquinas virtuales del entorno.
- Interfaz de red conectada a Internet para la descarga del software necesario.
- Sistema Operativo Linux Ubuntu 14.04.2 LTS.

Se dispone de cinco nodos, llamados Nodo1, Nodo2, Nodo3, Nodo4 y Nodo5. Además del *hardware* descrito, Nodo1, Nodo2 y Nodo3 disponen de un disco duro adicional de 100 GB. Generalmente Nodo1 y Nodo2 serán nodos de almacenamiento y Nodo3 será cliente.

4.1.3. Diagramas de los distintos escenarios

A continuación se incluye un diagrama del diseño de los distintos entornos de pruebas utilizados.

4.1.3.1. Entorno para Ceph

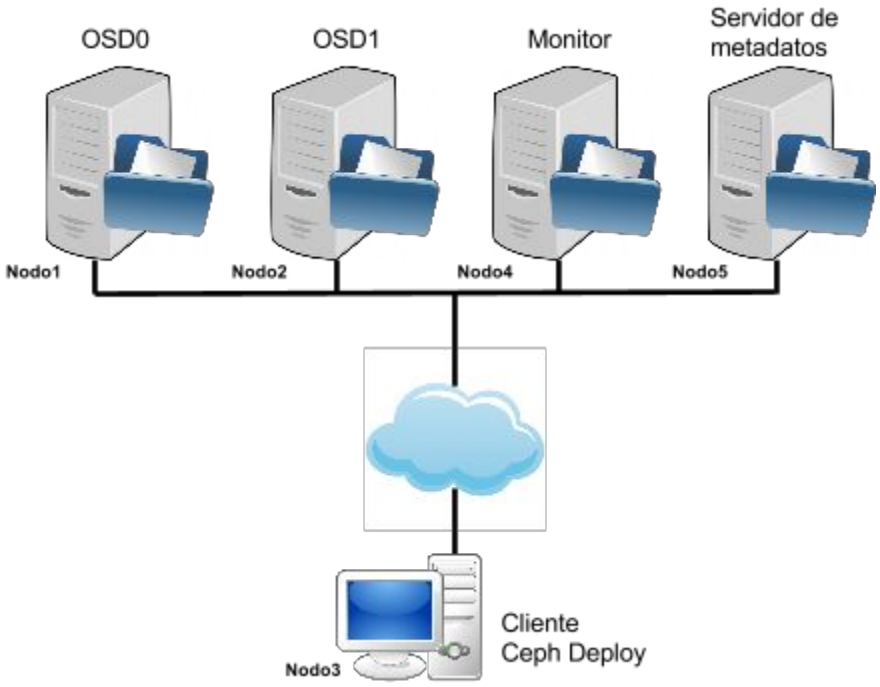


Figura 4.1: Esquema del entorno de pruebas para Ceph. Se puede observar qué rol adquiere cada nodo.

4.1.3.2. Entorno para GlusterFS

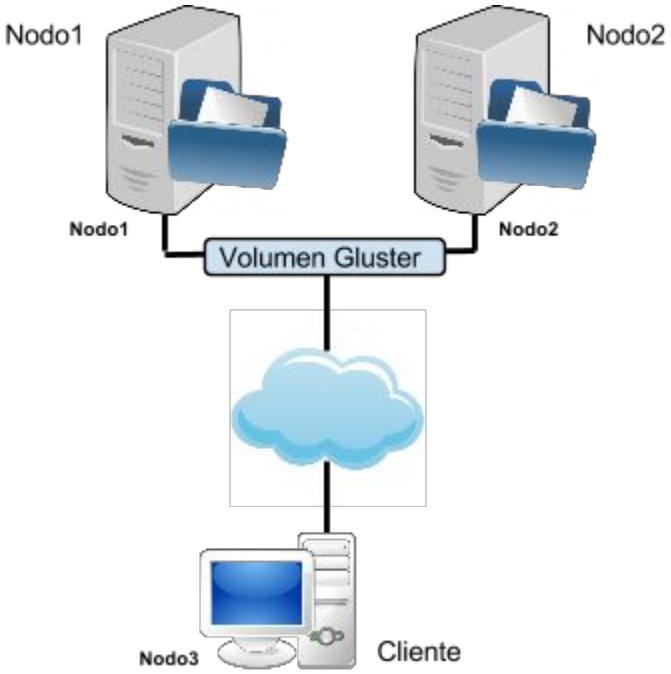


Figura 4.2: Esquema del entorno de pruebas para GlusterFS. Se puede observar qué rol adquiere cada nodo.

4.1.3.3. Entorno para LizardFS

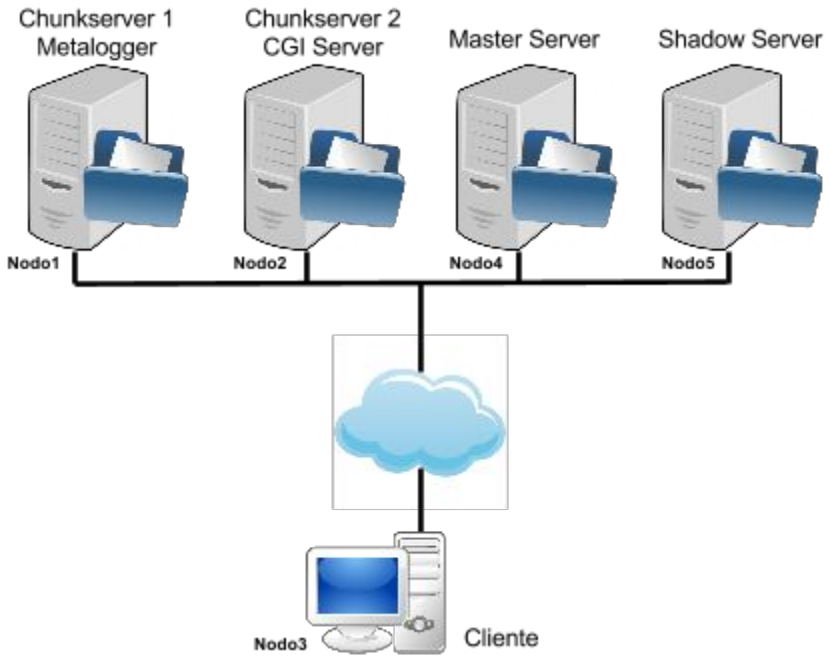


Figura 4.3: Esquema del entorno de pruebas para Ceph. Se puede observar qué rol adquiere cada nodo.

4.2. Despliegue del entorno

A continuación se detalla el proceso para reproducir el entorno de pruebas

4.2.1. Despliegue de las máquinas virtuales

En este apartado se explica el proceso de creación de una máquina virtual, obviando la instalación del *software* de virtualización.

VMware dispone de un sistema de instalación simplificada. Solo es necesario disponer de una imagen ISO del sistema operativo que se quiere instalar y suministrar al programa los datos

que solicite antes de iniciar la instalación. En el caso de Ubuntu, con el nombre de usuario para el administrador y su contraseña es suficiente.

Los pasos detallados del proceso que hay que seguir se encuentran en el apartado «Anexo 3» de esta memoria.

4.2.2. Despliegue de Ceph

La configuración que vamos a usar se compone de una máquina principal (master) y varios clientes (client1, client2, client3). La instalación y configuración se hace en tres fases; acondicionamiento y configuraciones previas del *cluster*, creación del *ceph-deploy* y configuración del cliente para que pueda conectarse a al *cluster* de Ceph.

Acondicionamiento y configuraciones previas del *cluster*:

1. Master: acondicionar la máquina para el funcionamiento de Ceph e instalarlo.

- Formatear con XFS.

```
apt-get install xfs xfsprogs
mkfs.xfs -i size=512 /dev/sdb1
```

- Actualizar apt-get.

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc'
sudo apt-key add -
echo deb http://ceph.com/debian-firefly/ $(lsb_release -sc) main | sudo tee
/etc/apt/sources.list.d/ceph.list
sudo apt-get update
```

- Instalar Ceph.

```
sudo apt-get install ceph-deploy
```

- Instalar Openssh-server.

```
sudo apt-get install openssh-server
```

2. Nodos: preparar usuarios e instalaciones varias.

- Instalar Openssh-server.

```
sudo apt-get install openssh-server
```

- Instalar *Network Time Protocol*, que sirve para sincronizar el reloj de los nodos con unos pocos milisegundos de error.

```
sudo apt-get install ntp
```

- Crear un usuario en todos los nodos.

```
ssh user@ceph-server
sudo useradd -d /home/cephuser -m cephuser
sudo passwd cephuser
```

- Conceder permisos de administrador a este usuario.

```
sudo echo "cephuser ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/cephuser
sudo chmod 0440 /etc/sudoers.d/cephuser
```

3. Master: configurar SSH.

- Generar clave SSH sin usuario ni contraseña.

```
sudo ssh-keygen
```

- Copiar a los nodos.

```
sudo ssh-copy-id cephuser@nodo1
sudo ssh-copy-id cephuser@nodo2
sudo ssh-copy-id cephuser@nodo3
```

- Modificar la configuración SSH.

```
sudo vim ~/.ssh/config
Host node1
        Hostname nodo1
        user cephuser
Host node2
        Hostname nodo2
        user cephuser
Host node3
        Hostname nodo3
        user cephuser
```

4. Master y nodos: terminar de configurar la red.

- Abrir **/etc/sysconfig/network-scripts** y ver que el fichero **ifcfg-{iface}** tiene **ONBOOT** puesto a ***yes***.
- Abrir los puertos necesarios para que las máquinas se puedan comunicar.

```
sudo iptables -A INPUT -i {iface} -p tcp -s {ip-address}/{netmask} --dport 6789 -j ACCEPT
sudo /sbin/service iptables save
```

Creación del *ceph-deploy*. Todos los pasos se hacen en la máquina master:

1. Reiniciar todo.

```
ceph-deploy purgedata node2 node4 node5
ceph-deploy forgetkeys
ceph-deploy purge node2 node4 node5
```

2. Crear carpeta.

```
mkdir my-cluster
cd my-cluster
```

3. Crear el monitor de Ceph.

```
ceph-deploy new node2
```

4. Configurar Ceph.

```
sudo vim ~/.ceph/config
//Debajo de la sección [global] poner:
osd pool default size = 2
//Si se tuviesen varios puertos de red (no es el caso), se debería modificar la siguiente línea:
public network = {ip-address}/{netmask}
```

5. Instalar los nodos.

```
ceph-deploy install admin-node node2 node4 node5
```

6. Instalar el monitor.

```
ceph-deploy mon node2
```

7. Crear las carpetas de los OSD en cada nodo a través de SSH.

```
ssh nodo2
sudo mkdir /var/local/osd0
exit

ssh nodo4
sudo mkdir /var/local/osd1
exit


ssh nodo5
sudo mkdir /var/local/osd3
exit
```

8. Crear los nodos del *cluster*.

```
ceph-deploy osd prepare node2:/var/local/osd0 nodo4:/var/local/osd1 nodo5:/var/local/osd2
ceph-deploy osd activate node2:/var/local/osd0 nodo4:/var/local/osd1 nodo5:/var/local/osd2
ceph-deploy admin admin-node nodo2 nodo4 nodo5
```

9. Cambio de permisos y comprobación del estado del *cluster*.

```
sudo chmod +r /etc/ceph/ceph.client.admin.keyring
ceph health
```

10. Por último, hay que crear el servidor de metadatos para poder interactuar con Ceph como un sistema de ficheros distribuido.

```
ceph-deploy mds create node2
```

Configuración del cliente:

1. Master: se da de alta el cliente.

```
ceph-deploy install ceph-client
ceph -s -m nodo3 -k ~/.client.admin.keyring
```

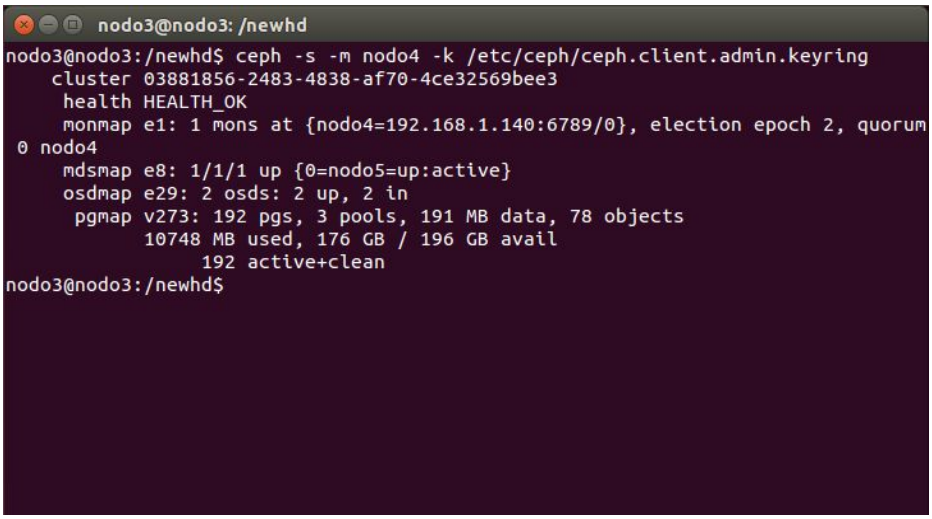


Figura 4.4: Salida del comando de alta del cliente en Ceph.

2. Master: se añade la clave.

```
cat ceph.client.admin.keyring
```

Copiar la clave a usar bajo [client.admin]

Salvar en un fichero la clave, por ejemplo *admin.secret* .

3. Cliente: se crea el sistema de ficheros para acceder a Ceph.

```
//<pg_num> por ejemplo es 12 y <fs_name> puede ser cephfs
ceph osd pool create cephfs_data <pg_num>
ceph osd pool create cephfs_metadata <pg_num>
ceph fs new <fs_name> cephfs_metadata cephfs_data
```

4. Cliente: para finalizar, se monta Ceph.

```
sudo mkdir ~/mycephfs
sudo ceph-fuse -m nodo2:6789 ~/mycephfssudo ceph-fuse -k ./ceph.client.admin.keyring -m
192.168.0.1:6789 ~/mycephfs
```

Referencia rápida

A continuación se resume la instalación al mínimo número de comandos imprescindibles para desplegar Ceph. Prácticamente todo el proceso se puede llevar a cabo desde la máquina que tiene el rol de administrador. Las configuraciones en los diversos componentes del *cluster* se puede hacer vía *ssh*.

1. En el administrador:

- sudo apt-get install aptitude
- sudo aptitude update
- sudo aptitude install ceph-deploy

2. En todos los nodos del *cluster*:

- sudo aptitude install NTP
- sudo aptitude install openssh-server
- sudo useradd -d /home/cephuser -m cephuser
- sudo passwd cephuser
- sudo echo “cephuser ALL = (root) NOPASSWD:ALL” | sudo tee /etc/sudoers.d/cephuser
- sudo iptables -A INPUT -i {interfaz de red} -p tcp -s {dirección IP}/{máscara de red} --dport 6789 -j ACCEPT
- sudo iptables -A INPUT -i {interfaz de red} -p tcp -s {dirección IP}/{máscara de red} --dport 6800:7300 -j ACCEPT
- sudo iptables-save

3. En los nodos destinados a ser OSD:

- sudo mkdir /newhd
- sudo mount {ruta del punto de montaje} /newhd

4. En el administrador:

- ssh-keygen //Introducir ruta y passphrase.
- ssh-copy-id -i sshkey.pub cephuser@{nombre del nodo} //A todos los nodos.
- sudo vim ~/.ssh/config //Añadir la información de cada nodo. Consultar guía detallada. guardar y salir.
- ceph-deploy new nodo4

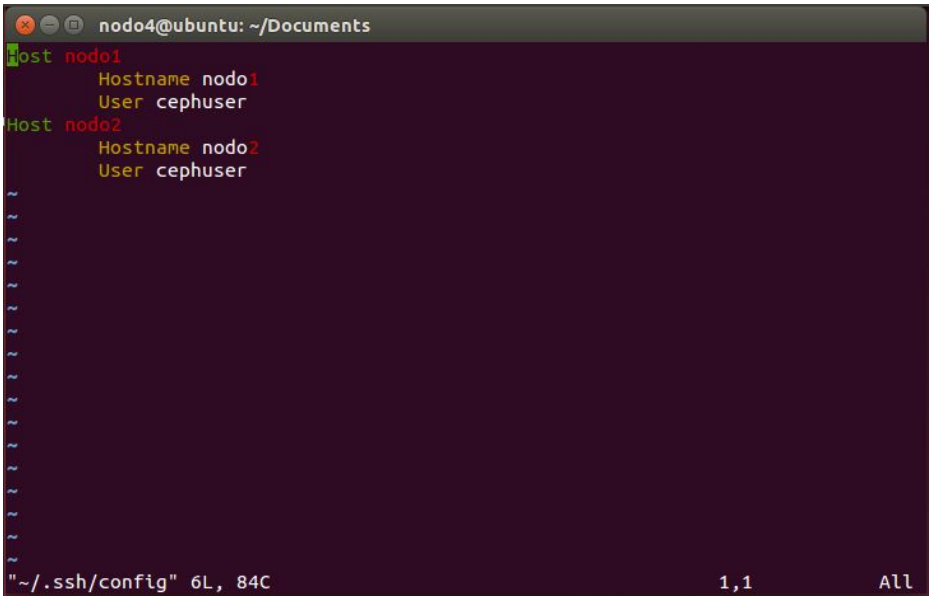


Figura 4.5: Fichero «~/.ssh/config». Es necesario añadir la información de cada nodo del sistema tal y como muestra la imagen.

- sudo vim ~/.cephdeploy.conf
- //Añadir la línea «osd pool default size = 2» guardar y salir.
- ceph-deploy --username cephuser install nodo3 nodo1 nodo2 nodo4 nodo5
- ceph-deploy --username cephuser mon create-initial

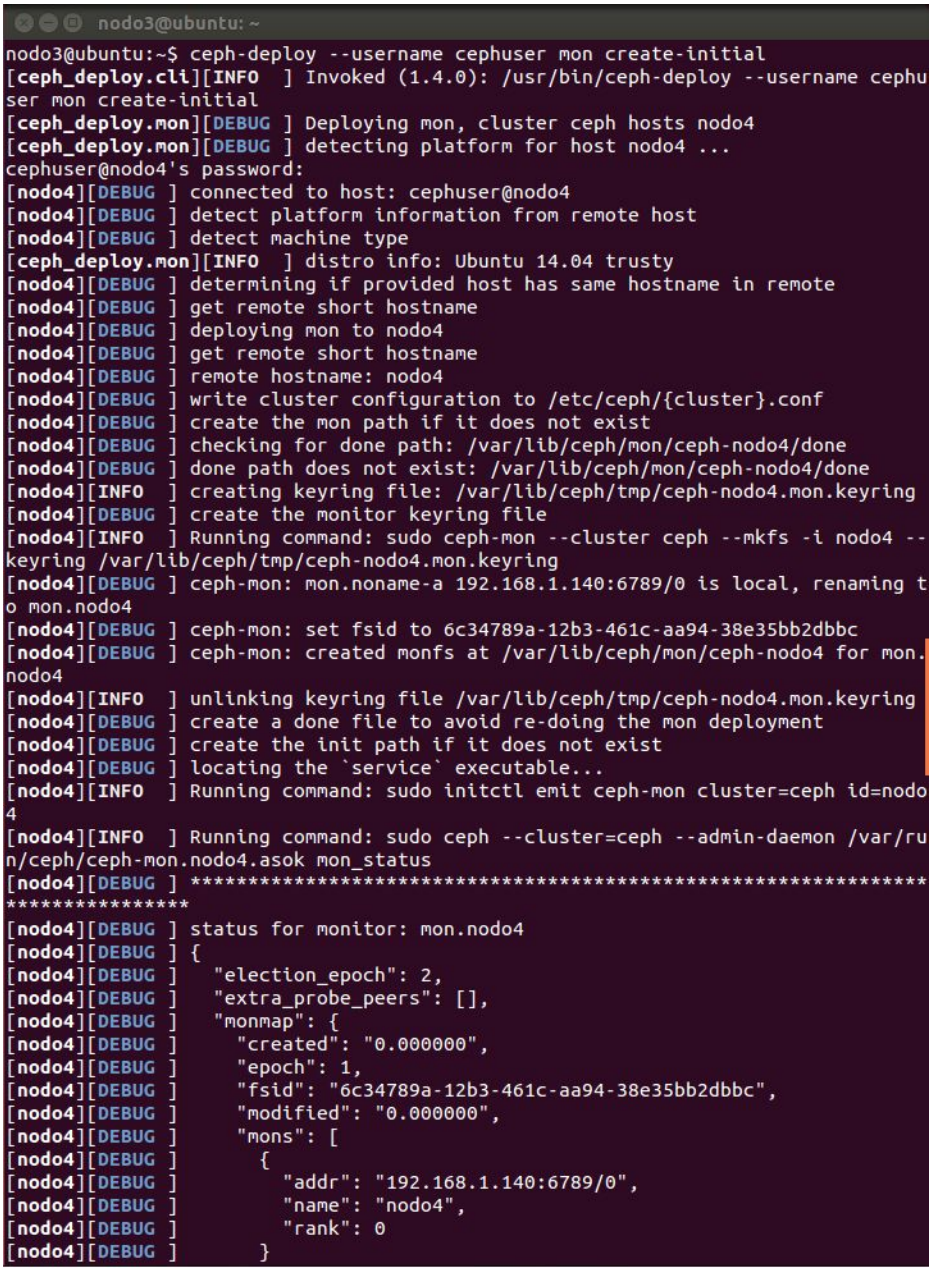


Figura 4.6: Salida del comando mon create-initial de Ceph. El objetivo de este comando es crear los monitores del sistema y el quórum.

- `ceph-deploy --username cephuser osd prepare nodo1:/newhd nodo2:/newhd`

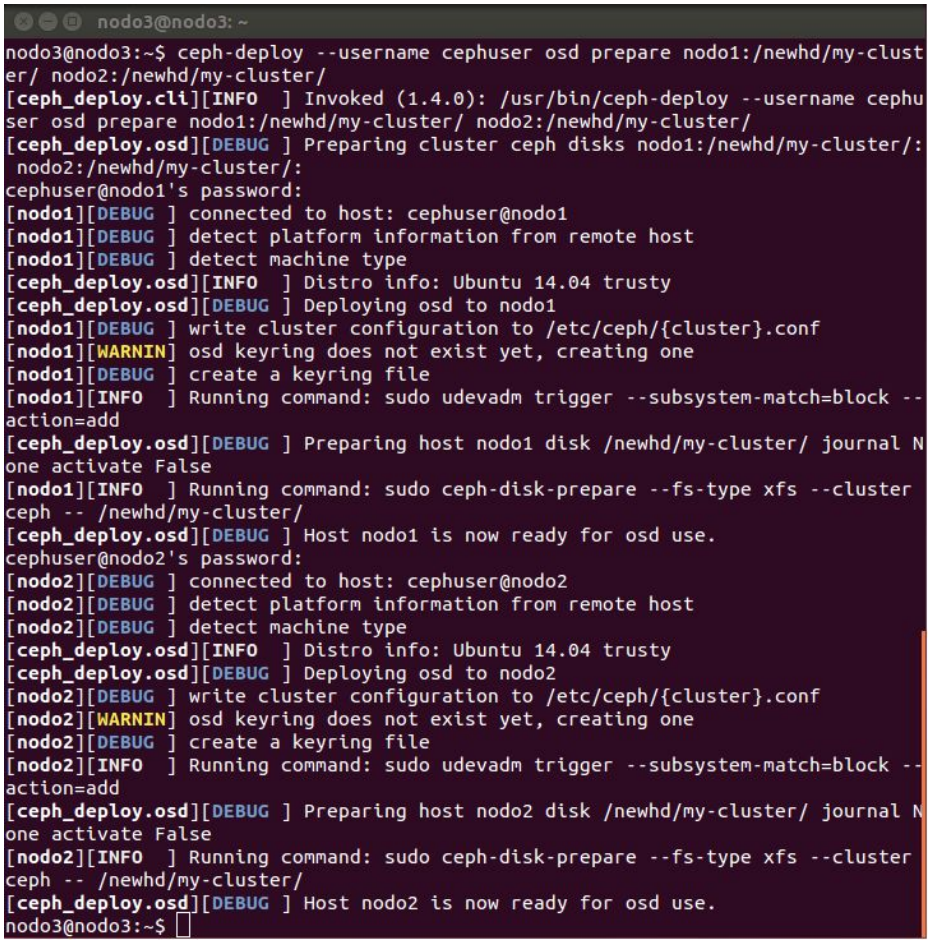


Figura 4.7: Salida del comando osd prepare de Ceph. Este comando envía todo lo necesario al nodo para convertirlo en OSD.

- `ceph-deploy --username cephuser osd activate nodo1:/newhd nodo2:/newhd`
- `ceph-deploy --username cephuser admin nodo3 nodo1 nodo2 nodo4 nodo5`

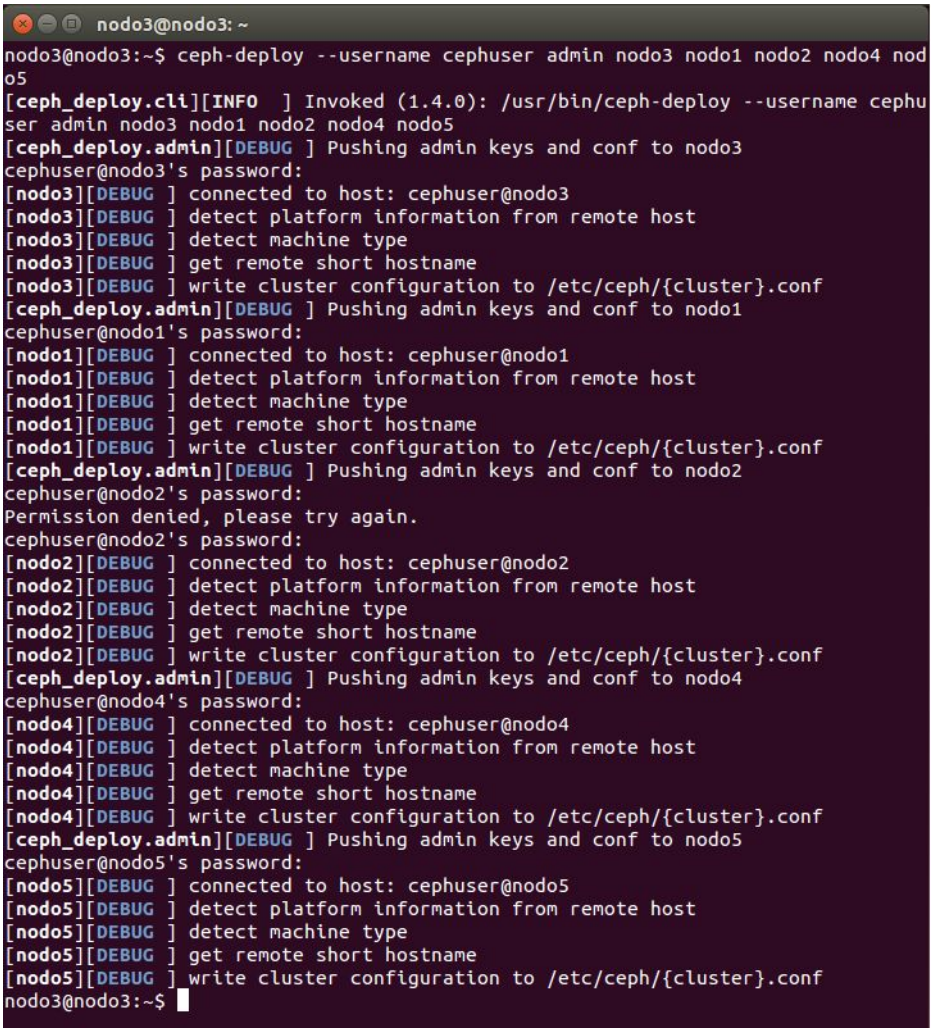


Figura 4.8: Salida del comando de creación del nodo administrador en Ceph. Adicionalmente, recoge la información relativa a los nodos que se se le indica en la llamada y la escribe en el fichero «/etc/ceph/{cluster}.conf»

- `sudo chmod +r /etc/ceph/ceph.client.admin.keyring`
- `ceph-deploy --username cephuser mds create nodo5`

5. En el cliente:

- `sudo aptitude install ceph-fuse`
- `sudo ceph-fuse -k ./ceph.client.admin.keyring -m nodo4:6789 /newhd -o nonempty`

4.2.3. Despliegue de GlusterFS

Para poder instalar GlusterFS, al menos se necesitan dos nodos. Para cada uno de los nodos servidores realizaremos los siguientes pasos:

1. Primero se crean los *bricks.*

```
apt-get install xfs xfsprogs
mkfs.xfs -i size=512 /dev/sdb1
mkdir -p /data/brick1
```

2. Insertamos al final del fichero la siguiente línea.

```
// /dev/sdb1 /data/brick1 xfs defaults 1 2
vi /etc/fstab
```

3. Se monta el *brick*.

```
mount -a && mount
```

4. Se instala GlusterFS en cada nodo.

```
apt-get install yum
yum install glusterfs-server
```

5. Se ejecuta el demonio glusterfs.

```
sudo service glusterfs-server start
```

6. Se configura el *trusted pool* de cada nodo.

   Esta instrucción se realiza por cada uno de los otros nodos usando el nombre de la máquina.

```
//En el nodo1 (ubuntuserver2 es el hostname del otro servidor)
sudo gluster peer probe ubuntuserver2

//En el nodo2
sudo gluster peer probe ubuntuserver1
```

7. Se crea el volumen de Gluster.

   En las dos máquinas se crea un directorio:

   ```
   sudo mkdir /data/brick1/gv0
   ```

   Y en una de las máquinas arrancamos el volumen.

   ```
   sudo gluster volume create gv0 replica 2 server1:/data/brick1/gv0 server2:/data/brick1/gv0
   sudo gluster volume start gv0
   sudo gluster volume info
   ```

Para poder probar Gluster en una máquina cliente es necesario realizar los siguientes pasos:

1. Comprobar que el módulo FUSE está instalado. Nota: Gluster trabaja en el espacio de usuario (*userspace*) a través de FUSE.

   ```
   sudo modprobe fuse
   //Y para comprobar que este está instalado
   sudo fuse init
   (API version 7.23)
   ```

2. Instalar OpenSSH.

   ```
   sudo apt-get install openssh-server vim wget
   ```

3. Instalar GlusterFS de la web http://download.gluster.com/pub/gluster/glusterfs

   ```
   sudo dpkg -i GlusterFS_DEB_file
   ```

4. Abrir puertos 24007 y 24008. Además hay que abrir puertos empezando por el 49152 por cada brick que exista.

   ```
   sudo iptables -A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 24007:24008 -j ACCEPT
   sudo iptables -A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 49152:49156 -j ACCEPT
   ```

5. Montar los volúmenes.
   Se puede hacer de forma manual o automática, es decir, cada vez que se encienda el ordenador.

Manual:

```
sudo mount -t glusterfs ubuntuserver:/gv0 /mnt/glusterfs
```

Automática:

Hay que editar el fichero /etc/fstab

```
ubuntuserver:/gv0 /mnt/glusterfs glusterfs defaults,_netdev 0 0
```

Referencia rápida

A continuación se resume la instalación al mínimo número de comandos imprescindibles para desplegar GlusterFS:

1. En todos los nodos de almacenaje:

- `sudo apt-get update`
- `sudo apt-get install python-software-properties`
- `sudo apt-get install glusterfs-server`
- `sudo mkdir {ruta del brick}`
- `sudo gluster peer probe {nombre del nodo}` (Una instrucción por cada uno de los otros nodos)

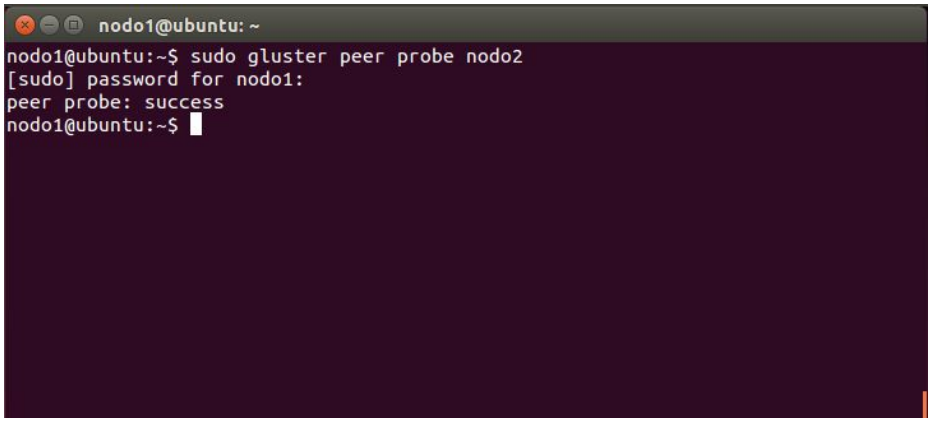


Figura 4.9: Salida del comando de sondeo de pares en GlusterFS. Con este comando localiza al nodo indicado y lo añade a la lista de pares con el resto de nodos de almacenamiento.

2. En uno de los nodos de almacenaje:

- `sudo gluster volume create volumen1 replica 2 transport tcp {nombre del nodo}:{ruta del brick}` (tantas parejas nombre-ruta como bricks haya) `force`
- `sudo gluster volume start volumen1`

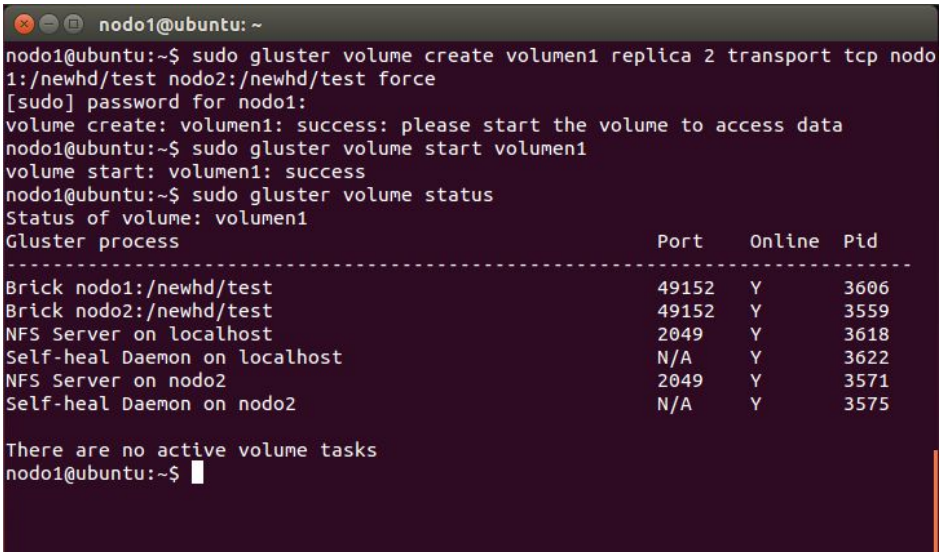


Figura 4.10: Secuencia de creación, arranque y comprobación del estado de un volumen en GlusterFS. Se puede apreciar cómo los dos *bricks* de Nodo1 y Nodo2 aparecen marcados como «Online», así como el resto de demonios que componen el *cluster*.

3. En el cliente:

- `sudo apt-get install glusterfs-client`
- `sudo mkdir {ruta del punto de montaje}`
- `sudo mount -t glusterfs {nombre del nodo donde se ha ejecutado gluster volume create}:/volumen1 {ruta del punto de montaje}`

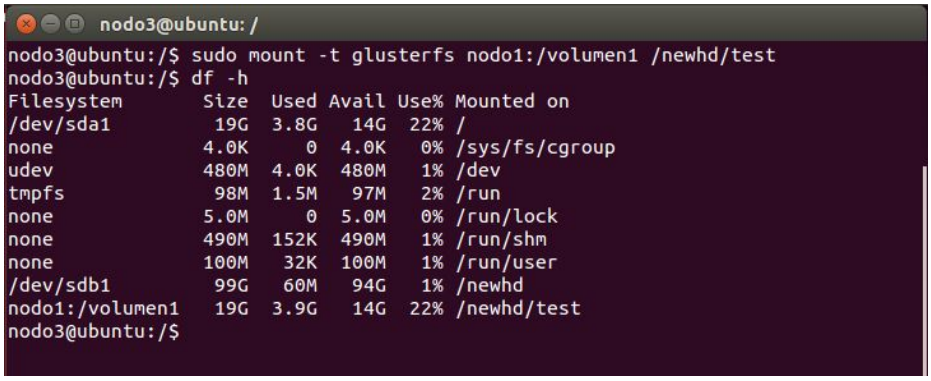


Figura 4.11: Montaje de GlusterFS y comprobación del volumen montado.

4.2.4. Despliegue de LizardFS

Para desplegar LizardFS en el entorno de pruebas es necesario seguir estos pasos:

1. Preinstalación de LizardFS. En todas las máquinas que van a formar parte del cluster:

```
- wget -O - http://packages.lizardfs.com/lizardfs.key > lizardfs.key
- sudo apt-key add lizardfs.key
- sudo rm lizardfs.key

- sudo chmod 777 /etc/apt/sources.list.d/
- sudo echo "deb http://packages.lizardfs.com/ubuntu/trusty trusty main" >
  /etc/apt/sources.list.d/lizardfs.list
- sudo echo "deb-src http://packages.lizardfs.com/ubuntu/trusty trusty main" >>
  /etc/apt/sources.list.d/lizardfs.list

- sudo apt-get update
```

2. Preparación del *Master Server.* En el Nodo4:

```
- sudo apt-get install lizardfs-master

- sudo cp /var/lib/mfs/metadata.mfs.empty /var/lib/mfs/metadata.mfs
- sudo cp /etc/mfs/mfsexports.cfg.dist /etc/mfs/mfsexports.cfg
- sudo cp /etc/mfs/mfsgoals.cfg.dist /etc/mfs/mfsgoals.cfg
- sudo cp /etc/mfs/mfsmaster.cfg.dist /etc/mfs/mfsmaster.cfg
- sudo cp /etc/mfs/mfstopology.cfg.dist /etc/mfs/mfstopology.cfg
- sudo rm /etc/mfs/*.dist

- sudo vim /etc/mfs/mfsexports.cfg  //Añadir la línea «192.168.1.*            /
  rw,alldirs,maproot=0» Debajo de la primera línea no comentada.
```

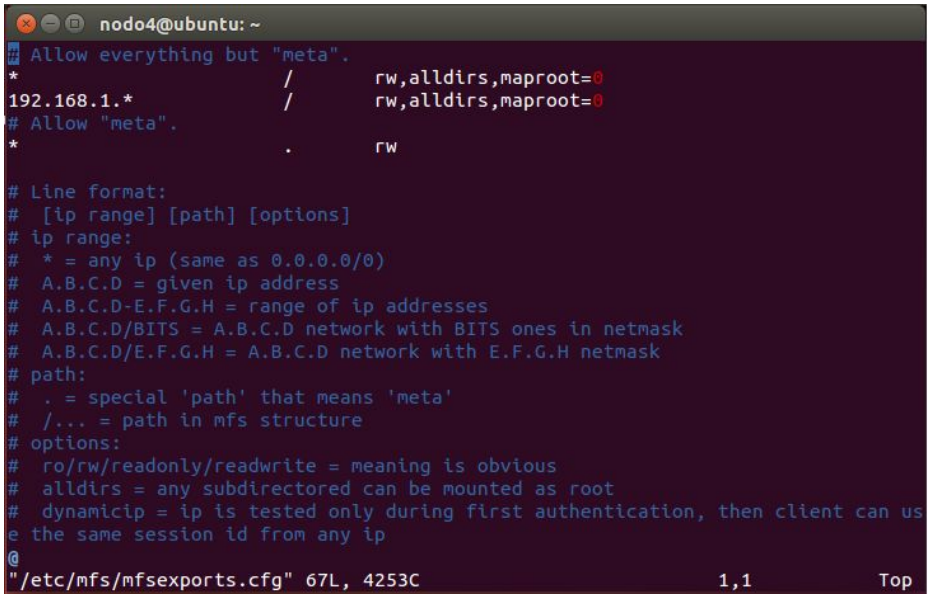


Figura 4.12: Contenido del fichero «/etc/mfs/mfsexports.cfg». El fichero debe quedar con este contenido. Para ello es necesario añadir la segunda línea no comentada. Desde aquí se controla la lista de acceso de los clientes.

- `sudo vim /etc/default/lizardfs-master //Modificar el valor «LIZARDFSMASTER_ENABLE» y ponerlo a «true».`
- `sudo service lizardfs-master start`

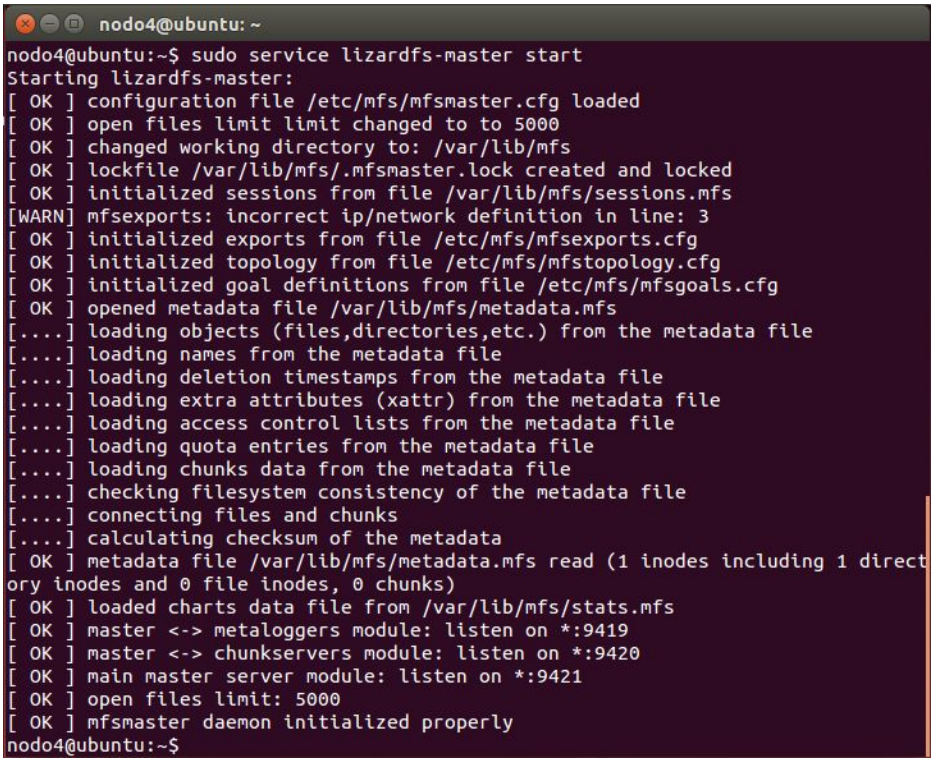


Figura 4.13: Salida del comando de arranque del demonio de LizardFS.

3. Instalación y configuración del Chunk Server 1 y 2. En el Nodo1 y 2:

- `sudo apt-get install lizardfs-chunkserver`
- `sudo cp /etc/mfs/mfschunkserver.cfg.dist /etc/mfs/mfschunkserver.cfg`
- `sudo cp /etc/mfs/mfshdd.cfg.dist /etc/mfs/mfshdd.cfg`
- `sudo rm /etc/mfs/*.dist`

- `sudo vim /etc/mfs/mfschunkserver.cfg //Activar la línea y cambiar el valor de «MASTER_HOST» a «nodo4».`

- `sudo vim /etc/mfs/mfshdd.cfg //Añadir una línea con el texto «/newhd».`
- `sudo mount /dev/sdb1 /newhd/ -t ext4`
- `sudo chown -R mfs:mfs /newhd`

- `sudo vim /etc/default/lizardfs-chunkserver //Modificar el valor de «LIZARDFSCHUNKSERVER_ENABLE» y ponerlo a «true».`

- `sudo service lizardfs-chunkserver start`

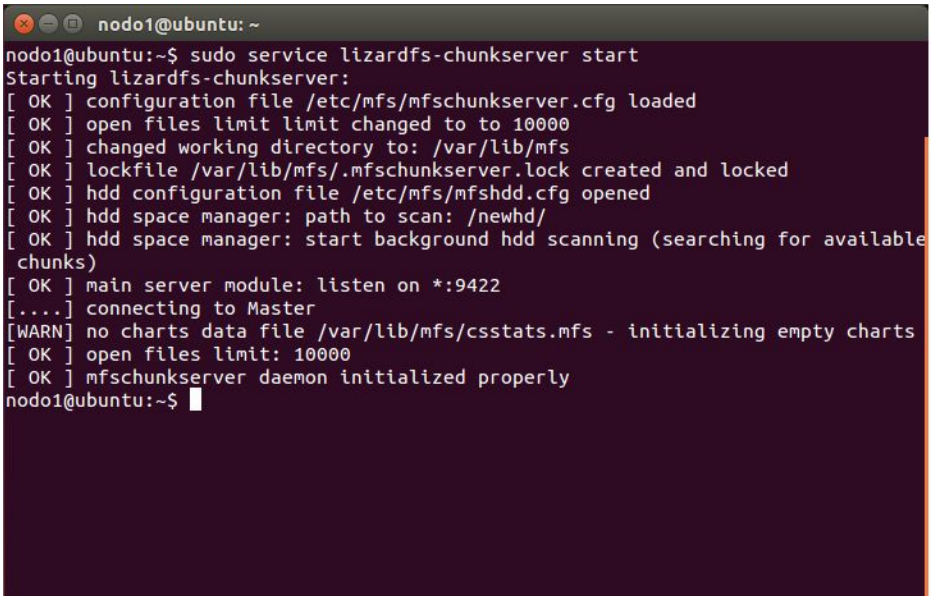


Figura 4.14: Salida del comando de arranque de los *chunkservers* de LizardFS.

4. Instalación y configuración del Metalogger. En el Nodo1:

- `sudo apt-get install lizardfs-metalogger`
- `sudo cp /etc/mfs/mfsmetalogger.cfg.dist /etc/mfs/mfsmetalogger.cfg`
- `sudo rm /etc/mfs/*.dist`

- `sudo vim /etc/mfs/mfsmetalogger.cfg //Activar la línea y cambiar el valor de «MASTER_HOST» a «nodo4».`
- `sudo vim /etc/default/lizardfs-chunkserver //Modificar el valor de «LIZARDFSMETALOGGER_ENABLE» y ponerlo a «true».`

- `sudo service lizardfs-metalogger start`

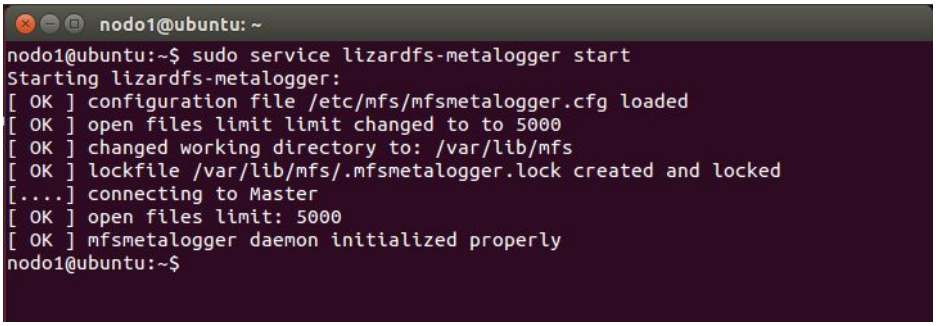


Figura 4.15: Salida del comando de arranque del *metalogger* de LizardFS.

5. Instalación del *CGI Server*. En el Nodo2:

- `sudo apt-get install lizardfs-cgiserv`
- `sudo vim /etc/default/lizardfs-cgiserv //Añadir línea con el valor «LIZARDFSCGISERV_ENABLE=true».`
- `sudo service lizardfs-cgiserv start`

6. Instalación y configuración del Shadow Server. En el Nodo5:

- `sudo apt-get install lizardfs-master`
- `sudo cp /var/lib/mfs/metadata.mfs.empty /var/lib/mfs/metadata.mfs`
- `sudo cp /etc/mfs/mfsexports.cfg.dist /etc/mfs/mfsexports.cfg`
- `sudo cp /etc/mfs/mfsgoals.cfg.dist /etc/mfs/mfsgoals.cfg`
- `sudo cp /etc/mfs/mfsmaster.cfg.dist /etc/mfs/mfsmaster.cfg`
- `sudo cp /etc/mfs/mfstopology.cfg.dist /etc/mfs/mfstopology.cfg`
- `sudo rm /etc/mfs/*.dist`
- `sudo vim /etc/mfs/mfsexports.cfg //Añadir la línea «192.168.1.* / rw,alldirs,maproot=0» Debajo de la primera línea no comentada.`
- `sudo vim /etc/default/lizardfs-master //Modificar el valor «LIZARDFSMASTER_ENABLE» y ponerlo a «true».`
- `sudo vim /etc/mfs/mfsmaster.cfg //Activar la línea «PERSONALITY = master» e igualarlo a «shadow».`
- `sudo service lizardfs-master start`

7. Instalación y configuración del cliente. En el Nodo3:

- `sudo apt-get install lizardfs-client`
- `sudo mount /dev/sdb1 /newhd/ -t ext4`
- `sudo mfsmount /newhd`
- `sudo mfssetgoal -r 2 /newhd`

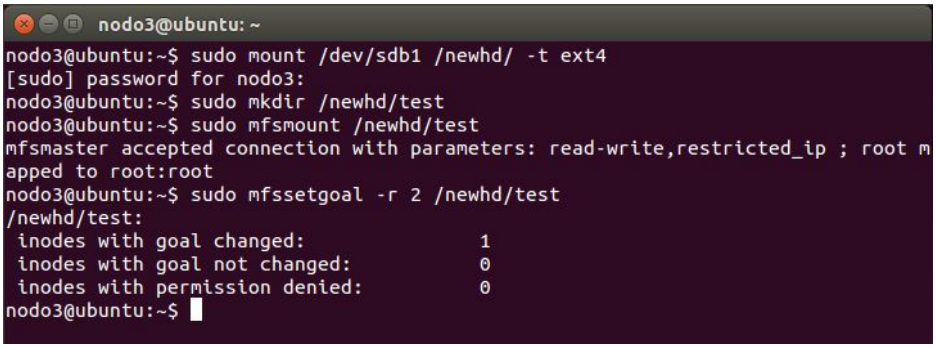


Figura 4.16: Secuencia de comandos de preparación del cliente de LizardFS. La secuencia incluye el montaje del disco duro adicional de la máquina, la creación del directorio donde se va a montar LizardFS, el montaje propiamente dicho y la asignación de valor al parámetro «goal» para la carpeta «/test», que es el responsable de controlar la replicación.

8. Opcional. Monitorización del sistema. En el Nodo2:
- Abrir el navegador y escribir la dirección: http://localhost:9425/mfs.cgi?masterhost=mfsmaster

Esta monitorización es uno de los puntos fuertes y el aspecto más diferenciador de LizardFS. El resultado del paso 8 se refleja en las siguientes capturas de pantalla.

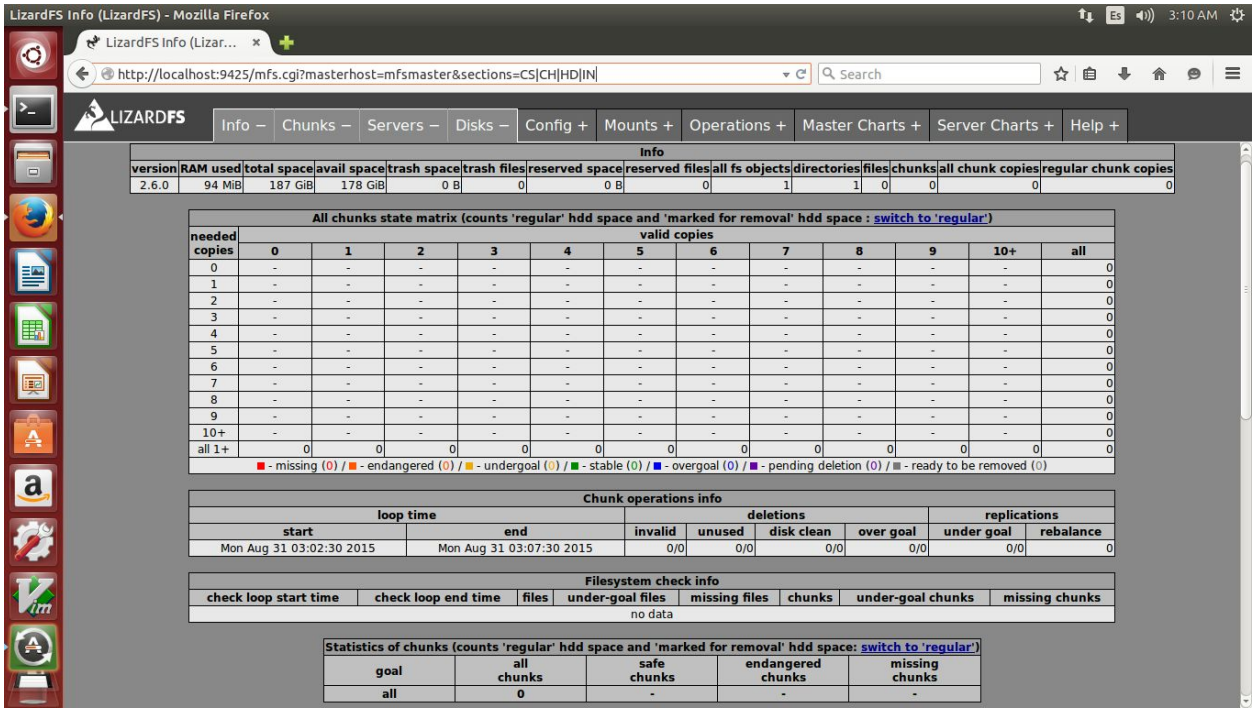


Figura 4.17: Interfaz web de administración de LizardFS, pestaña «Info».

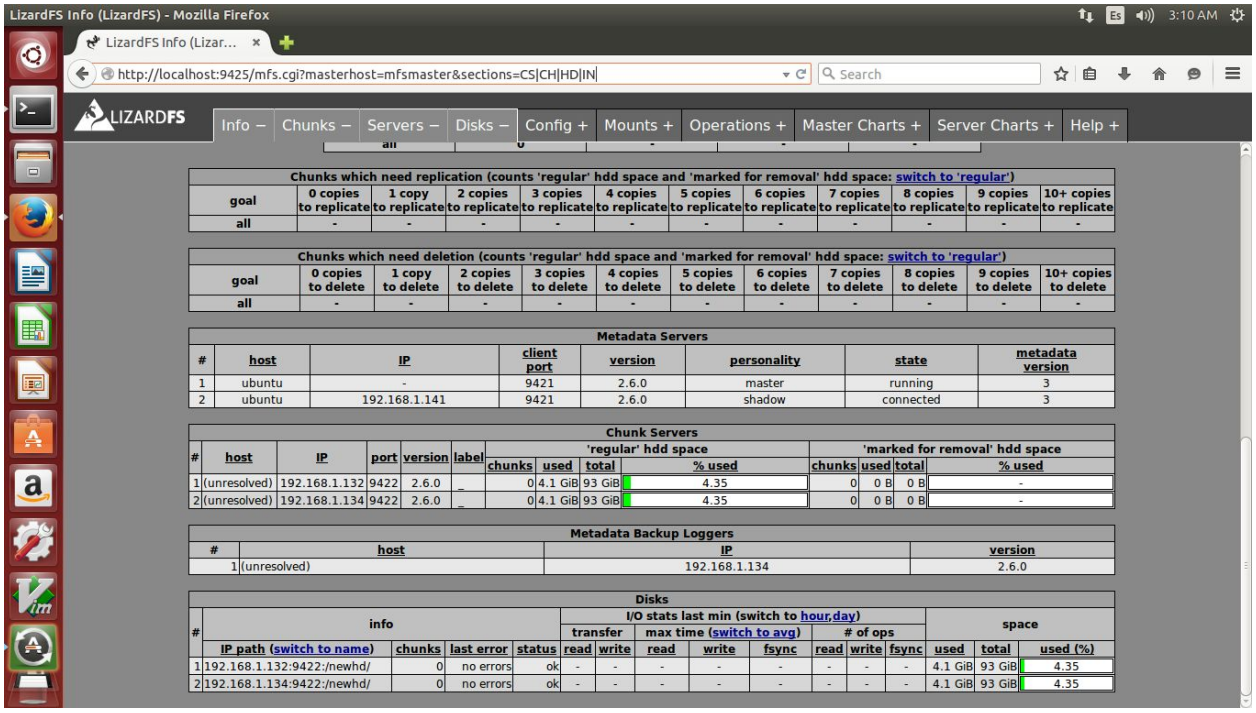


Figura 4.18: Interfaz web de administración de LizardFS, pestañas «Chunk», «Servers» y «Disks».

4.2.5. Despliegue de los *benchmarks*

A continuación se explica cómo se instala y despliega cada uno de los *benchmarks* que se van a utilizar. La mayoría son simples llamadas a apt-get, ya que se encuentran en los repositorios de Ubuntu. Todas estas herramientas deben instalarse en la máquina que haga de cliente en cada uno de los entornos de pruebas.

IOzone 3:

- `sudo apt-get install iozone3`

FIO 2.1.3:

- `sudo apt-get install fio`

Filebench 1.4.9.1:

- Descargar la última versión de http://sourceforge.net/projects/filebench/. En el caso concreto de este estudio se empleó la versión 1.4.9.1, que era la última cuando se escribió.
- Descomprimir el contenido del fichero en el escritorio.
- `cd /home/nodo3/Desktop/filebench-1.4.9.1/`
- `./configure`
- `make`
- `sudo make install`

Postmark v1.51:

- `sudo apt-get install postmark`

Fdtree 1.0.2:

- Descargar Fdtree de https://computing.llnl.gov/?set=code&page=sio_downloads
- Descomprimir el fichero descargado.
- Editar el fichero Fdtree.bash y añadir «#!/bin/bash» al principio del mismo.

4.3. Pruebas y características a evaluar

El objetivo de las pruebas es obtener unas cifras que clarifiquen cómo se comporta cada solución y a la vez sirvan como dato objetivo para comparar unas con otras. Como ya se introdujo en el capítulo 2.2., se dispone de una serie de *benchmarks* para obtener esos datos. Cada *benchmark* funciona de una forma y está orientado a medir un aspecto concreto. Lo que tienen en común es que todos van a dar como resultado la velocidad con la que se lleva a cabo la tarea que cada uno de ellos propone. Y dichas tareas están basadas en las mismas operaciones básicas que se pueden realizar sobre un sistema de ficheros.

Todas las pruebas que se presentan a continuación han sido diseñadas conforme a las diferentes necesidades expresadas en los perfiles de almacenamiento, en el capítulo 3.1.3.3.

Para detectar datos atípicos y evitar que contaminen el resultado del estudio, todas las pruebas se van a realizar entre tres y diez veces. Todos los valores presentados serán las medias aritméticas de los resultados obtenidos.

Por otro lado, los requisitos de un grupo de investigación, fueron descritos en el capítulo 3.1. Teniendo esto en cuenta, se ha diseñado el siguiente plan de pruebas de rendimiento:

- Simulación de perfil de almacenamiento de ficheros (RS-07):
  - *Benchmarks* utilizados: IOzone, FIO, Filebench.
  - Tamaño de fichero: 10 KB - 100 MB.
  - Relación lecturas/escrituras: mismo número de lecturas que de escrituras.
  - Objetivos de rendimiento: 200 KB/s para cada proceso cuando hay 4 procesos, 80 KB/s para 50 procesos.
  - Características analizadas: ancho de banda medio, mínimo y máximo en todo tipo de operaciones, cantidad de MB leídos y escritos, número total de operaciones y operaciones por segundo, latencia media total y de finalización de la operación.

- Simulación de perfil de almacenamiento de servidor web (RS-08):
  - *Benchmarks* utilizados: IOzone, FIO, Filebench.
  - Tamaño de fichero: 1536 KB - 2048 KB.
  - Relación lecturas/escrituras: solo lecturas.
  - Objetivos de rendimiento: 150 KB/s para cada proceso cuando hay 50 procesos, 70 KB/s cuando hay 80 procesos.
  - Características analizadas: ancho de banda medio, mínimo y máximo en operaciones de lectura, cantidad de MB leídos y escritos, número total de operaciones y operaciones por segundo, latencia media total y de finalización de la operación.

- Simulación de perfil de almacenamiento de repositorio de código (RS-09):
  - *Benchmarks* utilizados: IOzone, FIO, Filebench.
  - Tamaño de fichero: 3 - 7 KB. A efectos prácticos da igual 5 que 7 ya que el tamaño de bloque son 4 KB.
  - Relación lecturas/escrituras: 75% lecturas y 25% de escrituras en Filebench.
  - Objetivos de rendimiento: 1 MB/s para las lecturas y 400 KB/s para las escrituras.
  - Características analizadas: ancho de banda medio, mínimo y máximo en operaciones de lectura, cantidad de MB leídos y escritos, número total de operaciones y operaciones por segundo, latencia media total y de finalización de la operación.

- Simulación de perfil de almacenamiento de servidor de correo (RS-10):
  - *Benchmarks* utilizados: FIO (IOzone, Filebench y Postmark aportan información adicional no eliminatoria).
  - Tamaño de fichero: 10 - 75 KB.
  - Relación lecturas/escrituras: todo lecturas.
  - Objetivos de rendimiento: mínimo 25 KB/s para cada proceso con 50 procesos.
  - Características analizadas: ancho de banda medio, mínimo y máximo en operaciones de lectura, cantidad de MB leídos y escritos, número total de operaciones y operaciones por segundo, latencia media total y de finalización de la operación, duración de las transacciones, número total de operaciones y operaciones por segundo clasificadas por tipo.

- Simulación de perfil de almacenamiento de discos duros virtuales (RS-11):
  - *Benchmarks* utilizados: IOzone, FIO, Filebench.
  - Tamaño de fichero: 4 GB.
  - Relación lecturas/escrituras: depende de la prueba: 50 % - 50 % o 75 % - 25 %.
  - Objetivos de rendimiento: para los dos procesos secuenciales, 5 MB/s por proceso de lectura y de 2,5 MB/s por cada proceso de escritura. Para la prueba de los dos procesos aleatorios y para la prueba de los cuatro procesos aleatorios 125 KB/s por cada proceso.
  - Características analizadas: ancho de banda medio, mínimo y máximo en operaciones de lectura, cantidad de MB leídos y escritos, número total de operaciones y operaciones por segundo, latencia media total y de finalización de la operación.

4.4. Resultados obtenidos

4.4.1. Llamadas y salidas de los *benchmarks* utilizados

En este capítulo se explica en qué consisten los parámetros seleccionados y empleados en las llamadas que se han realizado para obtener los datos de rendimiento. De la misma manera se incluye una salida de ejemplo de cada *benchmark*. Estos informes son los que proporcionan los

datos que luego son filtrados, resumidos y presentados en tablas para facilitar su comparación entre tecnologías.

4.4.1.1. IOzone

Los parámetros utilizados en la llamada a IOzone significan:

- -R: formatear la salida para Excel.
- -a: modo automático. Analiza todas las operaciones posibles con un tamaño de bloque de 4 KB sobre ficheros con un tamaño de 64 KB. Luego duplica el tamaño del fichero y vuelve a llevar a cabo las mediciones. Este proceso se repite hasta que el fichero alcanza los 524288 KB. Posteriormente se duplica también el tamaño del bloque y se inicia de nuevo todo el proceso, que se va repitiendo de la misma manera hasta que el tamaño de bloque llega a los 16384 KB.
- -I: utilizar operaciones directas de entrada y salida (no utilizar *buffers*).
- -b: nombre del fichero donde se quiere guardar la salida.
- -l: mínimo número de procesos.
- -u: máximo número de procesos.
- -r: tamaño del registro que se va a tratar. Medido por defecto en *kilobytes*.
- -s: tamaño del fichero para las pruebas. Medido por defecto en *kilobytes*.

Un ejemplo de salida de IOzone es:

```
Children see throughput for  2 initial writers  =  142414.30 KB/sec
Parent sees throughput for  2 initial writers   =      38.82 KB/sec
Min throughput per process                      =       0.00 KB/sec
Max throughput per process                      =  142414.30 KB/sec
Avg throughput per process                      =   71207.15 KB/sec
Min xfer                                        =       0.00 KB
```

4.4.1.2. FIO

Los parámetros utilizados en la llamada a FIO significan:

- --output: ruta y nombre del fichero de salida.
- --ioengine: define la manera de tratar la tarea. Sirve para seleccionar el motor de entrada salida. Libaio es la librería asíncrona nativa de Linux.
- --direct: si se fija el valor «true» se usa acceso directo al disco duro, sin *buffers*.
- --iodepth: numéro de hilos concurrentes. Uno por defecto.
- --name: nombre de la tarea.
- --rw: tipo de patrón de entrada/salida. Puede tomar los valores lectura, escritura, lectura aleatoria, escritura aleatoria, lecturas y escrituras secuenciales mezcladas y lecturas y escrituras aleatorias mezcladas.
- --filesize: rango de tamaños de los ficheros.
- --numjobs: número de hilos que realizan el mismo trabajo. Uno por defecto.
- --runtime: duración del proceso en segundos.
- --name: la segunda aparición de este parámetro en la llamada sirve para nombrar cada uno de los trabajos (*jobs*) que hace la tarea.
- --time_based: al acabar la tarea, la reinicia si no ha transcurrido todo el tiempo que marca --runtime.
- --norandommap: hace que los accesos aleatorios sean realmente aleatorios sin tener en cuenta las elecciones aleatorias previas para descartarlas.
- --file_service_type: define el criterio para seleccionar el próximo fichero con el que se va a trabajar. El valor «roundrobin» sirve para que se utilice el algoritmo del mismo nombre a la hora de cambiar de fichero.
- -size: cantidad total de *bytes* que se van a mover en operaciones de lectura y escritura antes de finalizar la tarea.
- --bs: tamaño de bloque. Por defecto son 4 KB.

Un ejemplo de salida de FIO es:

```
writebw: (g=0): rw=read, bs=8K-8K/8K-8K, ioengine=sync, iodepth=1
fio 1.59
Starting 1 process
Jobs: 1 (f=1): [R] [100.0% done] [872.6M/0K /s] [109K/0  iops] [eta 00m:00s]
writebw: (groupid=0, jobs=1): err= 0: pid=3427
  read : io=3147.4MB, bw=772912KB/s, iops=96614 , runt= 58436msec
    clat (usec): min=1 , max=40964 , avg= 3.72, stdev=54.99
     lat (usec): min=2 , max=40964 , avg= 4.00, stdev=55.00
  cpu          : usr=25.48%, sys=65.50%, ctx=38064, majf=0, minf=26
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=5645737/0/0, short=0/0/0
     lat (usec): 2=0.05%, 4=98.67%, 10=0.36%, 20=0.17%, 50=0.09%
     lat (usec): 100=0.01%, 250=0.63%, 500=0.02%, 750=0.01%, 1000=0.01%
     lat (msec): 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=0.01%

Run status group 0 (all jobs):
   READ: io=44107MB, aggrb=772912KB/s, minb=791462KB/s, maxb=791462KB/s, mint=58436msec, maxt=58436msec

Disk stats (read/write):
  sda: ios=36253/11, merge=0/6, ticks=8624/40, in_queue=8656, util=14.42%
```

4.4.1.3. Filebench

La utilización de Filebench es sencilla. Se llama a Filebench y una vez dentro se emplean los siguientes comandos:

- run t: ejecuta Filebench durante t segundos.
- load: indica a filebench qué comportamiento se desea que adopte.
- set $dir: fija el directorio de trabajo de Filebench.

Un ejemplo de salida de Filebench es:

```
statfile1            374ops        6ops/s   0.0mb/s   1515.6ms/op    37513us/op-cpu [0ms - 10655ms]
deletefile1          382ops        6ops/s   0.0mb/s   1839.9ms/op    43665us/op-cpu [1ms - 10943ms]
closefile3           398ops        6ops/s   0.0mb/s      1.6ms/op      251us/op-cpu [0ms - 165ms]
readfile1            398ops        6ops/s   0.8mb/s    224.8ms/op     6683us/op-cpu [0ms - 5425ms]
openfile2            399ops        6ops/s   0.0mb/s   1513.0ms/op    36366us/op-cpu [0ms - 10965ms]
closefile2           411ops        7ops/s   0.0mb/s    265.5ms/op     6253us/op-cpu [1ms - 3041ms]
appendfilerand1      413ops        7ops/s   0.0mb/s     48.5ms/op     1889us/op-cpu [0ms - 1418ms]
openfile1            413ops        7ops/s   0.0mb/s   1538.6ms/op    37748us/op-cpu [0ms - 10938ms]
closefile1           424ops        7ops/s   0.0mb/s    129.2ms/op     2689us/op-cpu [1ms - 2606ms]
wrtfile1             424ops        7ops/s   0.8mb/s    113.4ms/op     2288us/op-cpu [0ms - 4568ms]
createfile1          424ops        7ops/s   0.0mb/s    590.2ms/op    13302us/op-cpu [1ms - 4656ms]
 3228: 500.851: IO Summary:  4460 ops, 70.703 ops/s, (6/13 r/w),   1.6mb/s,   6980us cpu/op, 2494.9ms
latency
 3228: 500.851: Shutting down processes
```

Nota: Las personalidades predefinidas de Filebench pueden usar *buffers* para las operaciones con metadatos. Se ha decidido no alterar dichos ficheros, ya que todos los sistemas analizados disfrutarán de la misma ventaja por igual.

4.4.1.4. Postmark

Postmark es muy sencillo de utilizar:

- se entra en el directorio donde se quiere ejecutar Postmark y se le llama,
- set size: establece el límite inferior y superior para el tamaño de los ficheros,
- set number: número de ficheros simultáneos,
- set transactions: establece el número de transacciones que hará Postmark.

Un ejemplo de salida de Postmark es:

```
Time:
        47 seconds total
        21 seconds of transactions (47 per second)

Files:
        1496 created (31 per second)
                Creation alone: 1000 files (55 per second)
                Mixed with transactions: 496 files (23 per second)
        483 read (23 per second)
        517 appended (24 per second)
```

```
        1496 deleted (31 per second)
                Deletion alone: 992 files (124 per second)
                Mixed with transactions: 504 files (24 per second)

Data:
        1.19 megabytes read (25.97 kilobytes per second)
        3.86 megabytes written (84.12 kilobytes per second)
```

4.4.1.5. Fdtree

Para utilizar Fdtree basta con añadir `#!/bin/bash` en el propio *script*. Los pasos siguientes son:

- Seleccionar la carpeta donde se quiere crear la estructura de directorios y ficheros e invocar a Fdtree desde allí.
- -l indica el número de niveles que va a tener la estructura.
- -d es el número de directorios que se van a crear por cada nivel.
- -f, el número de ficheros por directorio.
- -s determina el tamaño de los ficheros que se van a crear en número de bloques.

Un ejemplo de salida de Fdtree sin tratar es:

```
fdtree-1.0.2: starting at ./LEVEL0.ubuntu.4039/
        creating/deleting 1 directory levels with 50 directories at each level
        for a total of 51 directories
        with 500 files of size 40KiB per directory
        for a total of 25500 files and 1020000KiB
Tue Jul 28 13:16:02 PDT 2015
Tue Jul 28 13:16:03 PDT 2015
DIRECTORY CREATE TIME IN, OUT, TOTAL = 0, 1, 1
        Directory creates per second =  51
Tue Jul 28 13:16:03 PDT 2015
Tue Jul 28 13:31:41 PDT 2015
FILE CREATE TIME IN, OUT, TOTAL      = 1, 939, 938
        File creates per second      =  27
        KiB per second               =  1087
Tue Jul 28 13:31:41 PDT 2015
Tue Jul 28 13:39:36 PDT 2015
FILE REMOVE TIME IN, OUT, TOTAL      = 939, 1414, 475
        File removals per second     =  53
Tue Jul 28 13:39:36 PDT 2015
Tue Jul 28 13:39:37 PDT 2015
DIRECTORY REMOVE TIME IN, OUT, TOTAL = 1414, 1415, 1
        Directory removals per second =  51
```

4.4.2. Resultados de los *benchmarks*

En este capítulo se recogen los datos obtenidos en todas las pruebas realizadas en cada *benchmark*. Se ordenan por perfil de almacenamiento (capítulo 3.1.1.3), y dentro de cada perfil, por *benchmark*.

Para facilitar la lectura de las tablas, se ha resaltado el mejor resultado marcando su celda en verde. En algunos casos no hay un mejor valor claro.

De la misma manera, algunas tablas van acompañadas de gráficos para poder hacer una comparación más visual. No hay un patrón a la hora de ilustrar las tablas. Se ha hecho gráfico de las que se ha considerado que aportan más, y lo mismo ocurre con el tipo de gráfico; se ha elegido el tipo más adecuado en función de lo que se quiere mostrar.

Todas las operaciones se han hecho con un tamaño de bloque de 4 KB.

4.4.2.1. Almacenamiento de ficheros

IOzone:

La prueba seleccionada para medir el rendimiento de las soluciones tratando ficheros variados es un test automático limitado por el tamaño de fichero. Esto sirve para obtener una visión general tanto de las operaciones más habituales como de otras algo más específicas y menos frecuentes, pero que puede llevar a cabo algún software. Esta prueba no pretende ser eliminatoria. El resultado mostrado en la tabla se obtiene llamando a IOzone con los siguientes parámetros:

```
sudo iozone -RaIb autotest3.wks -g 102400 -n 10
```

	GlusterFS	Ceph	LizardFS
Writer	13680 - 18674	22922 - 38360	16642 - 34290
Re-writer	12323 - 17561	21105 - 41001	14081 - 32444
Reader	3852 - 6499	7596 - 40965	5073 - 10804
Re-reader	2327 - 6196	8308 - 42346	3184 - 10545
Random read	875 - 26289	3642 - 6036	2277 - 9763
Random write	3214 - 11683	24694 - 35826	10890 - 67824
Backward read	523 - 5157	459 - 4360	2197 - 9625
Record rewrite	4305 - 17468	22148 - 37537	17866 - 26036
Stride read	1848 - 6537	3651 - 23937	1805 - 7073
Fwrite	11474 - 32806	19825 - 35402	18805 - 53362
Re-fwrite	13140 - 40194	9533 - 35903	13188 - 57971
Fread	13585 - 281863	7912 - 181380	732538 - 4450021
Re-fread	15156 - 923513	9101 - 214738	1996610 - 5815182

Tabla 4.1: Prueba de IOzone para el perfil de almacenamiento de ficheros. Se muestran los anchos de banda mínimos y máximos para cada tipo de operación realizada. Todas las velocidades están medidas en KB/s.

Esta prueba automática comprueba un gran número de operaciones, explicadas en el apartado 2.2.1. En general se puede advertir que Ceph es el que presenta mejores cifras tanto mínimas como máximas en las operaciones más habituales, como lecturas y escrituras. LizardFS destaca mucho cuando se trata de operar con buffers, como se puede ver en las últimas operaciones.

En el caso de las operaciones de lecturas aleatorias GlusterFS sufre una diferencia enorme entre su mínimo y su máximo. Para averiguar qué tecnología supera a las demás, es necesario examinar una vez más la salida de todas ellas. Si se calculan las medias de velocidad que

presentan GlusterFS, Ceph y LizardFS, se obtienen los valores 17219, 4434 y 5960 respectivamente, siempre medido en KB/s. La cifra de GlusterFS demuestra que no solo es la mejor cuando se trata de lecturas aleatorias, sino que el valor de 875 es un dato atípico que se manifiesta solo en ficheros extremadamente pequeños.

FIO:

Para esta prueba se ha usado un rango de ficheros de 10 KB a 100 MB, la duración de la prueba ha sido de un minuto y se han evaluado lecturas y escrituras con dos procesos por operación. La llamada empleada es:

```
sudo fio --output=/home/nodo3/Desktop/FIO4.txt --ioengine=libaio --iodepth=1 --name=global --rw=rw
--filesize=10K-100M --numjobs=2 --runtime=60 --name=job1 --direct=1 --time_based
--file_service_type=roundrobin
```

	GlusterFS		**Ceph**		**LizardFS**	
	Lectura	Escritura	Lectura	Escritura	Lectura	Escritura
Número de MB	49,10	48,94	404,39	405,64	112,73	112,35
Ancho de banda total	837	835	6447	6467	1923	1917
Ancho de banda mínimo	417	416	354	354	953	949
Ancho de banda máximo	420	418	6520	6539	970	968
Latencia media de fin (µs)	2,22	2,04	1,27	1,24	1,55	1,31
Latencia media total (µs)	9208,46	331,71	5392,92	524,58	3930,18	223,31

Tabla 4.2: Prueba de FIO para el perfil de almacenamiento de ficheros. Los anchos de banda están medidos en *kilobytes* por segundo (KB/s).

Ceph es la tecnología más eficiente en esta prueba. Obtiene los mejores registros con facilidad, casi cuadruplicando a LizardFS en la cantidad de *megabytes* movidos. Es llamativo que los procesos que rodean a las transferencias de datos provoquen una mayor latencia total en Ceph.

Filebench:

Para este test se ha empleado la personalidad de Filebench que simula ser un servidor de ficheros. Tras cargar la personalidad, se puede observar cuál es la configuración por defecto que utiliza, y que resulta interesante no alterar. La ejecución se programa para durar diez minutos.

```
load fileserver
 5608: 47.296: File-server Version 3.0 personality successfully loaded
 5608: 47.296: Usage: set $dir=<dir>
 5608: 47.296:        set $meanfilesize=<size>     defaults to 131072
 5608: 47.296:        set $nfiles=<value>      defaults to 10000
 5608: 47.296:        set $nthreads=<value>    defaults to 50
 5608: 47.296:        set $meanappendsize=<value>  defaults to 16384
 5608: 47.296:        set $iosize=<size>  defaults to 1048576
 5608: 47.296:        set $meandirwidth=<size> defaults to 20
```

Esto crea 10000 (1,21 GB) ficheros repartidos en una estructura de directorios con una anchura media de 20 y una profundidad media de 3,1. El sistema genera un 79 % de la estructura antes de comenzar la prueba.

	GlusterFS	Ceph	LizardFS
Núm. de operaciones	47585	109958	141874
Operaciones por seg.	79,23	182,60	236,05
Ancho de banda (MB/s)	1,8	4,3	5,5
Latencia (ms)	2288,7	1000,8	774,9

Tabla 4.3: Prueba de Filebench para el perfil de almacenamiento de ficheros. La casilla roja muestra que se incumple un requisito.

Existen otros datos interesantes que se observan en la salida detallada del *benchmark*. Por ejemplo, Ceph invierte una media de 1395 ms. por cada operación de leer y anexar datos a los ficheros y tiene una media de tan solo 16 ms. en las operaciones de abrir y borrar. Sin embargo, Gluster presenta el comportamiento contrario, con lecturas rápidas y anexos rápidos (31 ms.) y una media lenta de 1579 ms. en aperturas y borrados. Por su parte, Lizard no presenta diferencias tan grandes, pero la operación que más tarda en completar es la de «fsync».

En esta prueba, los mejores resultados los aporta LizardFS en todos los aspectos, probablemente por la influencia de algún buffer en las operaciones implementadas en la personalidad de Filebench.

Según el RS-07, para esta prueba se requiere un ancho de banda total de 80 KB/s por proceso. Al haber 50 procesos, son 4000 KB/s o 3,90 MB/s. GlusterFS se queda muy alejado de dicha cifra y por lo tanto no cumple el requisito mínimo.

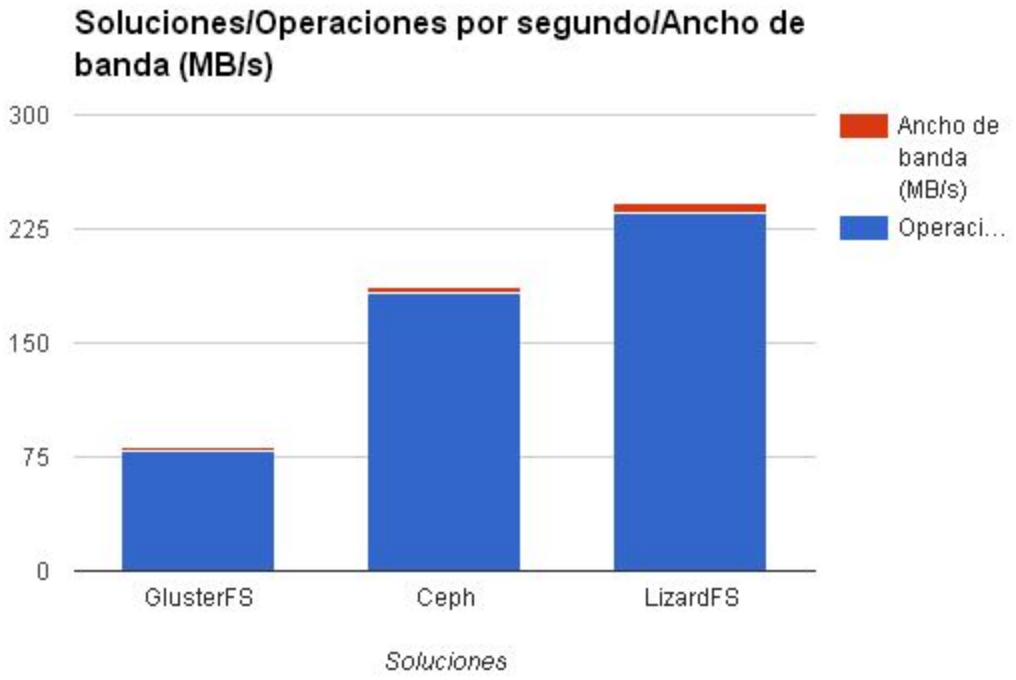


Figura 4.19: Gráfica ilustrativa de la tabla 4.3. Se muestran las operaciones por segundo que consigue cada tecnología y el ancho de banda en *megabytes* por segundo. (MB/s).

Conclusión para el perfil de almacenamiento de ficheros:

Dos de los tres *benchmarks*, IOzone y FIO, muestran mejores datos para Ceph. Esas pruebas son además las que reflejan la actividad del sistema de almacenamiento sin la injerencia de ningún buffer, por lo que nos muestran el rendimiento más real.

4.4.2.2. Servidor Web

IOzone:

Esta prueba se realiza fijando un tamaño de 2 MB, con 50 procesos simultáneos y ordenando pruebas de lectura y de escritura, ya que aunque estas sean muy poco relevantes en un servidor web, son necesarias para la creación de los ficheros que utilizan las pruebas de lectura.

```
sudo iozone -s 2m -i 0 -i 1 -RIb autotest11.wks -l 2 -u 50
```

	GlusterFS	Ceph	LizardFS
Read (Mínima)	8441	63039	16340
Read (Máxima)	30998	100626	50022
Read (Media)	**9982**	**78161**	**43354**
Re-read (Mínima)	9807	72080	14281
Re-read (Máxima)	41084	91578	49687
Re-read (Media)	**11123**	**80885**	**43666**

Tabla 4.4: Prueba de IOzone para el perfil de servidor web. Todos los anchos de banda están medidos en *kilobytes* por segundo (KB/s).

Una vez más, Ceph supera a los demás y casi dobla al segundo, LizardFS.

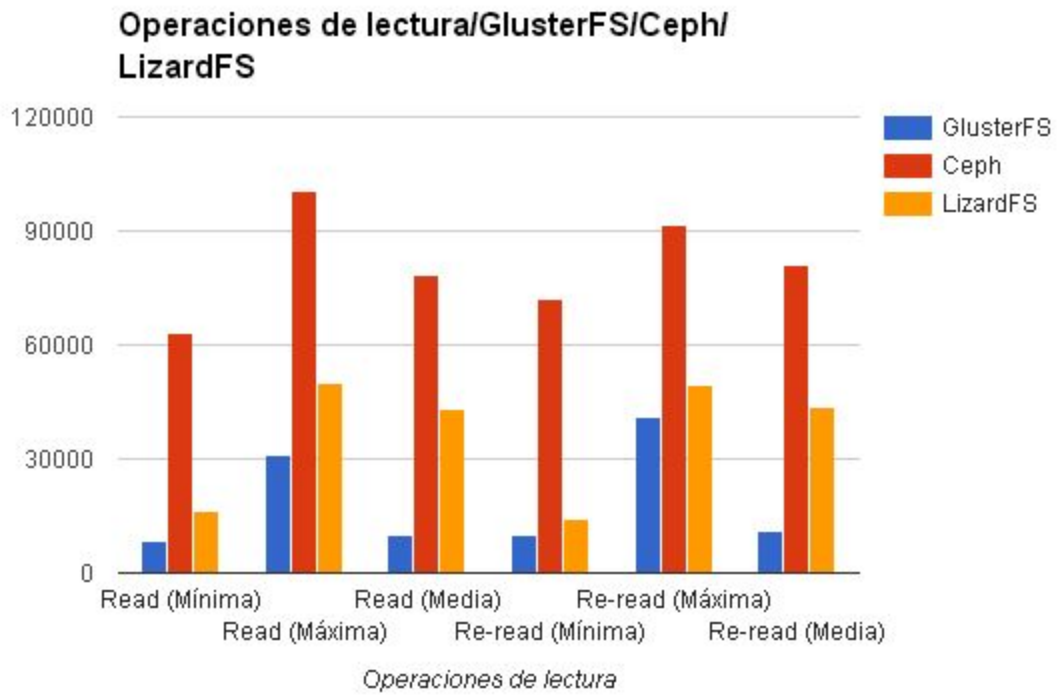


Figura 4.20: Gráfico que ilustra la tabla 4.4. Se puede ver el rendimiento mínimo, máximo y medio que ofrece cada tecnología en las dos operaciones que se analizan.

FIO:

La prueba de FIO para la simulación del servidor web consiste en hacer pruebas de solo lectura con 50 procesos durante un minuto.

```
sudo fio --output=/home/nodo3/Desktop/FIO7.txt --ioengine=libaio --iodepth=1 --name=global --rw=read
--filesize=1700K-2M --numjobs=50 --runtime=600 --name=job1 --direct=1 --time_based
--file_service_type=roundrobin
```

	GlusterFS	Ceph	LizardFS
Número de MB	5867	45519	26967
Ancho de banda total	10013	77685	46022
Ancho de banda mínimo	200	1528	891
Ancho de banda máximo	201	1573	953
Latencia media (µs)	19913,11	2546,05	4328,87

Tabla 4.5: Prueba de FIO para el perfil de servidor web. Los anchos de banda están medidos en *kilobytes* por segundo (KB/s).

Ceph destaca claramente en todas las características medidas y LizardFS mejora a su vez a GlusterFS con contundencia.

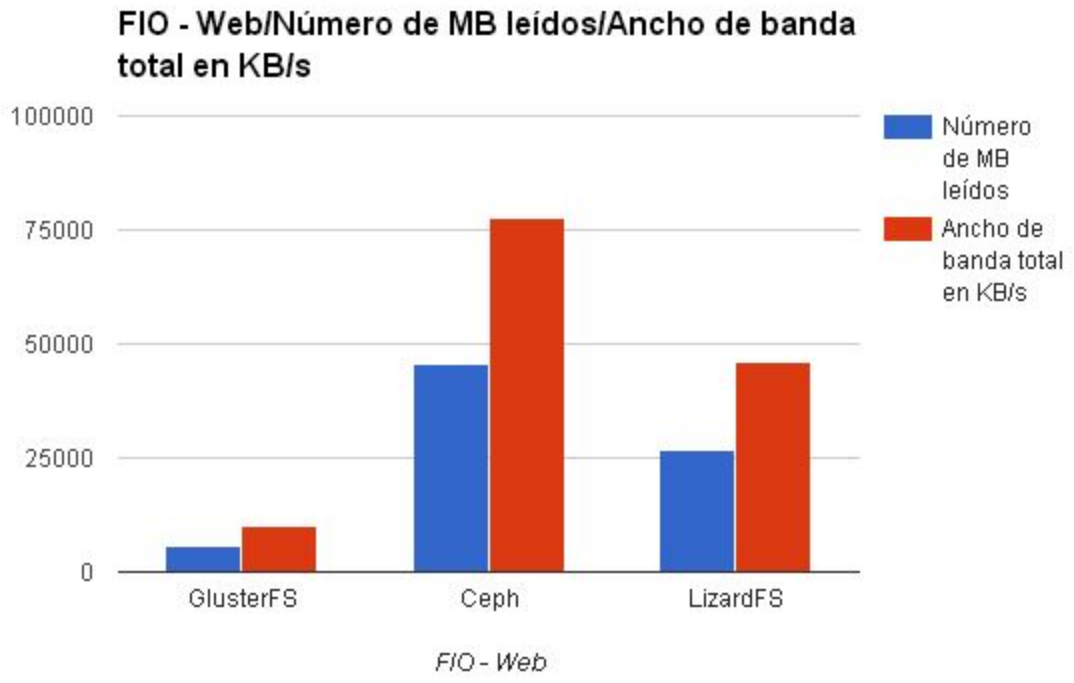


Figura 4.21: Gráfico que ilustra la tabla 4.5. Se muestra el número de *megabytes* leídos y el ancho de banda que ha alcanzado cada tecnología en esta prueba de 50 procesos de lectura.

Filebench:

La personalidad «webserver» de Filebench es una exigente prueba que muestra los siguientes datos una vez cargada:

```
filebench> load webserver
 5849: 6.161: Web-server Version 3.0 personality successfully loaded
 5849: 6.161: Usage: set $dir=<dir>
 5849: 6.161:        set $meanfilesize=<size>   defaults to 16384
 5849: 6.161:        set $nfiles=<value>    defaults to 1000
 5849: 6.162:        set $meandirwidth=<value>  defaults to 20
 5849: 6.162:        set $nthreads=<value>  defaults to 100
 5849: 6.162:        set $iosize=<size>     defaults to 1048576
```

La estructura con la que va a trabajar está compuesta por 1 000 ficheros, que suman 14,7 MB, repartidos en una estructura de directorios con una anchura media de 20 y una profundidad media de 2,3. Filebench crea el 100 % de la estructura antes de iniciar las mediciones.
Se ha modificado el número de hilos simultáneos a 80.

	GlusterFS	Ceph	LizardFS
Núm. de operaciones	404116	959270	417087
Operaciones por seg.	1344,19	3194,19	1388,90
Ancho de banda (MB/s)	6,7	16,0	7,8
Latencia (ms)	152,9	65,3	146,3

Tabla 4.6: Prueba de Filebench para el perfil de servidor web.

GlusterFS y LizardFS presentan datos de rendimiento muy semejantes, pero una vez más, son superados ampliamente por Ceph.

Conclusión para el perfil de almacenamiento web:

En este apartado no hay duda ninguna, todas las pruebas realizadas muestran la superioridad de Ceph cuando almacena un servidor web.

En la siguiente tabla se resume el resultado de las tres pruebas anteriores, mostrando el ancho de banda obtenido para cada benchmark con su número concurrente de procesos:

	GlusterFS	Ceph	LizardFS
IOzone (50)	9,7	76,3	42,3
FIO (50)	9,7	75,8	44,9
Filebench (80)	6,7	16,0	7,8

Tabla 4.7: Resumen de anchos de banda obtenidos en las pruebas del perfil de servidor web. Cada prueba tiene entre paréntesis el número de procesos empleados y las cifras representan el ancho de banda obtenido medido en *megabytes* por segundo (MB/s).

4.4.2.3. Repositorio de código

IOzone:

Se configura IOzone para manejar ficheros de 4 KB, que es el mínimo que permite, escrituras, lecturas y tres procesos simultáneos.

```
sudo iozone -s 4k -i 0 -i 1 -RIb autotest9.wks -l 1 -u 3
```

	GlusterFS	Ceph	LizardFS
Initial Write (Mín - Max)	4391 - 19134	128 - 6239	15682 - 21625
Initial Write (Media)	**11945**	**4680**	**18256**
Rewrite (Mín - Max)	1972 - 2496	5666 - 9704	4483 - 5763
Rewrite (Media)	**2205**	**7601**	**4946**
Read (Mín - Max)	4931 - 6079	304 - 619	1956 - 2379
Read (Media)	**5482**	**420**	**2110**
Re-read (Mín - Max)	4987 - 6460	6399 - 8987	1602 - 3937
Re-read (Media)	**5579**	**7724**	**2685**

Tabla 4.8: Prueba de IOzone para el perfil de repositorio de código. Las celdas que contienen dos valores muestran los anchos de banda mínimo y máximo registrados para una determinada operación. Debajo de estas se sitúa el valor del ancho de banda medio. Todas las velocidades están representadas en *kilobytes* por segundo (KB/s).

Esta prueba ofrece resultados llamativos. Cada tecnología destaca en un tipo de operación. LizardFS es el más rápido haciendo la escritura inicial, GlusterFS es realiza las lecturas más rápidas y Ceph es mejor cuando se trata de releer o de reescribir.

FIO:

Esta prueba ha analizado separadamente el comportamiento de escritura del de lectura. Se trata de tres procesos y ficheros de entre 4 y 7 KB en ambos casos. Las llamadas a FIO son las siguientes:

```
sudo fio --output=/home/nodo3/Desktop/FIO9.txt --ioengine=libaio --iodepth=1 --name=global --rw=read
--filesize=4K-7K --numjobs=3 --runtime=300 --name=job1 --direct=1 --time_based
--file_service_type=roundrobin

sudo fio --output=/home/nodo3/Desktop/FIO10.txt --ioengine=libaio --iodepth=1 --name=global --rw=write
--filesize=4K-7K --numjobs=3 --runtime=300 --name=job1 --direct=1 --time_based
--file_service_type=roundrobin
```

	GlusterFS		Ceph		LizardFS	
	Lectura	Escritura	Lectura	Escritura	Lectura	Escritura
Número de MB	1740,5	429,29	2254,8	1261,4	1749,4	726,6
Ancho de banda total	5941	1465	7696	4305	5971	2480
Ancho de banda mínimo	1980	482	2440	1428	1990	826
Ancho de banda máximo	1981	491	2813	1440	1990	827
Latencia media de fin (µs)	2,06	2,29	1,20	1,66	1,23	1,76
Latencia media total (µs)	1210,49	674,6	1165,31	691,79	1095,89	194,97

Tabla 4.9: Prueba de FIO para el perfil de repositorio de código. Todos los anchos de banda están medidos en *kilobytes* por segundo (KB/s).

Al igual que en la prueba del perfil de almacenamiento de ficheros, Ceph es la solución más rápida salvo en la latencia total, que se ve lastrada por los valores de la latencia de envío. En esta prueba, LizardFS y GlusterFS presentan unos resultados casi parejos.

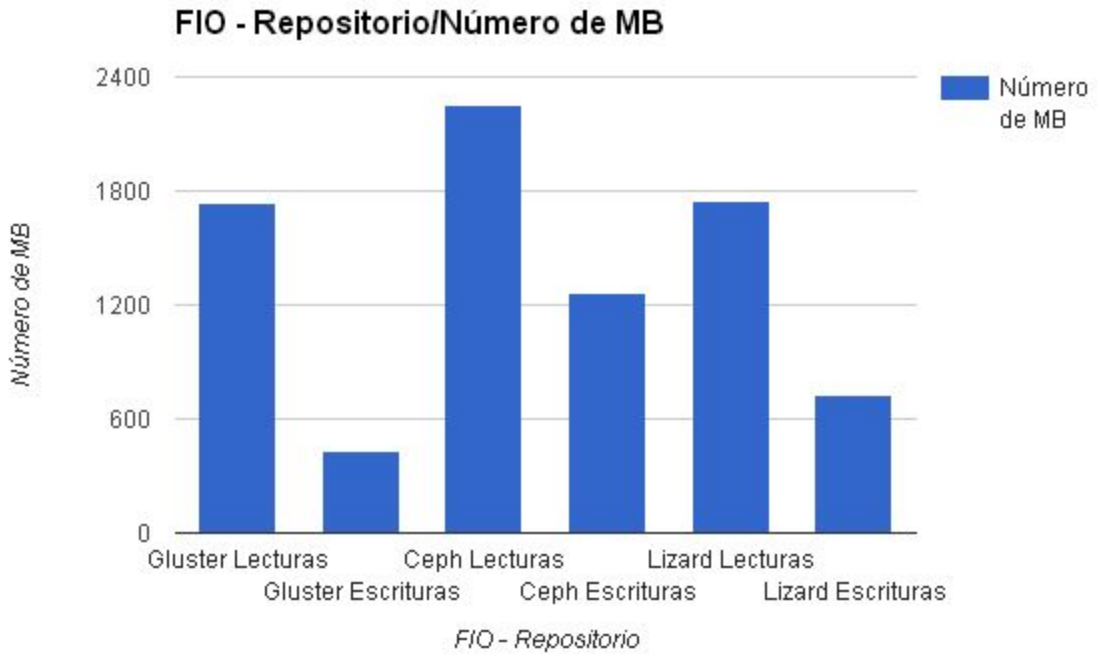


Figura 4.22: Gráfico que ilustra la tabla 4.9. Número de *megabytes* leídos y escritos por cada tecnología.

Filebench:

Para esta prueba se ha diseñado e implementado una nueva personalidad de Filebench, usando el lenguaje WML. Para más información al respecto, consultar el anexo «Nuevas personalidades de Filebench», donde se incluye el código. Los datos más importantes se pueden consultar tras la carga de la personalidad:

```
filebench> load Versions
 6397: 1842.944: Repository Version 1.0 personality successfully loaded
 6397: 1842.944: Usage: set $dir=<dir>
 6397: 1842.944:        set $filesize=<size>    defaults to 3072
 6397: 1842.944:        set $iosize=<size>  defaults to 4096
```

La estructura de ficheros y directorios (*fileset*) que se genera consta de 70 ficheros que suman 212 KB, simulando ser un repositorio de código, y se crea por completo antes de iniciar las mediciones.

	GlusterFS	Ceph	LizardFS
Núm. de operaciones	654193	593781	611617
Operaciones por seg.	1089,51	989,11	1018,60
Ancho de banda (MB/s)	968,6	818,89	844,83
Latencia (ms)	3,5	3,9	3,8

Tabla 4.10: Prueba de Filebench para el perfil de repositorio de código.

Los valores del ancho de banda son imposibles salvo si se está utilizando algún *buffer* que no se puede controlar, y probablemente esté producido por los ficheros tan pequeños que se están leyendo. En cualquier caso, aunque el dato del ancho de banda sea irreal, sí que existe una diferencia entre las tecnologías estudiadas y estas diferencias de rendimiento son coherentes y seguramente muestran cuál da un mejor rendimiento, y en este caso se trata de GlusterFS.

Conclusión para el perfil de almacenamiento de un repositorio de código:

En la prueba de IOzone se ven unos datos poco concluyentes para poder elegir a una tecnología que sea mejor que las demás. La escritura inicial es posiblemente el dato menos relevante, y es precisamente la única característica donde LizardFS destaca.

En FIO domina claramente Ceph y en la prueba de Filebench es Gluster quien queda por encima de las demás.

4.4.2.4. Servidor de correo electrónico

La única prueba eliminatoria en este apartado es la de FIO, ya que las pruebas de IOzone y Filebench se han ejecutado con otra configuración para que aporten información adicional.

IOzone:

Para obtener los datos de rendimiento de las lecturas se ha ejecutado un test automático limitado.

```
sudo iozone -RaIb autotest2.wks -g 75 -n 10
```

Tamaño de Fichero	GlusterFS	Ceph	LizardFS
20 KB	11173	10368	23065
40 KB	13455	2981	19492

Tabla 4.11: Prueba de IOzone para el perfil de servidor de correo electrónico. Se muestran los anchos de banda medidos en *kilobytes* por segundo (KB/s) de las tres tecnologías cuando leen ficheros de 20 y 40 KB.

Para ficheros de 20 y 40 KB, LizardFS es claramente la mejor solución, según IOzone.

FIO:

Esta prueba propone 50 hilos leyendo simultáneamente ficheros de 10 a 75 KB, uno tras otro como la lectura del buzón de un usuario.

```
sudo fio --output=/home/nodo3/Desktop/FIO3.txt --ioengine=libaio --iodepth=1 --name=global --rw=read
--filesize=10K-75K --numjobs=50 --runtime=600 --name=job1  --direct=1 --time_based
```

	GlusterFS	Ceph	LizardFS
Número de MB	5743	17055	20609
Ancho de banda total	9801	29106	35172
Ancho de banda mínimo	195	518	685
Ancho de banda máximo	196	662	721
Latencia media (µs)	18757,69	6349,94	4122,71

Tabla 4.12: Prueba de FIO para el perfil de servidor de correo electrónico. Todos los anchos de banda están en *kilobytes* por segundo (KB/s).

LizardFS presenta un rendimiento mejor que Ceph, y ambos a su vez mucho mejor que GlusterFS.

Filebench:

Para realizar esta prueba se utiliza la personalidad «varmail» de Filebench. Los parámetros son los siguientes:

```
load varmail
 5256: 1545.076: Varmail Version 3.0 personality successfully loaded
 5256: 1545.076: Usage: set $dir=<dir>
 5256: 1545.076:        set $meanfilesize=<size>    defaults to 16384
 5256: 1545.076:        set $nfiles=<value>     defaults to 1000
 5256: 1545.076:        set $nthreads=<value>   defaults to 16
 5256: 1545.076:        set $meanappendsize=<value> defaults to 16384
 5256: 1545.076:        set $iosize=<size>  defaults to 1048576
 5256: 1545.076:        set $meandirwidth=<size> defaults to 1000000
```

Filebench crea una estructura que tiene muy pocos niveles de profundidad (una media de 0,5) pero una anchura enorme, de hasta un millón de carpetas. Se van a generar el 80% de los ficheros antes de comenzar. Estos ficheros son 1000 y en total ocupan algo menos de 15 MB.

	GlusterFS	Ceph	LizardFS
Núm. de operaciones	125496	270306	174959
Operaciones por seg.	418,02	900,55	582,78
Ancho de banda (MB/s)	1,5	3,2	2,1
Latencia (ms)	124	55,9	88,7

Tabla 4.13: Prueba de Filebench para el perfil de servidor de correo electrónico.

Los resultados de Filebench muestran un vencedor claro en esta prueba, que es Ceph. La distancia entre los otros dos sistemas es menor que la que los separa del primero.

Postmark:

La prueba consiste en lanzar Postmark con la siguiente configuración:

```
pm>set size 10 75
pm>set subdirectories 50
pm>set number 1000
pm>set transactions 10000
pm>set buffering false
```

La ejecución se detendrá cuando se hagan 10000 transacciones. Eso significa que cuanto menor sea el valor en el campo «duración de las transacciones en segundos», mejor.

	GlusterFS	Ceph	LizardFS
Tiempo total en seg.	129	75	105
Duración de las transacciones en seg.	118	71	93
Transacciones por seg.	84	140	107
Nº de ficheros creados (ficheros por seg.)	6022 (46)	6022 (80)	6022 (57)
Nº de lecturas (lecturas por seg.)	4941 (41)	4941 (69)	4941 (53)
Nº de anexos (anexos por seg.)	4562 (38)	4562 (64)	4562 (49)
Nº de borrados (borrados por seg.)	6022 (46)	6022 (80)	6022 (57)
KB leídos (KB leídos por seg.)	251,4 (1,95)	251,4 (3,35)	251,4 (2,39)
KB escritos (KB escritos por seg.)	307,9 (2,39)	307,9 (4,11)	307,9 (2,93)

Tabla 4.14: Prueba de Postmark para el perfil de servidor de correo electrónico.

Todos los datos son iguales salvo los que están relacionados con el tiempo. Esta prueba impone una determinada carga de trabajo idéntica para todos. La diferencia está en el tiempo que tarda cada solución en completarla y de nuevo, la más eficiente es Ceph.

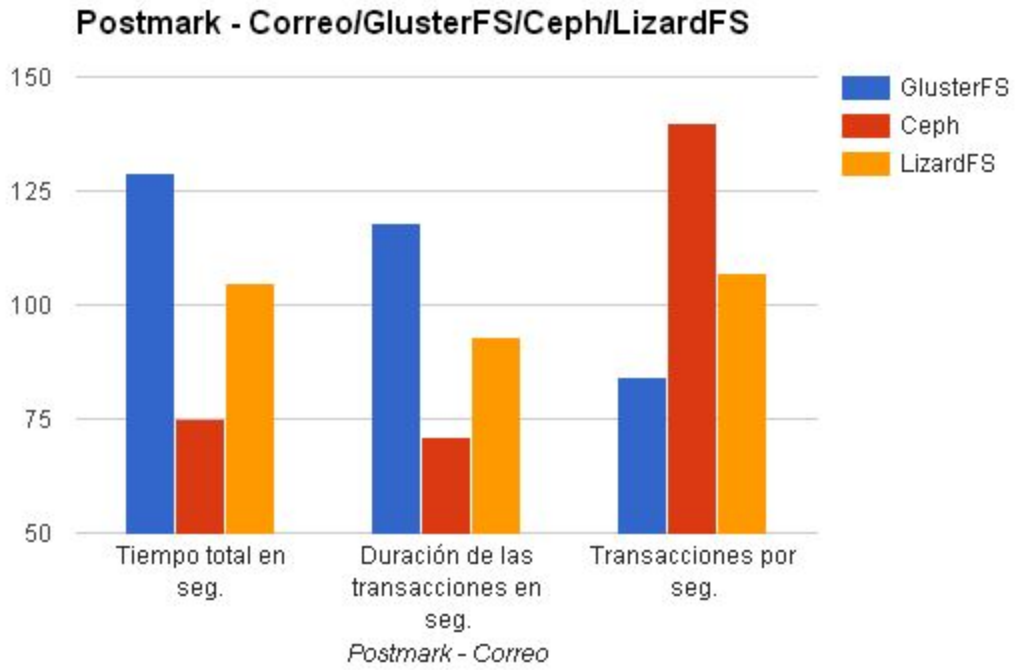


Figura 4.23: Gráfico que ilustra la tabla 4.12. Se ha seleccionado el tiempo total, la duración de las transacciones y las transacciones como indicadores del rendimiento de cada tecnología. En el caso de los tiempos, un valor menor indica que se ha completado la tarea en menos tiempo, por lo que es preferible.

Conclusión para el perfil de servidor de correo electrónico:

IOzone y FIO dan como vencedor a LizardFS mientras que Filebench y Postmark señalan a Ceph como el más rápido.

4.4.2.5. Almacenamiento de discos duros virtuales

IOzone:

Para simular el uso de un disco duro virtual se han medido lecturas secuenciales y accesos aleatorios mixtos. Para generar la siguiente tabla se han combinado los resultados de dos pruebas:

```
sudo iozone -s 4g -i 0 -i 1 -RIb autotest7.wks -l 2 -u 2
sudo iozone -s 4g -i 0 -i 8 -RIb autotest8.wks -l 2 -u 2
```

	GlusterFS	Ceph	LizardFS
Initial Write	22707	5811	49607
Rewrite	25091	8498	20463
Read	7806	14057	13615
Re-read	7946	14024	13803
Mixed Workload	732	488	269

Tabla 4.15: Prueba de IOzone para el perfil de almacenamiento de discos duros virtuales. Las casillas rojas indican que no se está cumpliendo un requisito.

Cada tecnología muestra sus puntos débiles y fuertes en esta exigente prueba. La velocidad máxima de creación del fichero corresponde a LizardFS, pero es el aspecto menos relevante de todos los medidos. Las lecturas más rápidas las realiza Ceph y las reescrituras las lleva a cabo GlusterFS. Además, el propio GlusterFS es quien hace las lecturas y escrituras aleatorias y entremezcladas más rápidas.

Según el RS-11, para esta prueba se requiere un ancho de banda en la lectura de 5 MB/s por proceso. Al haber dos procesos, son 10 MB/s. Dicha cantidad convertida a KB/s equivale a 10204 KB/s. Como se puede ver, GlusterFS no alcanza el ancho de banda mínimo exigido para las lecturas.

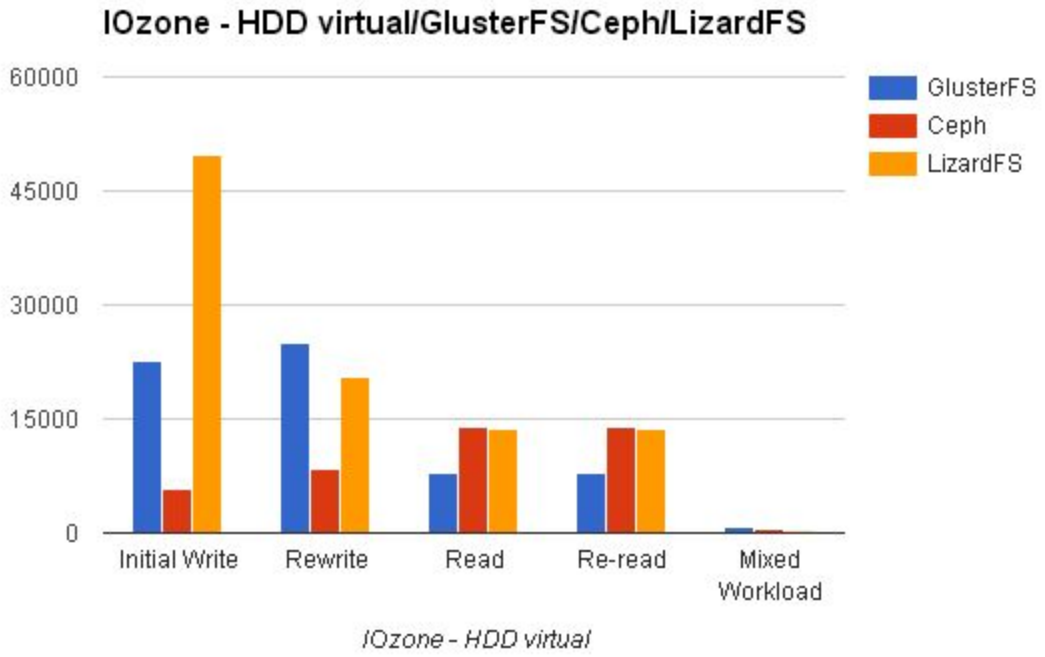


Figura 4.24: Gráfico que ilustra la tabla 4.13. Muestra el rendimiento de las tres tecnologías en las 6 operaciones seleccionadas.

FIO:

Esta es una exigente prueba que mide el rendimiento de dos procesos realizando lecturas y escrituras aleatorias en un fichero de 4 GB durante una hora.

```
sudo fio --output=/home/nodo3/Desktop/FIO17.txt --ioengine=libaio --iodepth=1 --name=global --rw=randrw
--filesize=4G --numjobs=2 --runtime=3600 --name=job1 --direct=1 --time_based
```

	GlusterFS		Ceph		LizardFS	
	Lectura	Escritura	Lectura	Escritura	Lectura	Escritura
Número de MB	565,17	564,68	994,29	992,88	877,84	878,40
Ancho de banda total	160	160	282	282	124	124
Ancho de banda mínimo	80	80	139	139	62	62
Ancho de banda máximo	83	83	143	142	62	62
Latencia media (µs)	49453,54	300,23	28124,80	160,61	63860,3	209,49

Tabla 4.16: Prueba de FIO para el perfil de almacenamiento de discos duros virtuales. Las casillas rojas contienen valores que no cumplen los requisitos. Todos los anchos de banda están medidos en KB/s.

Para FIO, el mejor resultado lo ofrece Ceph seguido de cerca por LizardFS.

Según el RS-11, para esta prueba se requiere un ancho de banda total de 125 KB/s por proceso. Al haber dos procesos, son 250 KB/s. Como se puede apreciar en la tabla, ni GlusterFS ni LizardFS cumplen el requisito.

Filebench:

Para realizar esta prueba ha sido conveniente diseñar e implementar una nueva personalidad de Filebench, usando el lenguaje WML. Para más información al respecto, consultar el anexo «Nuevas personalidades de Filebench», donde se incluye el código de las personalidades escritas para este proyecto. Como en cualquier personalidad, los parámetros más importantes se pueden consultar tras la carga de la personalidad:

```
filebench> load VM
 6797: 4.606: VM Software Version 1.0 personality successfully loaded
 6797: 4.606: Usage: set $dir=<dir>
 6797: 4.606:        set $filesize=<size>    defaults to 4294967296
 6797: 4.606:        set $iosize=<size>  defaults to 4096
```

Esta personalidad va a crear un fichero de 4 GB y va a realizar lecturas y escrituras aleatorias durante una hora.

	GlusterFS	Ceph	LizardFS
Núm. de operaciones	488882	649353	155327
Operaciones por seg.	135,68	180,20	43,10
Ancho de banda (MB/s)	0,5	0,7	0,2
Latencia (ms)	29,4	21,4	92,7

Tabla 4.17: Prueba de Filebench para el perfil de almacenamiento de discos duros virtuales. Las casillas rojas contienen valores que no cumplen los requisitos.

De nuevo Ceph consigue la mejor valoración con mucha diferencia. LizardFS es el mayor perjudicado con esta carga de trabajo.

Según el RS-11, para esta prueba se requiere un ancho de banda total de 125 KB/s por proceso. Al haber cuatro procesos, son 500 KB/s o 0,48 MB/s. Observando los datos ofrecidos en la tabla, LizardFS no llega al mínimo exigido.

4.4.2.6. Información adicional

En este apartado se incluye la información relevante que no está directamente relacionada con el rendimiento de los diferentes perfiles de almacenamiento.

Fdtree:

Esta prueba pretende ilustrar el rendimiento de las distintas soluciones cuando tienen que enfrentarse a tareas intensas de modificación de metadatos. La llamada a Fdtree es:

```
sudo ../../home/nodo3/Downloads/fdtree-1.0.2/fdtree.bash -l 3 -d 5 -f 5 -s 1
```

	GlusterFS	Ceph	LizardFS
Número total de directorios	156	156	156
Directorios creados por seg.	52	156	156
Directorios borrados por seg.	78	156	156
Número total de ficheros	780	780	780
Ficheros creados por seg.	78	260	60
KiB/s	312	1040	240
Ficheros borrados por seg.	390	390	390

Tabla 4.18: Prueba de Fdtree para valorar el tratamiento de los metadatos.

Con esta prueba se propone una determinada tarea que en cifras absolutas va a ser igual para todos los sistemas: mismo número de ficheros y de directorios. La diferencia está en lo rápido que la hace cada uno.

La primera tecnología que queda descolgada es GlusterFS, que no puede seguir el ritmo de creación y borrado de directorios de las otras dos. En la creación de ficheros, Ceph supera a LizardFS.

5. Planificación y presupuesto

5.1. Planificación

La planificación consta de dos fases: la identificación de las distintas tareas que componen el proyecto y la confección de un diagrama de Gantt para ilustrar los plazos.

5.1.1. Tareas

Es necesario subdividir el proyecto en tareas más pequeñas para su ejecución. Para poder mostrarlas con claridad y facilitar el seguimiento de las tareas identificadas, se presentan en forma de recuadro resumen.

Código de la tarea	Nombre	Duración estimada en horas
Descripción		

Tabla 5.1: Plantilla para la descripción de tareas.

- Análisis:

TSK_A_01	Reunión con miembros del grupo	8
Para identificar claramente las necesidades de almacenamiento del grupo de investigación, se debe mantener una reunión con los interesados. De esta manera se obtiene la información necesaria para elaborar los requisitos.		

Tabla 5.2: Tarea TSK_A_01.

TSK_A_02	Formalización de los requisitos	8
Se revisan las notas escritas o la grabación de la reunión y se elabora un documento que recoja todas las necesidades de almacenamiento del grupo y que las describa de manera precisa y sin ambigüedades. Este documento debe ser aprobado por el responsable del grupo.		

Tabla 5.3: Tarea TSK_A_02.

- Diseño:

TSK_D_01	Investigación de las tecnologías de almacenamiento existentes	30
Primero hay que localizar las tecnologías de almacenamiento distribuido más extendidas, estudiar sus características principales y comprobar si cumplen los requisitos. Las tecnologías que pasen este primer filtro son sometidas a estudios más profundos (consultar Anexos).		

Tabla 5.4: Tarea TSK_D_01.

TSK_D_02	Diseño del entorno de pruebas para cada tecnología	4
Cada una de las tecnologías estudiadas puede tener requisitos de funcionamiento diferentes. En esta tarea se diseñan los entornos apropiados y se hacen sus esquemas respectivos para facilitar tareas posteriores.		

Tabla 5.5: Tarea TSK_D_02.

TSK_D_03	Investigación de productos de *benchmarking*	10
Búsqueda de información y características del *software* de *benchmarking* de almacenamiento. Comparación entre los más populares y selección de los más adecuados.		

Tabla 5.6: Tarea TSK_D_03.

TSK_D_04	Diseño de un plan de pruebas	16
Consiste en crear un listado de pruebas que han de ejecutarse con la ayuda de los *benchmarks* en las soluciones de almacenamiento seleccionadas, a fin de obtener datos de rendimiento comparables que sirvan para tomar una decisión.		

Tabla 5.7: Tarea TSK_D_04.

- Implementación:

TSK_I_01	Instalación del *software* de virtualización	2
Adquisición de la licencia, descarga e instalación del *software* de virtualización con el que se montará el entorno de pruebas.		

Tabla 5.8: Tarea TSK_I_01.

TSK_I_02	Creación de las máquinas virtuales	8
Descarga del sistema operativo. Creación de las máquinas virtuales conforme a las especificaciones del diseño del entorno de pruebas. Instalación del S.O. en todas las máquinas.		

Tabla 5.9: Tarea TSK_I_02.

TSK_I_03	Conexión y configuración de las máquinas virtuales	2
Configuración de red de cada máquina virtual. Configuración de los cortafuegos para permitir el tráfico de las tecnologías de almacenamiento.		

Tabla 5.10: Tarea TSK_I_03.

TSK_I_04	Instalación de las soluciones de almacenamiento	40
Descarga, instalación, configuración y prueba de las tecnologías de almacenamiento.		

Tabla 5.11: Tarea TSK_I_04.

TSK_I_05	Instalación de los benchmarks	2
Descarga, instalación, prueba y en algunas casos compilación de los *benchmarks* que se van a utilizar.		

Tabla 5.12: Tarea TSK_I_05.

- Pruebas:

TSK_P_01	Ejecución del plan de pruebas	60
Ejecutar todas las pruebas descritas en el plan de pruebas y guardar la salida de cada *benchmark*.		

Tabla 5.13: Tarea TSK_P_01.

TSK_P_02	Ordenación y presentación de los resultados	16
Recopilar todos los datos de los tests, depurarlos y presentarlos de manera legible y cómoda. Por último se comparan los datos de cada *benchmark* en las distintas tecnologías y se resumen las diferencias encontradas. Con esta información finalmente se puede tomar una decisión sobre la idoneidad de cada una.		

Tabla 5.14: Tarea TSK_P_02.

- Evaluación:

TSK_E_01	Evaluación de los datos	2
Con la información obtenida anteriormente se puede tomar una decisión sobre la idoneidad de cada tecnología de almacenamiento estudiada.		

Tabla 5.15: Tarea TSK_E_01.

5.1.2. Diagrama de Gantt

Se han fijado jornadas de ocho horas de trabajo. Se estima que no hay días de descanso salvo los fines de semana. Se excluyen imprevistos como falta de disponibilidad de los miembros del grupo de investigación para las reuniones, etcétera.

Etapas proyecto	Duración estima...
Análisis	**Horas**
TSK_A_01 Reunión con miembros del grupo	8
TSK_A_02 Formalización de los requisitos	8
Diseño	**Horas**
TSK_D_01 Investigación de las tecnologías de almacenamiento existentes	30
TSK_D_02 Diseño del entorno de pruebas para cada tecnología	4
TSK_D_03 Investigación de productos de benchmarking	10
TSK_D_04 Diseño de un plan de pruebas	16
Implementación	**Horas**
TSK_I_01 Instalación del software de virtualización	2
TSK_I_02 Creación de las máquinas virtuales	8
TSK_I_03 Conexión y configuración de las máquinas virtuales	2
TSK_I_04 Instalación de las soluciones de almacenamiento	40
TSK_I_05 Instalación de los benchmarks	2
Pruebas	**Horas**
TSK_P_01 Ejecución del plan de pruebas	60
TSK_P_02 Ordenación y presentación de los resultados	16
Evaluación	**Horas**
TSK_E_01 Evaluación de los datos	2

Figura 5.1: Listado de tareas del diagrama de Gantt.

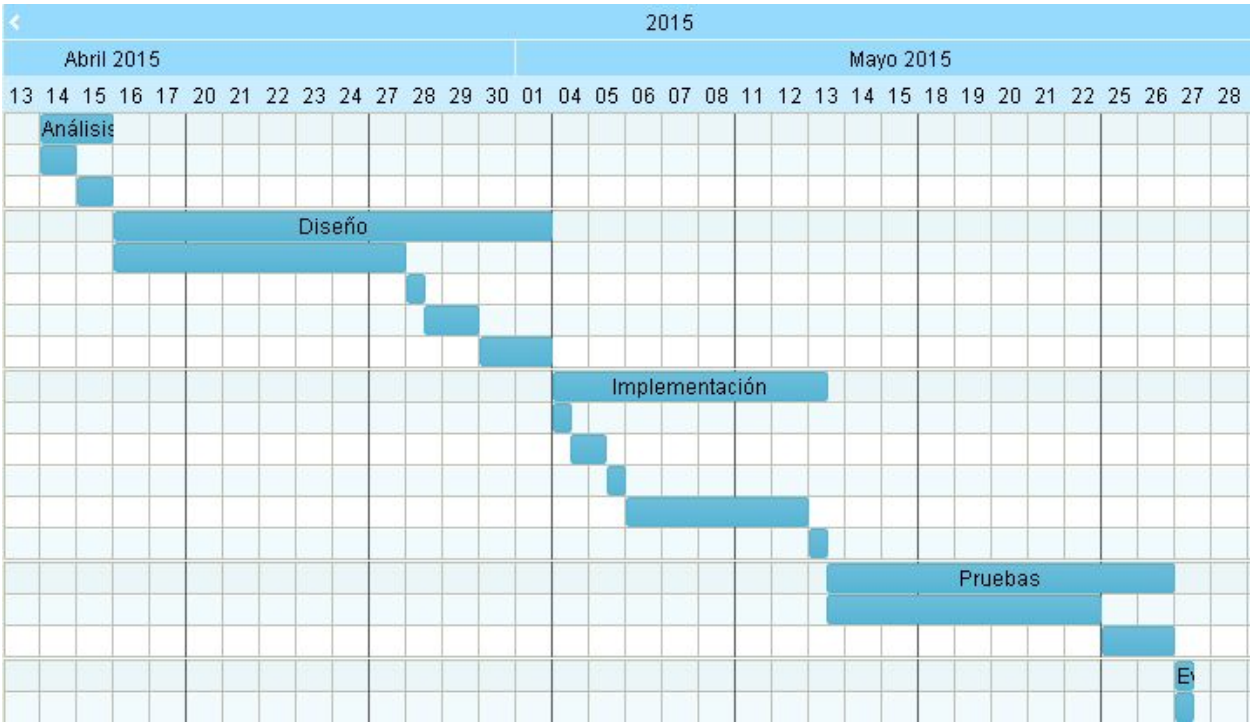


Figura 5.2: Planificación temporal del diagrama de Gantt.

5.2. Presupuesto

Una vez que se tienen los plazos, se puede calcular el presupuesto total del proyecto. Para ello se calculan todos los tipos de costes que hay que afrontar:

- Costes de *hardware*.
- Costes de *software*.
- Costes de personal.
- Costes de material fungible.
- Otros costes.

Todos los costes que aparecen en este apartado tienen el IVA incluido.

5.2.1. Costes de *hardware*

Es necesario calcular el coste que un servicio de almacenamiento distribuido, como la solución óptima, supone para el grupo de investigación. Pero no solo se va a calcular el coste del despliegue en la propia universidad, sino una alternativa viable como puede ser el alquiler de máquinas virtuales que una empresa ofrezca como servicio.

Ya que la solución seleccionada se beneficia de la capacidad de procesamiento de cada nodo, no es una alternativa válida comparar el despliegue de esta con un simple alquiler de almacenamiento remoto.

Conviene resumir algunas de las diferencias más significativas antes de entrar a valorar los costes.

	***Hardware* local**	**Virtualización externalizada**
Adquisición del *hardware*	*Hardware* convencional. No es necesario un gran desembolso ya que incluso se pueden reutilizar máquinas obsoletas provenientes de aulas docentes, equipos de ofimática, etc.	Se trata de un servicio. No se adquiere nada más que el derecho de uso de las máquinas virtuales o de los ciclos de reloj.
Seguridad	Las máquinas estarían dentro del ámbito universitario y se puede limitar la salida a Internet de las partes más delicadas o confidenciales. Cuenta con la seguridad implementada en el campus.	No se tiene el control real de los datos, ya que no están ubicados físicamente en el campus, sino en la nube. Cualquier acceso hace que viajen los datos desde el servidor de máquinas virtuales hasta la universidad. La seguridad depende de las medidas implementadas por la empresa proveedora del servicio.

Mantenimiento	Al ser el *hardware* propio, cualquier avería o mejora corre a cargo del grupo de investigación.	El *hardware* real sobre el que se asientan las máquinas virtuales es competencia de la empresa.
Tiempos de respuesta	Lo normal es que los datos no se almacenen en dispositivos de máximo rendimiento y se acceda a través de una LAN.	Hay disponibles SSD para el almacenamiento. La parte negativa es que la latencia a la hora de comunicar con el servidor de máquinas virtuales es con seguridad, mayor que la que pueda encontrarse en una LAN.
***Hardware* de red**	Como el resto del *hardware* en esta opción, corre a cargo de la universidad. Tanto los dispositivos como su mantenimiento y cableado.	La infraestructura de red también es incumbencia de la empresa.
Consumo energético	El coste de tener un *cluster* encendido repercute directamente sobre la universidad.	El consumo viene incluido en el precio, aunque no especificado.
Limitaciones especiales del servicio de alquiler	-	Existe una serie de limitaciones propias de esta solución, como por ejemplo los costes asociados a la ejecución de código del cliente basado en eventos, o el coste anual por giga.

Tabla 5.16: Tabla comparativa resumen de las características más relevantes de la compra de *hardware* y del alquiler de un servicio de virtualización externo.

5.2.1.1. *Hardware* local

El coste del *hardware* real sobre el que implementar la solución definitiva depende del número de nodos del *cluster*. Para poder hacer un cálculo aproximado, se supone que el *cluster* de almacenamiento está integrado inicialmente por dos máquinas.

En el caso de que la universidad no dispusiera del *hardware* obsoleto o sobrante para destinar a este *cluster*, el cálculo de los costes se hace sobre *hardware* nuevo.

	***Hardware* nuevo**
CPU	AMD A4-4000, 2 núcleos, 3GHz
Memoria	4 GB, DDR3, 1333 MHz
Almacenamiento en GB	1024
Coste total	233,73 €

Tabla 5.17: Coste de una máquina nueva con las características detalladas. Precio con IVA incluido.

Un ordenador con estas características encendido las 24 horas del día pero sin funcionar a pleno rendimiento todo el tiempo consume aproximadamente 28W que multiplicado por 24 horas al día es igual a 0.672 kWh. Teniendo en cuenta que el coste medio de la electricidad en España es de 0.145 € / kWh, el coste diario de mantener encendido el nodo sería de 0.097 €, o lo que es lo mismo, **35,56 € anuales**.

Para interconectar los nodos del *cluster* entre sí y con vistas a que pueda crecer, se debe adquirir *hardware* de red, de nuevo contando con que no se disponga de esta infraestructura disponible en el grupo de investigación. Un *switch* de 48 entradas RJ-45 y algunos cables de red tienen un precio de **334 €**.

	Inversión inicial	Coste anual
Ordenadores	233,73 € x 2	-
Infraestructura de red	335 €	-
Coste energético	-	35,56 € x 2
Coste total	802,46 €	71,12 €

Tabla 5.18: Coste total del primer año: adquisición y mantenimiento. Todos los precios tienen IVA.

5.2.1.2. Virtualización externalizada

Como contrapartida al montaje sobre *hardware* real y mantenimiento de la infraestructura de almacenamiento distribuido, existe la posibilidad de externalizar el servicio a un proveedor de máquinas virtuales a través de Internet.

Para realizar una comparativa con los costes de la anterior solución, se estudia el sistema Amazon EC2. Explicado brevemente, este sistema consiste en el alquiler de máquinas virtuales que se ejecutan en los servidores de Amazon. Los precios varían según varios factores, como las características técnicas de las máquinas virtuales, la disponibilidad, el *software* instalado, su ubicación geográfica real y hasta la modalidad de pago escogida.

La configuración del paquete de EC2 será lo más semejante posible a la del entorno real para que la comparativa sea lo más próxima a la realidad. Se han escogido dos posibles opciones; la primera con una configuración normal y la segunda con las mínimas características que se permiten.

Todos los precios mostrados son válidos solo para el primer año. Incluye dos máquinas virtuales de cada tipo, situadas en la UE (Franckfurt), y con disponibilidad de 24 horas al día durante un año. La modalidad de pago seleccionada es un pago anual por adelantado. Los impuestos no están incluidos. Estas son las condiciones más ventajosas.

	t2.medium	t2.micro
vCPU	2	1
Memoria en GB	4	1
Almacenamiento en GB	30	30
Coste anual	352 $	88 $

Tabla 5.19: Comparativa de características y precio entre dos tipos de máquinas t2. Impuestos no incluidos.

Una vez vistos estos precios, se descarta inmediatamente la opción de *hardware* intermedio, t2.medium. La opción más modesta, sigue pareciendo atractiva en lo que se refiere al aspecto económico, pero su capacidad de almacenamiento es irrisoria para las necesidades del grupo de investigación: solo 30 GB por máquina, que en cuanto haya una redundancia mínima, se convierten en 30 GB totales de los que hay que descontar el sistema operativo, el *software* necesario y el espacio libre que demanda la solución para su funcionamiento óptimo.

Pero, ¿a qué precio es posible aumentar el almacenamiento en la solución t2.micro? Un cálculo rápido en el configurador web de Amazon [I] nos hace ver que añadir 500 GB a cada máquina virtual a velocidad de disco duro magnético, incrementa el coste en un único pago anual de 59 $ más otros 57,23 $ mensuales, lo que hace **un total de 1008,84 $** (88 dólares + 59 dólares + 57,23 dólares multiplicado por 12 meses + 21% de IVA). Es una cantidad enorme por apenas 500 GB de almacenamiento.

5.2.1.3. Comparación de costes de *hardware*

	t2.micro (500 GB)	***Hardware* propio (1024 GB)**
Coste primer año	916,46	873,58
Coste segundo año	916,46	71,12
Coste tercer año	916,46	71,12
Coste cuarto año	916,46	71,12
Coste quinto año	916,46	71,12
Coste total primer quinquenio	4582,30 €	1158,06 €

Tabla 5.20: Coste de los cinco primeros años de las dos posibles soluciones. La diferencia hace que incluso en un año sea mejor la opción del *hardware* en propiedad.

Todos los precios se expresan en euros. La tasa de conversión USD - EUR aplicada a los 1008,84 dólares para convertirlos en los 916,46 euros es, a fecha de la estimación de los costes, de 1,1008.

Si se elige un periodo de tiempo superior a un año la opción más favorable es la del hardware en propiedad. Incluso durante el periodo de un año, la capacidad de almacenamiento de la virtualización es la mitad que la de la otra opción.

5.2.2. Costes de *software*

	Precio
Ubuntu 14.04.2 LTS	-
Sistemas de almacenamiento distribuido	-
Benchmarks	-
VMware Workstation 11	226,22 €
VMware Player	-
Ofimática (Google Docs)	-
Diagrama de casos de uso y Gantt (Recursos web) [II, III]	-
TOTAL	226,22 €

Tabla 5.21: Coste de los recursos *software* necesarios para la ejecución del proyecto completo.

Los costes de *software* son muy reducidos. Prácticamente todo el *software* es libre o gratuito.

5.2.3. Costes de personal

La siguiente tabla resume el coste total de personal. Muestra los roles que intervienen en el proyecto, el precio aproximado por hora de trabajo de cada uno de ellos, el número de horas necesario y los totales.

	Coste por hora	Número de horas	Coste total
Jefe de proyecto	56,25	8	450
Analista	41,25	14	577,5
Diseñador	41,25	58	2392,5
Administrador de sistemas	35	128	4480
TOTALES	-	208	**7900 €**

Tabla 5.22: Coste del personal implicado en el proyecto. Todos los costes están en euros. Datos de la empresa Vass Consultoría de Sistemas SL.

El administrador de sistemas se encarga de las fases de implantación, pruebas y evaluación. Se ha optado por roles de perfil alto, es decir, sénior.

5.2.4. Costes de material fungible

En este apartado se detallan los costes relacionados con el material de oficina empleado.

	Coste
Papel DIN A4 80 g/m²	2,50
Tóner negro	1,14
Impresión y encuadernación	200
Otros	15
Total	**218,64 €**

Tabla 5.23: Costes del material fungible empleado durante el proyecto.

5.2.5. Otros costes

Aquí se incluyen los gastos que por sus características no se incluyen en otras partidas presupuestarias de mayor cuantía, pero que no deben dejar de tenerse en cuenta. Por ejemplo, los gastos en transporte (combustible y transporte público), los gastos en comunicaciones (llamadas y datos de navegación móvil para trabajo conectado), etcétera.

	Coste
Transporte	14,28
Comunicaciones	7,43
Otros	5
Total	**26,71 €**

Tabla 5.24: Costes adicionales derivados de la ejecución del proyecto.

5.2.6. Coste total

En resumen, el coste total del proyecto se recoge en la siguiente tabla:

Coste de *hardware* (primer año)	873,58
Coste de *software*	226,22
Coste de personal	7900
Coste de material fungible	218,64
Otros costes	26,71
Subtotal	9245,15
Costes indirectos (21%)	1941,49
Beneficios (14%)	1294,32
TOTAL	**12480,96 €**

Tabla 5.25: Coste total del proyecto y desglose de todos los costes.

6. Conclusiones

6.1. Conclusiones del resultado del proyecto

Una vez recogidos, resumidos y presentados todos los datos de las pruebas realizadas se puede analizar y comparar cada una de las tecnologías.

Lo primero que hay que señalar es que como se ha podido ver en el capítulo 4, GlusterFS no ha alcanzado el mínimo ancho de banda exigido por los requisitos en tres ocasiones, mientras que LizardFS ha fallado en una sola ocasión. **La única tecnología que ha cumplido siempre los requisitos ha sido Ceph, por lo que se considera la solución más adecuada, en función de los requisitos, para el problema presentado.**

Si se cuentan las veces que cada tecnología ha resultado ser la que ofrece un mayor rendimiento en una prueba, se encuentran los siguientes datos:

	GlusterFS	Ceph	LizardFS
Almacenamiento de ficheros	0	2	1
Servidor web	0	3	0
Repositorio de código	1	2	0
Servidor de correo	0	2	2
Discos duros virtuales	0	2	1
Rendimiento de metadatos	0	1	0

Tabla 6.1: Número de veces en las que una tecnología ha resultado ser la más rápida en una prueba.

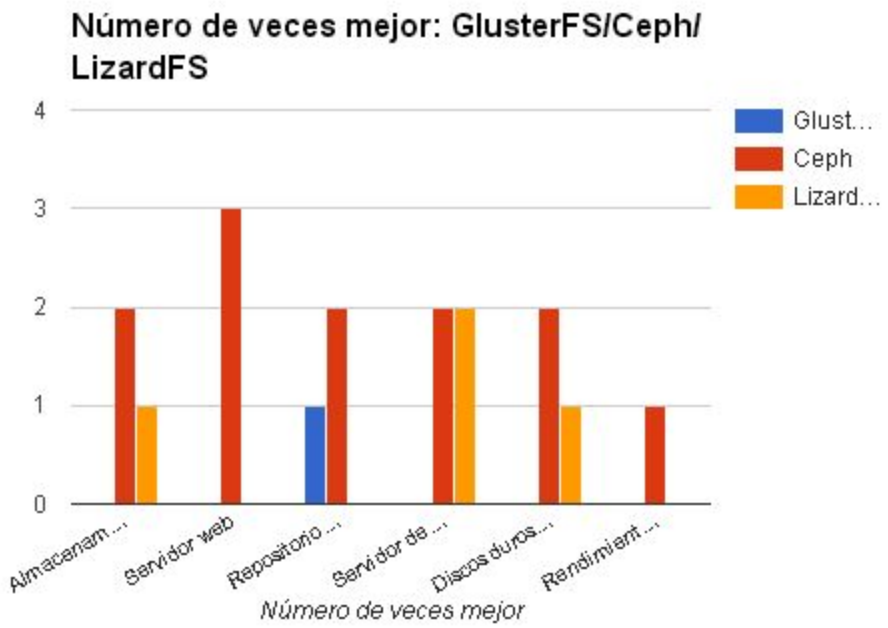


Figura 6.1: Gráfico que ilustra la tabla 6.1. Número de veces que cada tecnología ha resultado ser la más rápida en las pruebas de cada perfil de almacenamiento.

Aunque LizardFS ha fallado una prueba en un perfil de almacenamiento, casi siempre ha mantenido un rendimiento cercano al de Ceph y en ocasiones lo ha superado. Eso, sumado a la simplicidad a la hora de desplegarlo, lo convierte en una alternativa excelente a Ceph si se puede sacrificar un poco de rendimiento en favor de una mayor sencillez en la instalación, configuración y sobre todo, monitorización del sistema.

Se consideran perfectamente cumplidos los objetivos propuestos al inicio del estudio. Se ha conseguido determinar cuál es la mejor solución de almacenamiento distribuido para las necesidades planteadas por el grupo de investigación. Además de eso, se ha comprobado que LizardFS es una gran alternativa que supera en algunas facetas a Ceph.

Finalmente **el precio de todo el proyecto asciende a 12480,96 €,** que es una cantidad ajustada y razonable. El siguiente gráfico muestra el desglose de dicho importe.

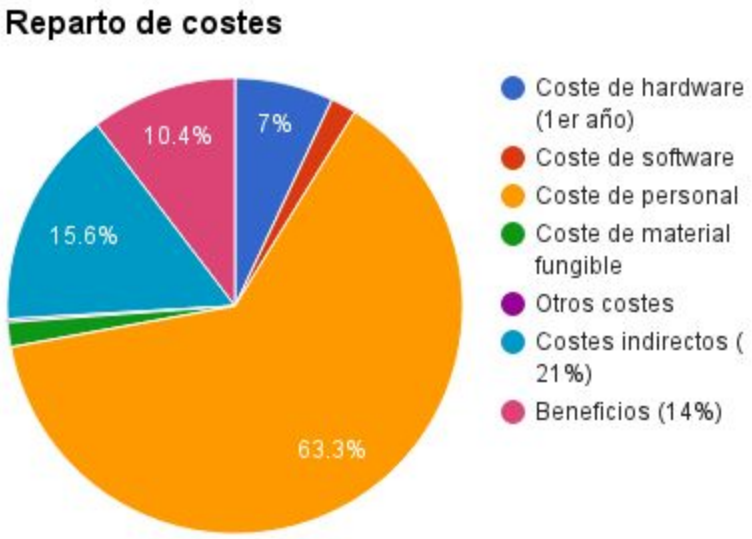


Figura 6.2: Gráfico que ilustra la tabla 5.25. Distribución de los distintos costes.

6.2. Conclusiones del proceso

El proceso seguido para la implementación de las soluciones ha estado plagado de complicaciones. Estas complicaciones son propias del primer proceso de montaje, no se reproducirían en el entorno de producción. Los dos peores inconvenientes han sido:

- Limitaciones de recursos: ejecutar cinco máquinas virtuales con una intensa carga de trabajo más el sistema operativo anfitrión es una tarea realmente exigente para un ordenador de sobremesa doméstico. Aun así, el rendimiento fue el adecuado como para poder trabajar y medir las velocidades de E/S de los sistemas de almacenamiento.
- *Bug* en el entorno: cuando se inició la construcción del entorno de pruebas, el *software* escogido era diferente. La plataforma de virtualización era VMware Workstation 9.0 y el sistema operativo destinado a albergar los nodos de almacenamiento era Ubuntu 10.10. Después de haber hecho todo el proceso de creación, instalación y configuración de los nodos, se iniciaron las primeras pruebas. Durante estas pruebas se empezaron a advertir problemas en la conectividad de las máquinas; al principio no había conexión y

tras establecer en numerosas ocasiones la configuración de red, se conseguía que funcionara durante unos instantes, pero al poco rato las máquinas volvían a ser inalcanzables desde sus pares. Tras un gran número de horas de investigación, documentación y prueba de posibles soluciones al problema, se optó por abandonar esa selección de *software* ya que el problema parecía sobrepasar los límites de un comportamiento lógico y determinista desde el punto de vista del administrador del sistema, ya que ocurría solo tras un periodo de tiempo aparentemente aleatorio. El problema se manifestaba primero con Ubuntu anunciando que se había desconectado el cable de red y eso provocaba que las máquinas perdieran su configuración de red, aunque siguiese presente en su fichero de configuración /etc/network/interfaces. Al final se optó por cambiar a las últimas versiones del software implicado y el problema nunca más se reprodujo, con los mismos pasos seguidos para la creación del entorno y las mismas configuraciones de red.

La ejecución de las pruebas ha durado más de lo previsto inicialmente. En algunas ocasiones, una sola prueba podía durar hasta 13 horas, lo que retrasaba el avance del proyecto notablemente por no poderse utilizar el ordenador para ninguna otra tarea a fin de evitar que los datos que se medían se vieran alterados. Todas las mediciones se hicieron un mínimo de tres veces para las más largas y un máximo de diez para las más breves y por lo tanto más sensibles a bajadas de rendimiento puntuales, para detectar datos atípicos producidos por factores externos como la descarga de una actualización del sistema operativo anfitrión u otros similares. También se evitó hacer todas las repeticiones de una misma prueba consecutivamente, sino que se fueron alternando con otras.

Solo en dos ocasiones fue necesario descartar pruebas por aparecer unos datos extremadamente alejados de los demás. Salvo esas dos excepciones, la desviación típica ha sido siempre muy reducida.

En cuanto a los *benchmarks*, son herramientas funcionales pero no muy extensamente documentadas, concretamente en el caso de Filebench, incluso con una errata en la descripción de uno de los comandos de su lenguaje WML. Adicionalmente, se desaconseja el uso de los resultados de la medición de microsegundos por operación de CPU («us/op-cpu» en la salida del programa) porque no resulta fiable.

Proceso software:

A diferencia de otros proyectos de desarrollo, este en concreto se habría beneficiado más de haber dispuesto de mayor cantidad de medios en forma de máquinas físicas en vez de más personal. El haber podido lanzar simultáneamente varias pruebas en varias plataformas distintas habría recortado notablemente el tiempo de ejecución del proyecto.

La manera en la que se ha desarrollado este estudio se asemeja más a un ciclo de vida en cascada que a uno en espiral. Las únicas fases que se han alimentado formando un bucle han sido la de diseño y la de pruebas. Tras algunas pruebas fue necesario retocar los diseños porque ofrecían resultados no concluyentes y porque el aprendizaje del uso de las herramientas de medición se hizo al mismo tiempo que las pruebas.

De hecho, esta falta de conocimientos concreta ha sido uno de los principales motivos de que la conclusión del proyecto finalmente se haya demorado aproximadamente un 10% más de lo que se previó en la estimación y que está plasmado en el diagrama de Gantt.

6.3. Conclusiones personales

Tras haber trabajado con estas tecnologías que antes de iniciar el proyecto me eran ajenas, la sensación que me han dejado, al margen de los datos de rendimiento, es que Ceph es un producto complejo, versátil y potente, mientras que GlusterFS es un producto más accesible aunque mejorable en ciertos aspectos.

Al haber tenido que comparar un sistema de almacenamiento distribuido controlado por el departamento con sistemas remotos, como el de Amazon, he llegado a la conclusión de que su modelo de negocio no está tanto en la venta de almacenamiento *online* como en la venta de computación en la nube; están más preparados para medir y vender disponibilidad y ciclos de reloj a un precio competitivo que capacidad de almacenamiento, donde los precios se disparan hasta el absurdo si se compara con un disco duro físico local.

Para poder sacar adelante este proyecto han sido especialmente importantes los conocimientos adquiridos en las asignaturas de Sistemas Operativos, Seguridad y Protección de la Información, Arquitectura de Computadores, Transmisión de Datos y Redes e Ingeniería del Software.También mi experiencia laboral con entornos de virtualización ha sido clave.

Sin embargo, he necesitado adquirir otros conocimientos más profundos durante el desarrollo del estudio, como nuevas habilidades de administración de entornos Linux, nuevos conocimientos de sistemas de ficheros distribuidos en general y en particular los tratados en este proyecto. Además de esto, también he conocido otros sistemas menos difundidos y diversas alternativas que no han llegado a pasar ni siquiera la primera criba para aparecer en este trabajo.

6.4. Trabajos futuros

Una de las ventajas que presenta Ceph es la posibilidad de comportarse como un dispositivo de bloques. Esto es especialmente útil a la hora de almacenar discos duros virtuales. En este estudio no se ha podido configurar un escenario para poder probar el rendimiento de Ceph en esas circunstancias por la propia naturaleza del entorno de pruebas. Al tener el cliente virtualizado, no se puede instalar un sistema de virtualización en él para probar la interfaz de dispositivo de bloques de Ceph con un *software* de virtualización real.
Convendría obtener datos reales de rendimiento para determinar de qué manera conviene más almacenar ficheros grandes correspondientes a discos duros virtuales.

Este proyecto tomó como punto de partida un trabajo dirigido anterior que es el estudio de los principales tipos de almacenamiento distribuido que hay. Cuando se desarrolló el resto del proyecto y se habló de las necesidades reales del grupo de investigación se eliminaron las soluciones SQL y NoSQL por ser el montaje en un directorio un requisito fundamental. Sin embargo, existe una especificación para MongoDB que facilita el almacenamiento de ficheros. Se llama GridFS [IV] y su función es fragmentar el fichero que se quiere almacenar en partes de 255 KB para eliminar la restricción de tamaño a 16 MB que tienen los documentos BSON. Para esto emplea dos colecciones; en una guarda los datos y en otra los metadatos. Para complementar esto, existe un proyecto llamado GridFS-FUSE [V] que es un *wrapper* que permite montar GridFS como un sistema de ficheros local con la ayuda de FUSE.
Lamentablemente, en el momento de la finalización de este proyecto, el *wrapper* estaba aún en un estado muy primitivo de su desarrollo en el que no soportaba directorios, no contaba con métodos de autenticación ni con permisos, e incluso las creaciones y escrituras se encontraban en un estado experimental.

Sería interesante seguir de cerca este desarrollo por los posibles beneficios que pueda aportar tener una base de datos NoSQL que pueda almacenar ficheros y que sea a la vez accesible como un sistema de ficheros.

Continuando con la labor iniciada en este proyecto, las tareas inmediatamente siguientes deberían ser las pruebas en máquinas físicas y la simplificación del proceso de despliegue. Se debería instalar cada nodo virtualizado en una máquina física y tomar datos para comprobar si hay una desviación muy grande en comparación a los datos tomados en este estudio. En cualquier caso, los nuevos datos mostrarían siempre un mejor rendimiento que los obtenidos en este proyecto, así que no invalidarían los resultados y las conclusiones aquí obtenidas.
En cuanto al proceso de despliegue, sería conveniente clonar una de las máquinas virtuales que hacen de nodo de almacenamiento y así se simplificaría tanto el despliegue inicial como el proceso de ampliar el *cluster* de almacenamiento añadiendo una nueva máquina física. Solo con copiar la máquina virtual y realizar una leve configuración en ella, ya se dispondría de otro nodo.

Otra línea de trabajo interesante podría ser comparar a Ceph, la tecnología que ha obtenido mejor rendimiento, con Lustre. Lustre es otro conocido sistema de ficheros distribuidos cuyas principales diferencias son que no se menciona Ubuntu como sistema operativo válido en su documentación y que requiere recompilar el kernel de Linux. El cliente de Ceph también se puede instalar de esta manera y sería interesante medir las diferencias de rendimiento entre ambos, suponiendo que el grupo de investigación estuviese interesado en cambiar Ubuntu por CentOS, por ejemplo.

Anexo 1: Ceph

1. ¿Qué es Ceph?

Ceph es una plataforma de almacenamiento distribuido diseñada para funcionar sobre *hardware* convencional (*commodity hardware*) y para dar soporte a almacenamiento de objetos, bloques y archivos.
Sus principales características son:

- Es escalable hasta el *exabyte*.
- Ceph es *software* libre.
- *Self-healing*: Descubrimiento y reparación automática de errores para recuperar el funcionamiento normal del sistema.
- *Self-managing*: Configuración y optimización automáticas en función de la carga de trabajo que recibe el sistema.
- No existe ningún SPOF (*single point of failure*): No hay ningún punto del sistema que en caso de fallo por cualquier motivo inutilice por completo el propio sistema.

2. Tipos de almacenamiento soportados

Como se ha mencionado anteriormente, Ceph es capaz de almacenar datos de tres formas diferentes:

- almacenamiento de objetos,
- almacenamiento de bloques,
- almacenamiento de ficheros.

Es importante diferenciar los tres tipos y saber en qué consiste cada uno.

2.1. Almacenamiento de objetos

El almacenamiento de objetos se caracteriza por que su unidad básica de almacenamiento es el objeto. Un objeto posee:

- los propios datos que se quieren almacenar,
- una cierta cantidad de metadatos asociados,
- un identificador único.

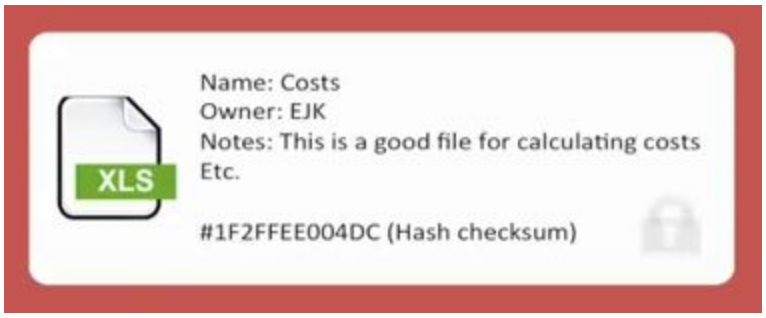


Figura A1.1: Representación de un objeto en Ceph. [10] **(Killian 2014, min. 4:00)**

El identificador suele ser el resultado de una función de *hash* de los datos junto con sus metadatos. Esto aporta integridad a los datos, ya que se puede controlar si ha habido algún cambio debido a corrupción o manipulación no autorizada.

Los metadatos pueden incluir cualquier información que se considere útil. Cuando el sistema tenga que buscar un objeto, podrá hacer búsquedas a través de los metadatos o indexarlos.

Aunque forman parte de un mismo objeto y se tratan siempre de forma conjunta, los metadatos se almacenan separados de los datos. Esto permite incluso almacenar los datos y metadatos de una manera diferente y óptima para la forma en la que se va a usar. Por ejemplo, los metadatos se pueden almacenar en bases de datos o en almacenamientos de tipo clave-valor, mientras que los datos se almacenan simplemente en binario no estructurado.

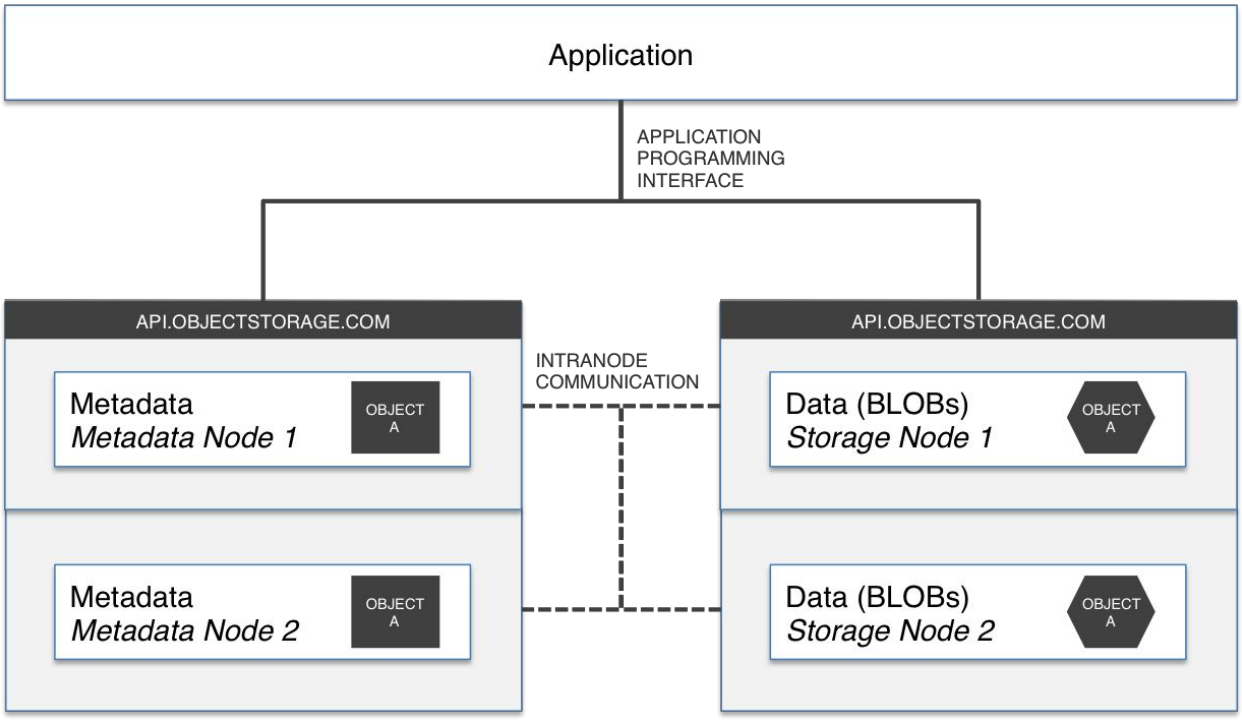
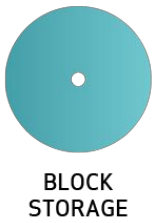


Figura A1.2: Diagrama del funcionamiento de un almacén de objetos. La aplicación se conecta a través de una API al almacén de objetos donde se aprecian dos tipos de nodos distintos: los que albergan datos y los que almacenan metadatos. Todos los nodos están conectados unos con otros. [11] **(Wikipedia 2015, Object Storage)**

Todos los objetos se almacenan al mismo nivel. Ceph se puede configurar para que exista un determinado número de copias de cada objeto, de tal manera que el sistema esté protegido frente a una posible pérdida de datos.

El almacenamiento a nivel de objeto, además de en Ceph, se puede encontrar en otros productos y servicios como GlusterFS, Amazon AWS S3, Microsoft Azure, Google Cloud Storage, Lustre y Swift, entre otros.

2.2. Almacenamiento de bloques



Se denomina almacenamiento de bloques a la manera de agrupar los datos y almacenarlos que tienen los dispositivos físicos como los discos duros. El bloque es la unidad mínima con la que trabajan los dispositivos de bloques en operaciones de lectura y escritura, y se trata de una secuencia de bits o *bytes* con un tamaño determinado denominado tamaño de bloque.

Es un sistema universalmente aceptado e implantado en la inmensa mayoría de soportes magnéticos y ópticos empleados en la actualidad.

Ceph se puede montar como si se tratara de un dispositivo de bloques. Recibe los bloques y los distribuye y replica en el *cluster*. Gestiona internamente los bloques como si fueran objetos mientras que externamente puede presentarse y comunicarse como un dispositivo de bloques.

Esta característica está especialmente indicada para plataformas de virtualización.

2.3. Almacenamiento de ficheros

Se trata del almacenamiento tradicional en el que el sistema operativo agrupa los bloques de datos en ficheros que son interpretados por las aplicaciones correspondientes. El sistema de ficheros en el que está formateado un dispositivo almacena una tabla en la que los ficheros se identifican por su nombre y su ubicación en una estructura jerárquica de directorios en forma de árbol.

Ceph posee una capa (CephFS) que le permite presentarse como un sistema de ficheros compatible con POSIX. El *cluster* de servidores de metadatos (*metadata server cluster*) proporciona un servicio semejante al de una tabla MFT del sistema de ficheros NTFS por ejemplo, y a su vez relaciona los nombres de los ficheros o directorios con los objetos en los que están realmente almacenados.

3. Arquitectura

Para comprender cómo Ceph lleva a cabo toda la funcionalidad anteriormente descrita, se puede observar el siguiente gráfico.

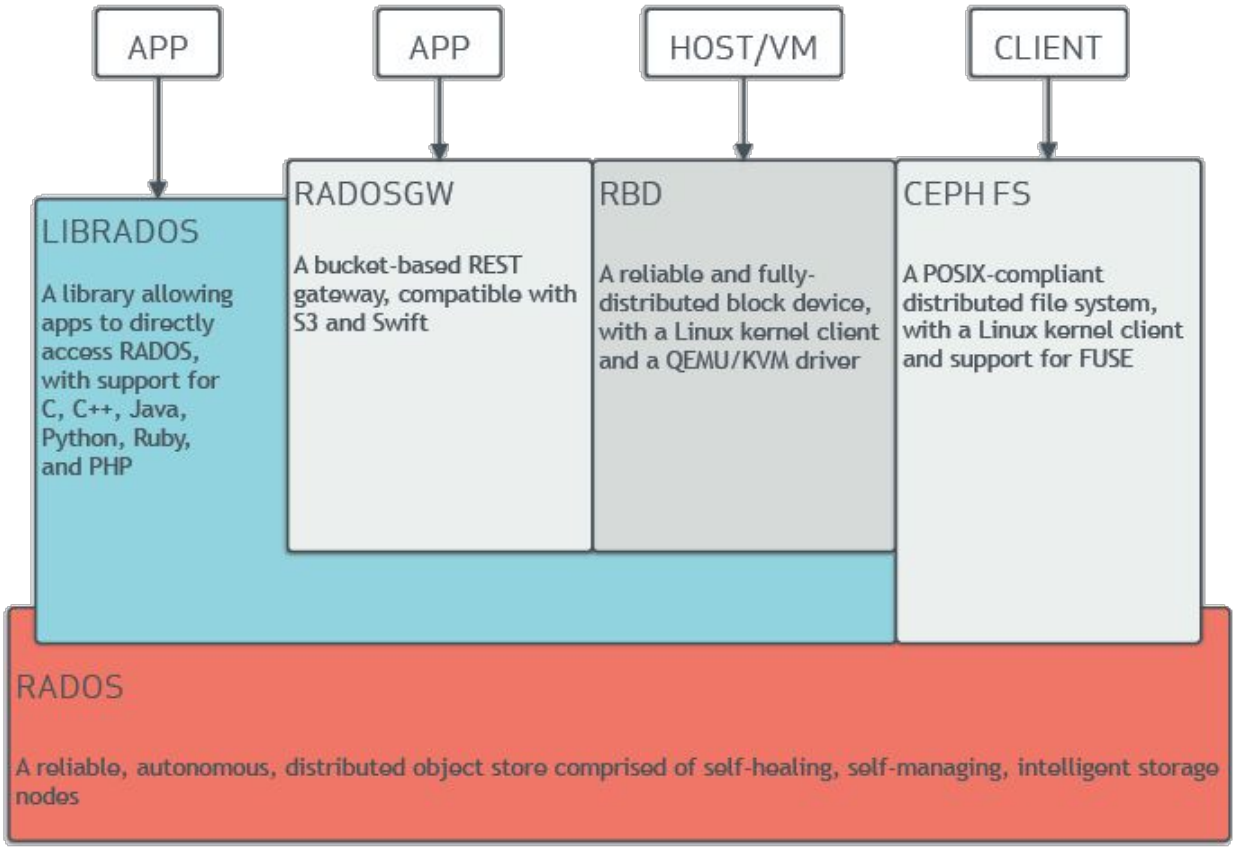


Figura A1.3: La pila de Ceph. RADOS es el almacén de objetos sobre el que se asientan los diversos servicios que ofrece Ceph. [12] **(Inktank Storage Inc. 2015, Architecture)**

Se puede observar que el diseño de la plataforma sigue una especie de pila. A continuación se explica en qué consiste cada componente de la pila.

3.1. RADOS

El acrónimo *RADOS* significa “almacén de objetos distribuido autónomo y fiable” (*Reliable Autonomous Distributed Object Storage*). RADOS es un sistema que busca sacar partido de la inteligencia y a las posibilidades que brinda un sistema de almacenamiento de bloques, para distribuir la complejidad que rodea al acceso a datos consistentes, el almacenamiento redundante, la detección de fallos y la recuperación de errores en *clusters* compuestos de miles de dispositivos de almacenamiento.
Construido como parte de Ceph, RADOS facilita una distribución evolutiva y equilibrada de los datos y de la carga de trabajo a lo largo de un *cluster* de almacenamiento dinámico y heterogéneo, mientras proporciona a las aplicaciones la ilusión de ser un único almacén lógico de objetos con una semántica de seguridad bien definida y una consistencia fuertemente garantizada. [18] (Brandt, Leung, Maltzahn y Weil 2007, p. 2)

Cuando se trata de una capacidad de almacenamiento tan grande como un *petabyte*, el *cluster* está sometido a constantes cambios; se añaden nuevos discos para incrementar la capacidad, hay constantes lecturas y escrituras, hay fallos, hay discos que se retiran debido a errores de *hardware* u obsolescencia. El objetivo de RADOS es asegurar una vista consistente de la distribución de los datos, así como lecturas y escrituras consistentes a los objetos. Para esto se emplea un mapa del *cluster* (*cluster map*) que poseen todas las partes, tanto los nodos cliente como los nodos de almacenamiento. El mapa se mantiene actualizado propagando pequeñas actualizaciones incrementales.

Como cada nodo tiene una visión del conjunto completo, pueden actuar de una manera semiautónoma, comunicándose con los demás mediante protocolos de tipo *peer-to-peer* para controlar la replicación de los datos, actualizarse de forma segura y consistente, participar en la detección de fallos y responder a ellos.

Un sistema RADOS está formado por dos grupos fundamentales:

- Un grupo grande de dispositivos de almacenamiento de objetos (Object Store Device), en adelante OSD. Un OSD tiene capacidad de procesamiento; incluye una CPU, RAM volátil, una interfaz de red y por supuesto, tiene conectado uno o varios discos duros o RAID.
- Un pequeño grupo de controladores (*monitor cluster*) que se encarga de controlar el mapa del *cluster*, donde se encuentra el número de miembros que forman parte del *cluster* y la distribución de los datos.

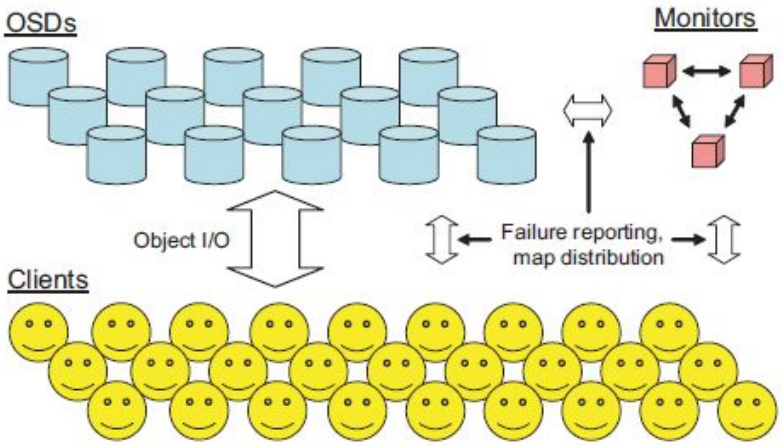


Figura A1.4: Diagrama del funcionamiento de Ceph. Los clientes tienen comunicación directa con el *cluster* de monitores y con el de OSD. A su vez los monitores están conectados con los OSD para poder actualizar y propagar la última versión del *cluster*. [13] **(Brandt, Leung, Maltzahn y Weil 2007, p. 2)**

3.1.1. Mapa del *cluster*

La gestión del *cluster* de almacenamiento se realiza exclusivamente a través del mapa del *cluster*. En el mapa se enumeran los OSD que forman parte del *cluster* y se especifica la distribución de todos los datos a lo largo de todos los OSD, es decir, en qué OSD se encuentra cada objeto. Todos los nodos de almacenamiento y todos los clientes del sistema poseen una copia del mapa del *cluster*. Como todos los nodos tienen la información global de la distribución de los objetos en el *cluster*, cada nodo posee una interfaz que permite tratar globalmente con el *cluster*.

Cada vez que el mapa cambia debido a un cambio de estado en un OSD, cambia la época (*epoch*) del mapa. Este campo funciona de manera análoga a un número de versión. La época de un mapa sirve para que los nodos se pongan de acuerdo en quién tiene la versión más reciente y por lo tanto, la información correcta. Como el sistema puede gestionar una enorme cantidad de nodos, el hecho de que un OSD cambie de estado es algo que ocurre con relativa frecuencia. Para evitar un exceso de mensajes, las actualizaciones se envían como pequeños mapas incrementales, donde se incluyen solamente los cambios que hay entre la versión antigua y la nueva.

3.1.2. Colocación de los datos

RADOS implementa una política seudoaleatoria de distribución de los objetos entre los dispositivos. Cuando se añade un nuevo dispositivo de almacenamiento al *cluster*, una submuestra de los datos existentes se traslada al nuevo espacio disponible. Esto se hace de tal manera que la carga de trabajo de todos los dispositivos del cluster se mantenga probabilísticamente equilibrada. La colocación de los datos es un proceso que se lleva a cabo en dos fases y que calcula la ubicación adecuada de los objetos sin necesidad de una gran tabla de asignación centralizada.

Antes de almacenar un objeto en el sistema se asigna a un grupo de ubicación (*placement group*, en adelante PG), que es un grupo lógico de objetos que está replicado en los mismos dispositivos. El PG que se le asigna a cada objeto viene determinado por un *hash* del nombre del objeto, el nivel de replicación deseado y una máscara de bits que controla el número total de PG del sistema.

Los PG se asignan a los OSD basándose en el mapa del cluster, que relaciona cada PG con una lista de OSD, tantos como réplicas tenga configurado. Quien se encarga de este mapeo de distribución de réplicas es el algoritmo CRUSH. El acrónimo *CRUSH* significa “ubicación descentralizada, escalable y controlada de datos replicados” (Controlled, Scalable, Decentralized Placement of Replicated Data). A grandes rasgos, CRUSH funciona como un función *hash* o resumen; los PG se distribuyen de forma seudoaleatoria pero determinista. A diferencia de una función resumen convencional, CRUSH es estable. Esto quiere decir que si uno o varios OSD abandonan el *cluster* y por lo tanto el conjunto imagen de la función se recorta, la mayoría de los PG permanecen donde están y solo se reubican los directamente afectados y pocos más para mantener el equilibrio en la carga de trabajo. Para poder controlar la carga de cada OSD, se utiliza un sistema de pesos basado en su capacidad máxima de trabajo.

3.1.3. Estado de un dispositivo

El mapa del *cluster* mantiene información relativa al estado de cada OSD. Un OSD tiene dos parejas de posibles estados:

- *up* si está *online* y se puede llegar a él, *down* si no;
- *in* si está incluido en el mapeo y tiene PGs asignados, *out* si no.

Las combinaciones de estados ofrecen cuatro posibles escenarios:

- *up* e *in* denotan el funcionamiento normal de un OSD;
- *up* y *out* significa que un OSD está disponible pero ocioso;
- *down* e *in* quiere decir que ha habido un problema y no se puede comunicar con el OSD, pero sus datos no han sido replicados en otros OSDs todavía (similar al *degraded mode* de un RAID);
- *down* y *out* es el estado de un OSD cuando falla y se le aparta del mapeo de PGs.

Para cada PG, CRUSH genera una lista de *r* OSD que que están *in*, siendo *r* el nivel de replicación del PG. Una vez generada la lista, RADOS elimina de ella los OSD que están *down* y el resultado es una lista de OSD activos para ese PG. Si el resultado es una lista vacía, RADOS considera no disponible a ese PG y bloquea todas las operaciones de E/S para sus datos.

3.1.4. Propagación del mapa

Mantener informados de la última versión del mapa a todos los OSD del sistema y en todo momento sería poco práctico, sabiendo lo numerosos que pueden llegar a ser. El único momento en el que es relevante que un OSD tenga la última versión del mapa es cuando se comunica con otro OSD o cliente. Esto hace que se pueda implementar una propagación del mapa perezosa, aprovechando para tal fin los mensajes entre nodos.

Cada OSD guarda un historial con todas las actualizaciones que ha recibido y las etiqueta con su *epoch* correspondiente. Asimismo tiene un registro con la última *epoch* conocida de cada uno de sus pares (*peers*). Cuando un OSD se comunica con un igual y en su registro consta que tiene una versión del mapa anterior a la suya, le envía las actualizaciones incrementales del mapa de manera preventiva. Si un OSD recibe un mensaje de otro OSD con una versión del mapa anterior a la suya, le actualiza. Los mensajes “latido” que intercambian para detectar errores, también sirven para propagar el mapa.

La estrategia de envío preventivo de las actualizaciones incrementales del mapa es conservadora. Un OSD envía solo las actualizaciones que no está seguro que haya recibido ya su par, para evitar que reciba información duplicada. El número de actualizaciones duplicadas que se reciben, viene dado por el número de pares que tiene un OSD, y este número a su vez está determinado por μ, que es el número de PG que gestiona.

3.1.5. Replicación

RADOS puede manejar tres esquemas distintos de replicación, que son copia primaria, cadena y separación (*primary copy*, *chain* y *splay* respectivamente). Todos tienen en común que el cliente se comunica solo con un OSD y que las réplicas son actualizadas de forma segura y consistente. Una vez que todas las réplicas están actualizadas se envía un mensaje de confirmación (*ack*) al cliente.

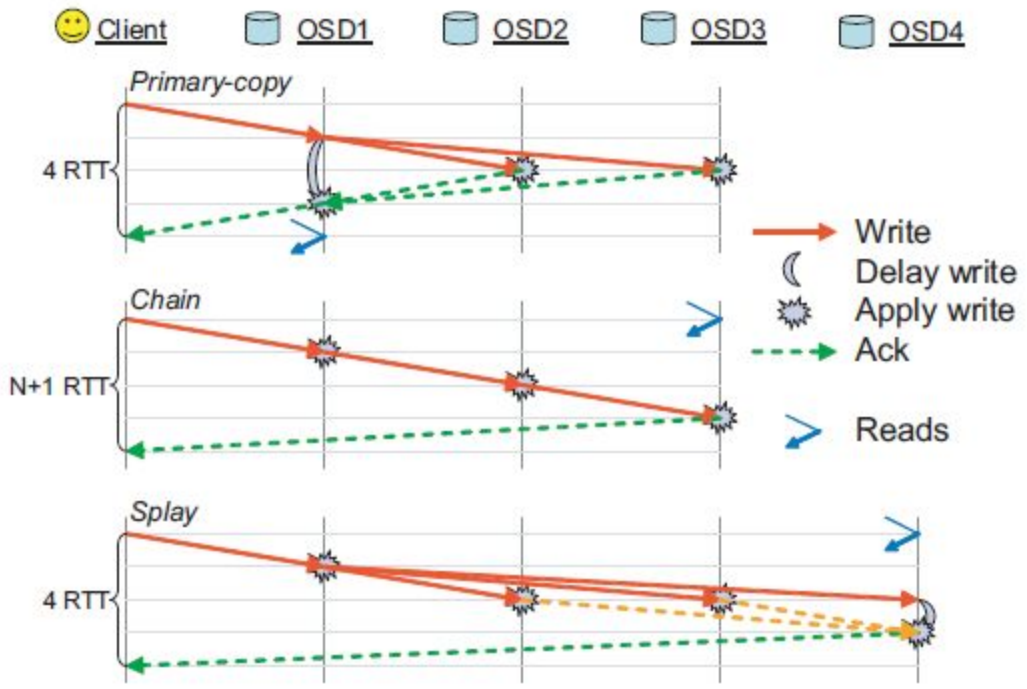


Figura A1.5: Métodos de replicación ante una escritura. Diagrama de secuencia de cada uno de los distintos protocolos que siguen los OSD de Ceph para dar por correcta una escritura y garantizar la replicación. [13] **(Brandt, Leung, Maltzahn y Weil 2007, p. 4)**

El funcionamiento de cada esquema se puede resumir así:

- Copia primaria: Actualiza todas las réplicas en paralelo. Cuando el OSD primario recibe la orden de actualización se la envía al resto de OSDs simultáneamente. Cuando le llegan las confirmaciones de escritura, actualiza su propia copia y confirma al cliente el proceso completo. Las lecturas y escrituras se llevan a cabo en el OSD primario.

- Cadena: Las actualizaciones se hacen en serie. Cuando termina de escribir en un OSD se envía la orden para que se actualice el siguiente y cuando termina el último, él mismo devuelve el *ack* al cliente. Las escrituras se envían al primer OSD de la lista (*head*) y las lecturas se hacen en el último (*tail*). Así se puede garantizar que solo se leen datos que están completamente replicados.
- Separación: Combina las actualizaciones en paralelo del esquema de copia primaria con la separación de roles a la hora de escribir y leer propia del esquema de cadena. En este caso, el OSD primario escribe y envía la orden de escritura en paralelo. El último OSD espera la confirmación de escritura de todos los demás antes de hacer la suya propia y es quien confirma al cliente.

3.1.6. Consistencia

Todas las interacciones de los clientes con los OSD llevan la versión del mapa del cluster para asegurar que las operaciones son consistentes. Si un OSD recibe una comunicación con una versión del mapa anterior a la suya, responderá con su versión actualizada. En la mayoría de las ocasiones el cliente aprenderá nuevos datos del cluster que no tienen nada que ver con su orden de lectura o escritura de ese momento, pero puede ocurrir que los datos que haya ido a buscar ya no se encuentren en ese OSD, por lo tanto puede redirigir su petición una vez que tiene el nuevo mapa.

Si la copia maestra (*master copy*) del mapa del *cluster* ha sido modificada para que un determinado PG cambie de grupo de OSD, el grupo antiguo de OSD puede seguir gestionando operaciones de lectura y escritura siempre que no se hayan enterado todavía del cambio. Si un OSD réplica se ha enterado del cambio, enviará las actualizaciones incrementales del mapa del cluster al OSD primario cuando este le envíe la orden de escritura y detecte que el valor de *epoch* no coincide. Este es un proceso seguro, porque cuando se nombra a un grupo de OSD nuevo responsable de un PG, estos contactan con todos los antiguos nodos responsables para determinar cuál es el contenido correcto del PG. Al contactar provoca que se actualicen los mapas del cluster del grupo de OSD anterior y por tanto que se detengan las operaciones de lectura o escritura de ese PG. Se detienen directamente si el primer OSD en enterarse es el primario y se detienen durante la operación de actualización de las réplicas si el primero en enterarse es un OSD que almacena un PG réplica.

En el caso de las lecturas, si un OSD sufre un fallo de red que lo haga inalcanzable durante un periodo de tiempo provocará que los procesos de lectura se marquen como fallidos. Mientras

tanto, el mapa se puede actualizar y designar otro OSD en su lugar. Si ese OSD procesa una actualización mientras otros nodos tienen una versión del mapa anterior, podrían llegar al OSD original para procesar una lectura de unos datos que ya han dejado de ser los actuales.
Para evitar que se pueda dar una situación así y que la lectura se lleve a cabo en el OSD nuevo, se utiliza un sistema de mensajes latido (*heartbeat messages*). Esto consiste en que los OSD reciben constantemente pequeños mensajes de las réplicas de los PG que indican que siguen «vivos». Si en *h* segundos no se recibe la señal de alguna réplica, ese PG queda bloqueado y deja de ser legible. Para que otro OSD obtenga el rol de OSD primario para encargarse de las lecturas, debe o bien obtener la aceptación del cambio de roles del OSD que quiere sustituir o bien esperar *h* segundos. El valor actual de *h* es dos. Así se garantiza una detección de errores rápida y un tiempo mínimo de indisponibilidad de los datos en caso de que un OSD principal falle.

3.1.7. Migración de datos y recuperación de errores

Las migraciones de datos se hacen siempre y exclusivamente a causa de actualizaciones del mapa del *cluster* y sus reubicaciones de PG. Estos cambios pueden ser causados por caídas y recuperaciones de los OSD, por expansión o contracción del *cluster* añadiendo o retirando OSD e incluso por un cambio en la política de distribución de réplicas de CRUSH que puede provocar un remodelado completo de los PG.

RADOS emplea un robusto algoritmo basado en la autogestión de cada nodo y su comunicación con sus pares para poder establecer una visión consistente del contenido de cada PG y poder controlar tanto la replicación y migración de datos como la recuperación de errores. Cada OSD mantiene y replica de manera agresiva un registro que contiene lo que cada PG debería contener, como por ejemplo las versiones de los objetos que contiene. Esto lo hace incluso cuando las réplicas de los objetos no se encuentran localmente. Al guardar tan celosamente los metadatos de los PG se facilita la recuperación de errores y se simplifica su algoritmo, y la detección fiable de errores.

3.1.7.1. *Peering*

Cuando un OSD recibe un paquete de actualización del mapa del cluster, examina una por una todas las actualizaciones incrementales que contiene hasta la más reciente. Si en lista de OSD de un PG ha habido alguna modificación, todos los OSD de la lista deben volver a sincronizarse, es decir, a ponerse de acuerdo sobre el estado de todos sus objetos y sus

metadatos. Esto no significa que sincronicen los datos de los objetos en sí. A este proceso se le llama *peering*. Este proceso tiene en cuenta no solo la actualización con la *epoch* más reciente sino todas para evitar que se den casos en los que un OSD desaparece de la lista del PG y vuelve a aparecer (por ejemplo tras una caída momentánea) y mientras estaba fuera se han actualizado los datos del PG. La replicación de los datos y el *peering* son procesos que se llevan a cabo independientemente para cada PG del sistema.

El OSD primario de un PG es quien realiza el *peering*. Cada OSD envía un mensaje de notificación *Notify* por cada réplica de PG que almacena localmente, a sus respectivos OSD primarios. Cada mensaje está compuesto de la actualización más reciente, los límites del registro de PG (PG *log*), y la *epoch* más reciente en la que se hizo correctamente el proceso de *peering* por última vez. A continuación, el OSD primario genera una lista de todos los OSD que han sido responsables del PG desde la última sincronización correcta. Una vez tiene la lista solicita activamente a todos los miembros un mensaje de notificación para conseguir los metadatos del PG. Cuando los tiene, el OSD primario puede determinar cuál es la última actualización que ha recibido cada réplica y solicitar cualquier actualización de la que no disponga a un OSD que era anteriormente responsable del PG. Por último, el OSD primario comparte los fragmentos perdidos del *PG log* con los OSD réplicas para que estos sepan exactamente qué objetos deben contener, aunque todavía no los tengan localmente. Así pueden empezar a procesar operaciones de lectura y escritura mientras comienza la restauración real en segundo plano.

3.1.7.2. Recuperación

Una ventaja crítica del almacenamiento desagrupado de RADOS es la paralelización de la recuperación de errores. Cuando un disco falla en un sistema tradicional hay que replicar toda la información que hubiera en ese disco desde la copia de seguridad que se tenga a otro disco para sustituir al anterior. En el caso de RADOS cuando un disco falla, las réplicas de los PG que contenía se restauran desde un un número de dispositivos tan grande como esté configurada la replicación de cada PG y a tantos OSD como CRUSH determine. En un sistema grande, de media cada OSD implicado en una recuperación de datos tiene que encargarse de enviar o recibir los datos de un solo PG, haciendo el proceso de recuperación rapidísimo.

La recuperación está coordinada por el OSD primario de cada PG. Como el primario sabe qué objetos le faltan a cada réplica gracias al proceso de *peering*, puede enviar preventivamente los objetos que faltan y que van a ser modificados durante la restauración. Así se asegura de que los datos de origen se leen una sola vez. Si el primario tiene que enviar un objeto, o acaba de

leerlo, aprovecha mientras lo tiene en memoria para enviar una copia a todos los OSD réplica que lo necesiten.

3.1.8. Monitores

Un pequeño *cluster* de monitores se encargan de gestionar el sistema de almacenamiento guardando la copia principal (*master copy*) del mapa del *cluster* y modificándola cuando varía la configuración del *cluster* o hay un cambio de estado en un OSD. El *cluster* está diseñado para favorecer la consistencia y la durabilidad antes que la disponibilidad. Está basado parcialmente en el algoritmo Paxos, que es un protocolo que sirve para llegar a consensos en sistemas distribuidos en los que tanto los nodos como la red sobre la que se comunican son propensos a tener fallos.

3.1.8.1. El servicio Paxos

El *cluster* está basado en una máquina de estados distribuida cuyo estado actual es el mapa del *cluster* y cada actualización da lugar a una nueva *epoch*. Se emplea un Paxos simplificado, porque solo se permite que haya una mutación del mapa cada vez y además hay implementado un sistema de préstamos que permite que cualquier monitor del *cluster* pueda atender lecturas y escrituras mientras se garantiza su ordenamiento consistente.

Inicialmente el *cluster* elige a un líder para distribuir las actualizaciones del mapa y para controlar la consistencia. Si la mayoría de los monitores están activos se unen al *quórum* y se inicia la primera fase del algoritmo Paxos, que asegura que todos los monitores van a obtener la última versión del mapa, solicitando las actualizaciones incrementales de otros monitores.

A partir de ahí, se empiezan a repartir los préstamos de permisos a los monitores activos para que puedan distribuir las copias del mapa del *cluster* a los OSD que las soliciten. Si el tiempo de duración del permiso expira sin que se haya renovado, se asume que el líder no está funcionando y se convoca una nueva elección. Los permisos deben ser confirmados cuando se reciben. Si eso no ocurre, el líder asume que el monitor en cuestión no está funcionando y se convoca una nueva elección de *quórum*. Si por cualquier motivo una elección no termina en un tiempo prefijado, se convoca otra.

Cuando un monitor recibe noticias de un OSD indicándole por ejemplo una caída de un OSD y este ya estaba marcado como *down*, el monitor responde al OSD enviándole el mapa del

cluster para actualizarle. Si por el contrario desconocía esta situación, reenvía la notificación al líder del *cluster* de monitores, que añade la nueva información a una actualización incremental. El líder inicia periódicamente una actualización del mapa incrementando su *epoch* y utilizando el algoritmo Paxos para distribuir la propuesta de actualización entre el resto de monitores y anulando al mismo tiempo los permisos de estos. Si la mayoría acepta la propuesta, se incluyen los cambios en el mapa definitivamente y se vuelven a distribuir los permisos.

Este sistema garantiza que todos los permisos de los monitores expiran antes de que tenga lugar una actualización del mapa y, por tanto, que la versión del mapa no pueda retroceder nunca.

3.1.8.2. Carga de trabajo y escalabilidad

El trabajo de los monitores es más bien escaso. El mapa se propaga por el *cluster* por las interacciones entre los propios OSD. Los fallos en el *cluster* son relativamente poco frecuentes. El sistema de permisos temporales permite que cualquier monitor pueda atender las peticiones de los OSD y ocurren muy rara vez debido a la política de enviar el mapa preventivamente. Los clientes solo solicitan el mapa cuando se quedan sin respuesta tras una petición y sospechan entonces que ha podido haber un error. El *cluster* de monitores puede expandirse para sistemas muy grandes, aunque generalmente se hace más por motivos de fiabilidad que de reparto de la carga de trabajo.

Las notificaciones que requieren una actualización del mapa, como por ejemplo el arranque de un OSD o la aparición de un error desconocido para el *cluster* de monitores, se tratan reenviándolas al líder del *cluster*. El líder acumula una cierta cantidad de cambios en una sola actualización y así la frecuencia del envío de las actualizaciones es configurable e independiente del tamaño del *cluster*.

El peor pico de trabajo que puede aparecer se da si falla un número *n* de OSD que contienen *m* PG cada uno. Esto significa que se generarán *nm* mensajes de error, que puede llegar a ser una cifra considerable. Para evitar que se generen tantos mensajes simultáneamente, el sistema de latidos, explicado en el apartado de la propagación del mapa, se hace a intervalos semialeatorios. Posteriormente, los monitores no líderes envían solo una vez cada error al líder del *cluster*, de tal manera que el número máximo que este recibirá será *on* siendo *o* el número de monitores del *cluster*.

3.2. LIBRADOS

Como ya se ha señalado anteriormente, Ceph puede mostrarse como un almacén de objetos, un dispositivo de bloques y un sistema de ficheros, pero no está limitado a usar solo las interfaces RESTful, de bloque o POSIX. Basándose en RADOS, LIBRADOS es una librería que permite acceder nativamente a RADOS. Está disponible en C, C++, Python, Erlang, PHP y JAVA (JNA). Oculta la distribución, migración y replicación de los datos, así como los fallos de los nodos.
LIBRADOS permite al usuario diseñar su propia interfaz para comunicarse con los dos tipos de demonio (*daemon*) del *Cluster* de Almacenamiento Ceph: Ceph Monitor y Ceph OSD Daemon.

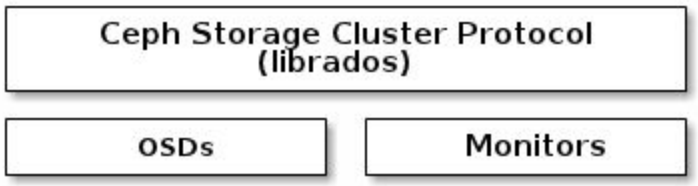


Figura A1.6: Diagrama de LIBRADOS. La librería proporciona acceso al *cluster* de monitores y al de OSD. [14] **(Inktank Storage Inc. 2015, Architecture)**

3.3. *Ceph Object Gateway* (radosgw)

La puerta de entrada de objetos de Ceph (*Ceph Object Gateway*) es una interfaz de almacenamiento de objetos construida a partir de LIBRADOS. Otorga a las aplicaciones una interfaz RESTful al *cluster* de almacenamiento de Ceph. El almacenamiento de objetos de Ceph (*Ceph Object Storage*, compuesto por *Ceph Storage Cluster* y *Ceph Object Gateway*) es compatible con dos APIs: Amazon S3 RESTful y OpenStack Swift.

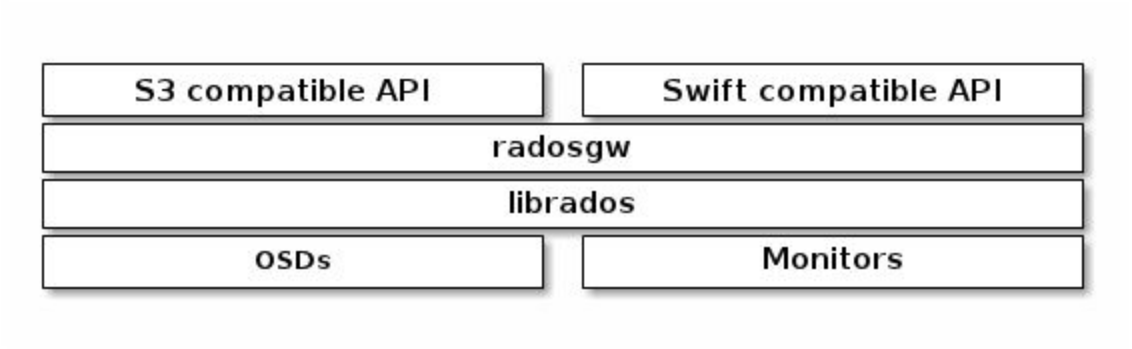


Figura A1.7: Diagrama de radosgw. La puerta de entrada de objetos se puede comunicar con cualquier API compatible con S3 o Swift y hace de intermediario con LIBRADOS. [15] **(Inktank Storage Inc. 2015, Ceph Object Gateway)**

El almacén de objetos de Ceph emplea el demonio de la puerta de entrada de objetos, *radosgw*, para comunicarse con el *cluster* de almacenamiento. La puerta de entrada tiene que gestionar a sus propios usuarios, y puede almacenar sus datos en el mismo *cluster* de almacenamiento donde se guardan los datos de los usuarios del sistema de ficheros de Ceph o del dispositivo de bloques de Ceph. Las APIs de S3 y de Swift comparten un mismo *namespace*, lo que significa que se pueden almacenar datos a través de una API y recuperarlos a través de la otra.

3.4. *Ceph Block Device* (RBD)

El dispositivo de bloques de Ceph le permite presentarse como dicho tipo de dispositivos. Si se le monta así, Ceph emula ser un dispositivo de bloques que al estar construido sobre LIBRADOS se puede beneficiar de las ventajas que esta interfaz ofrece. Los datos se reparten por el *cluster* y se replican como cualquier otra colección de PG. Se puede utilizar el mismo *cluster* simultáneamente para el sistema de ficheros de Ceph y para el almacenamiento de objetos.

Una ventaja clara de este modo es que al dividir los datos almacenados de un dispositivo de bloques virtual en el *cluster* de OSD, el rendimiento a la hora de leer los datos es mucho mayor, ya que hay muchos dispositivos físicos preparados para recuperar los datos en vez de uno.

Otra ventaja es que puede ofrecer más capacidad de almacenamiento en el dispositivo de bloques virtual de la que realmente se tiene (*thin provisioning*).

También se puede aprovechar de la capacidad de LIBRADOS para la toma y recuperación de *snapshots*, lo que es particularmente útil para un dispositivo de bloques. El desarrollador afirma ofrecer un gran rendimiento cuando se emplea con máquinas virtuales basadas en el *kernel*, o KVMs (*Kernel-based Virtual Machines*).

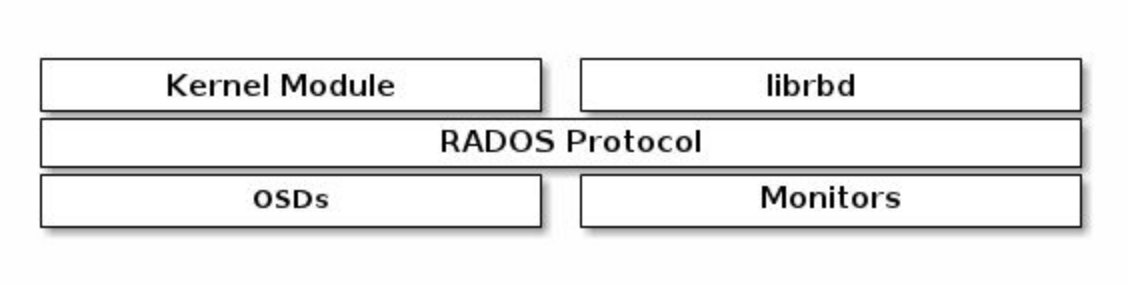


Figura A1.8: Diagrama del dispositivo de bloques. Las dos maneras de presentarse como un dispositivo de bloques son mediante librbd o a través de un módulo del kernel. Ambos métodos dependen de LIBRADOS para comunicarse con el *cluster*. [16] **(Inktank Storage Inc. 2015, Ceph Block Device)**

Los dispositivos de bloques de RADOS, o RBD (RADOS Block Devices), se comunican con el *cluster* mediante módulos propios del *kernel* o mediante la librería librbd.

3.5. Ceph Filesystem

El sistema de ficheros de Ceph, Ceph FS, ofrece una interfaz tradicional con semántica POSIX. Este sistema de ficheros se apoya en el mismo *cluster* de almacenamiento de objetos que la puerta de entrada de objetos y el sistema de almacenamiento de bloques.

Para poder montar este sistema de ficheros es necesario que exista al menos un servidor de metadatos de Ceph (*Ceph Metadata Server*, MDS). Este servidor o *cluster* de servidores proporciona un servicio que relaciona los directorios y los nombres de fichero del sistema de ficheros con los objetos donde están almacenados en el *cluster* de RADOS.

El *cluster* de servidores de metadatos puede expandirse o contraerse y puede reequilibrar la carga de trabajo del sistema distribuyendo los datos entre los distintos OSD como considere óptimo.

Para acceder al Ceph FS, se puede utilizar el propio cliente de Ceph, ya que está incluido en el kernel de Linux. También es posible usar un cliente basado en FUSE (Filesystem in Userspace), que permite ejecutar el código de un sistema de ficheros en modo usuario, sin privilegios.

El sistema de ficheros de Ceph se encontraba en desarrollo en el momento en el que se realizó este estudio, y aunque era posible utilizarlo con normalidad, no había herramientas de comprobación y recuperación de errores en el sistema de ficheros en caso de desastre. En la documentación del sistema se advertía de esto a los posibles usuarios y se les aconsejaba precaución a la hora de almacenar datos importantes.

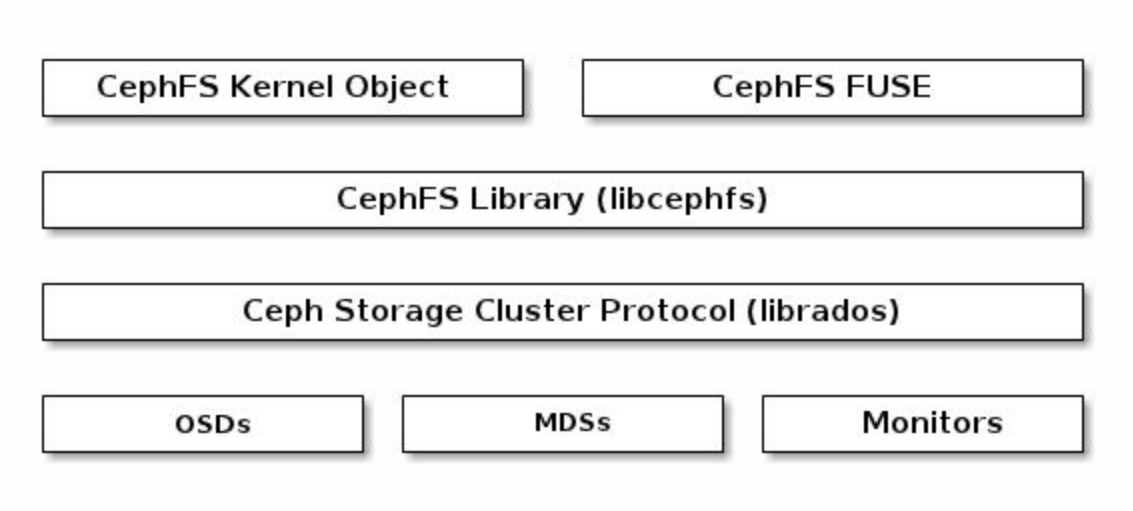


Figura A1.9: Diagrama del uso de Ceph como sistema de ficheros. Para emplear Ceph como un sistema de ficheros es necesario conectarse desde el cliente por medio de FUSE o de un módulo del kernel. En este caso existe una capa intermediaria que es libcephfs y que se comunica con LIBRADOS. En este caso LIBRADOS además de tratar con los OSD y los monitores, también lo hace con los servidores de metadatos. [17] **(Inktank Storage Inc. 2015, Architecture)**

Anexo 2: GlusterFS

1. ¿Qué es GlusterFS?

GlusterFS es un sistema de ficheros distribuido con semántica POSIX, diseñado para funcionar sobre *hardware* heterogéneo y convencional (*commodity hardware*). GlusterFS ofrece la sencillez y las características fundamentales de un NAS al mismo tiempo que la escalabilidad de un sistema complejo y caro como un SAN.
Sus principales características son:

- no requiere un servidor centralizado de metadatos,
- el gasto en infraestructuras es mínimo,
- tiene un alto nivel de redundancia,
- el despliegue es sencillo y barato.

2. Tipos de almacenamiento soportados

GlusterFS es un sistema de ficheros distribuido que se apoya sobre protocolos como NFS v3, NFS v4 o SAMBA. También es capaz de comunicarse con el almacenamiento de objetos de OpenStack (Swift). Conviene remarcar que GlusterFS no puede almacenar objetos por sí mismo, sino que tiene la capacidad de comunicarse con un sistema que lo puede hacer.

Físicamente los datos se almacenan como ficheros en un sistema de ficheros convencional (EXT3, EXT4, XFS). GlusterFS es la capa lógica que aporta la distribución de los datos, redundancia, etc. Un dispositivo de almacenamiento utilizado por GlusterFS puede ser leído por cualquier sistema operativo compatible con esos sistemas de ficheros aunque Gluster no esté presente.

3. Arquitectura

En este apartado se detalla cómo funciona el sistema de ficheros de GlusterFS. Para empezar, se presentan los términos que aparecerán con más frecuencia:

- *Brick* (ladrillo): es un punto de montaje de un sistema de ficheros que se le indica a GlusterFS y que este puede usar para el almacenamiento de datos. Es la unidad de almacenamiento para GlusterFS.
- Traductor: es una capa de *software* que se encarga de presentar los ladrillos de los que se compone el *cluster* como parte del *namespace* global. Se pueden apilar todos los traductores que sean necesarios.

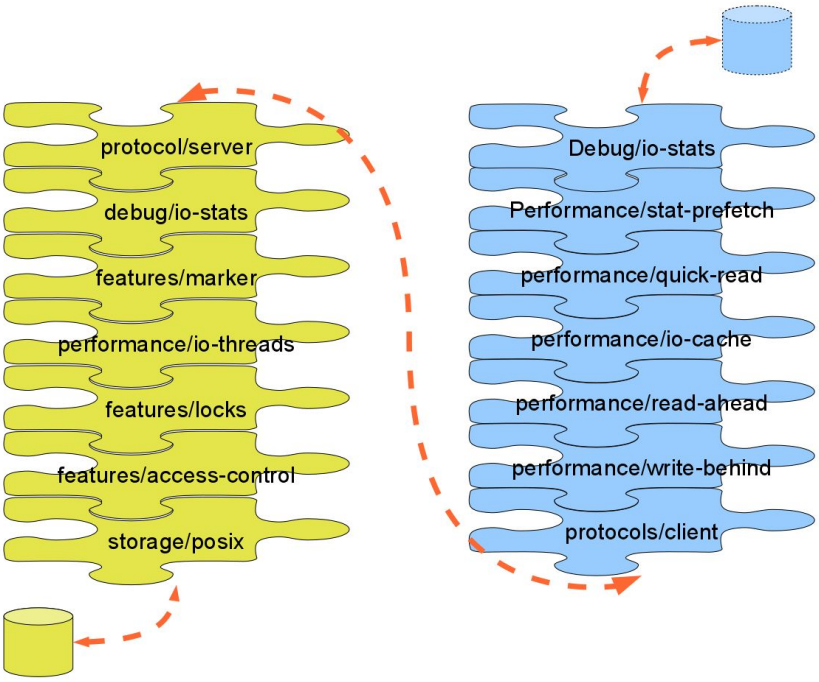


Figura A2.1: Ejemplo de pila de traductores de GlusterFS. [18] (Black 2014)

- Volumen: lo que se le presenta al usuario final como almacenamiento disponible. Se trata de *bricks* combinados y pasados por un traductor.
- Nodo: máquina en la que se ejecuta un servicio de GlusterFS y que comparte los datos con los clientes.

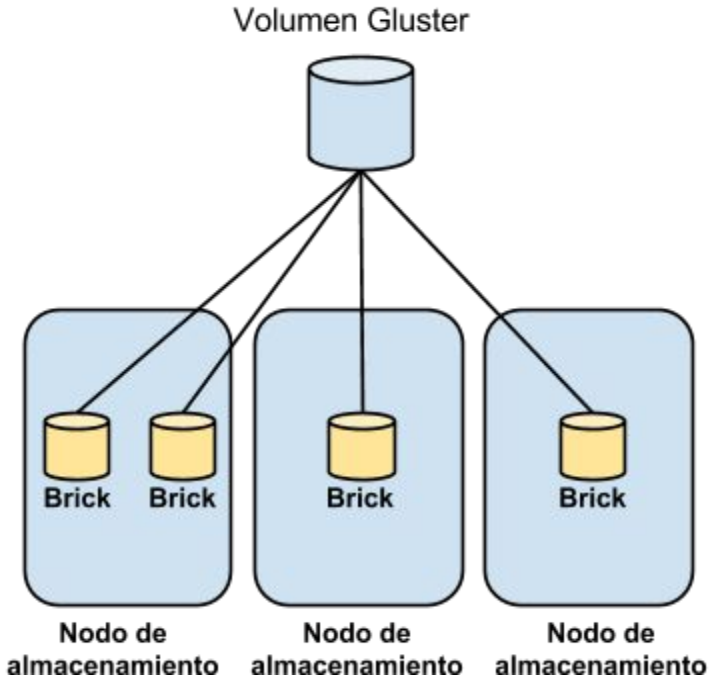


Figura A2.2: Ejemplo de un volumen GlusterFS. En este esquema el *cluster* está compuesto por tres nodos y uno de ellos tiene más de un *brick*.

- Elasticidad: es la capacidad de un sistema para adaptarse a las necesidades reales del entorno donde se utiliza. Esto significa que debe ser capaz de crecer o decrecer sin que ello impacte negativamente en ningún aspecto de su funcionamiento.
- Crecimiento lineal: significa que el rendimiento y la capacidad de almacenamiento del *cluster* debe aumentar en la misma proporción en que lo hace la cantidad de nodos del sistema. Esta es una característica que no siempre se cumple en otros sistemas, ya que hay muchos factores de los que depende el rendimiento, tales como la velocidad de la CPU, la velocidad de la red, la capacidad máxima soportada por el sistema de ficheros, etcétera.
- Crecimiento logarítmico: es el tipo de crecimiento que suelen mostrar los sistemas tradicionales de almacenamiento. Cuantos más nodos se añaden, menos beneficio se obtiene de cada nuevo nodo.
- Agrupación de almacenamiento (*storage pool*): los servidores de almacenamiento conectados en red entre sí.

Un servidor Gluster puede tener conectado un dispositivo de almacenamiento (*Direct Attached Storage*, DAS), varios (*Just a Bunch Of Disks*, JBOD) o como recomienda RedHat, el propietario de Gluster, un RAID 6, aunque la redundancia ya la gestione Gluster a nivel de *software*. Es necesario que el sistema de ficheros que vaya a funcionar con GlusterFS tenga soporte para atributos ampliados, porque es ahí donde se van a almacenar los metadatos relevantes. Algunos sistemas de ficheros válidos son XFS (recomendado), EXT3, EXT4 y BTRFS.

Para acceder a los datos, se pueden utilizar distintos sistemas:

- El cliente nativo de Gluster (*Gluster Native Client*), que funciona a través de FUSE.
- NFS. Gluster tiene incorporado NFS. Los servicios de Gluster ejecutan un servidor NFS v3.
- Samba. Es posible acceder a través de Samba, pero para ello es necesario tener un entorno configurado independiente de Gluster. Este acceso se realiza indirectamente a través del cliente nativo de Gluster.
- Archivo y objeto unificados (*Unified File and Object*, UFO). Se trata de una capa lógica que permite el acceso a los datos simultáneamente como objetos y como ficheros.

Cada nodo de Gluster tiene los siguientes componentes:

- glusterd: es el servicio principal que gestiona las comunicaciones y el volumen. Para comunicarse con este demonio se utiliza la interfaz de línea de comandos de Gluster (Gluster *Command Line Interface* o Gluster CLI).
- glusterfsd: es el demonio encargado de gestionar cada *brick*. Se encuentra una instancia en ejecución por cada *brick* que haya en una máquina. Glusterd gestiona a todos los glusterfsd.
- glusterfs: es un proceso que se encuentra tanto en clientes como en servidores. Proporciona dos servicios fundamentales; en el lado del cliente, gestiona las comunicaciones con FUSE y en el lado del servidor es quien ejecuta el servidor NFS.

En GlusterFS se entiende la escalabilidad de dos maneras; aumentando la capacidad de almacenamiento de un nodo o añadiendo otro nodo al cluster. En el primer caso, se gana solamente capacidad de almacenamiento, mientras que en el segundo caso se gana además un mayor rendimiento del sistema.

En la práctica, Gluster puede crecer linealmente. Esto es posible gracias al empleo de tres técnicas: la eliminación de un servidor central de metadatos, una distribución efectiva de los

datos para obtener fiabilidad y escalabilidad, el uso del paralelismo para obtener el máximo rendimiento en un entorno completamente distribuido.

3.1. Gestión de los metadatos

En un sistema escalable de estas características, uno de los mayores desafíos consiste en saber cuál es la ubicación física y lógica de cada fichero (metadatos de localización).
Hay dos maneras de solucionar este problema; centralizando la información de los metadatos en un índice o distribuyendo ese índice a través de muchos servidores.

3.1.1. Servidor de metadatos centralizado

Muchos sistemas distribuidos optan por almacenar esta información en un índice centralizado. Inmediatamente esta solución crea otros dos nuevos problemas: genera un SPOF (*single point of failure*) al mismo tiempo que un potencial cuello de botella. Si además las características del entorno exigen el almacenamiento de muchos ficheros de poco tamaño, la relación cantidad de metadatos / cantidad de datos almacenados aumenta y hace que los problemas de rendimiento asociados a un índice centralizado se manifiesten antes.

Si el índice no solo almacena metadatos de localización sino que almacena todo tipo de metadatos, como es práctica habitual, el problema del cuello de botella se acentúa. Por ejemplo, cada vez que se accede a un fichero para su lectura, es necesario actualizar el *timestamp* del fichero y esto implica una operación de escritura en los metadatos. A medida que aumenta el número de ficheros, estas operaciones se hacen más frecuentes.

Aunque el problema más grave es la aparición de un SPOF en el sistema. Si el servidor centralizado de metadatos pierde la conexión, cesan todas las operaciones del sistema de almacenamiento. Lo que es más grave, si tiene un problema de corrupción de datos hay que parar el sistema para realizar un lento proceso de recuperación (FSCK) y en el peor de los casos, los datos pueden quedar irrecuperables si no se puede reparar el índice.

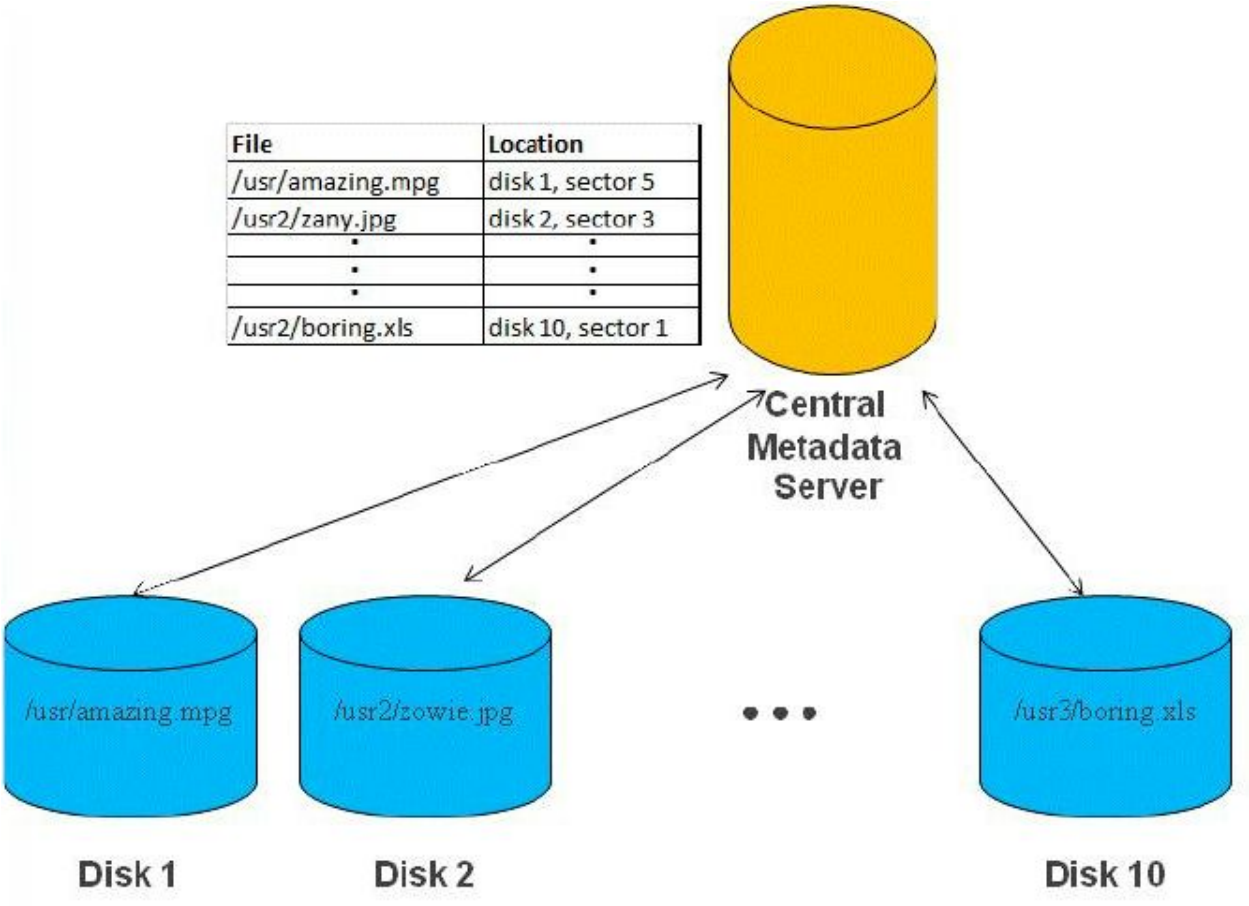


Figura A2.3: Esquema de una gestión centralizada de metadatos. Los metadatos se almacenan en un único servidor que se encarga de responder a todas las peticiones. [19] **(Gluster Inc. 2011, p. 13)**

3.1.2. Sistemas de metadatos distribuidos

En este tipo de modelo, el índice de metadatos se distribuye por un gran número de sistemas de almacenamiento. Esta aproximación no está exenta de problemas.

El primero de los problemas que aparecen es un gran coste en recursos debido a la necesidad de que los metadatos se mantengan sincronizados. Para esto se emplean técnicas de bloqueo y sincronización. En cuanto a rendimiento, tanto los índices centralizados como los distribuidos padecen el mismo problema según aumenta el número de ficheros, las operaciones con los

ficheros, los sistemas de almacenamiento, los nodos o cuando disminuye el tamaño de los ficheros que hay que almacenar.

El segundo gran problema que afronta este modelo es la corrupción de los datos. Lo primero que se debe señalar es que en cuantos más sitios se encuentre el índice, más posibilidades hay de que en algún sitio aparezca un problema de corrupción. Aunque la distribución del índice elimine el SPOF, puede dar lugar otros problemas de concurrencia en los metadatos, como por ejemplo que un fichero se actualice en más de una ubicación distinta al mismo tiempo de manera que luego no haya acuerdo entre las instancias.

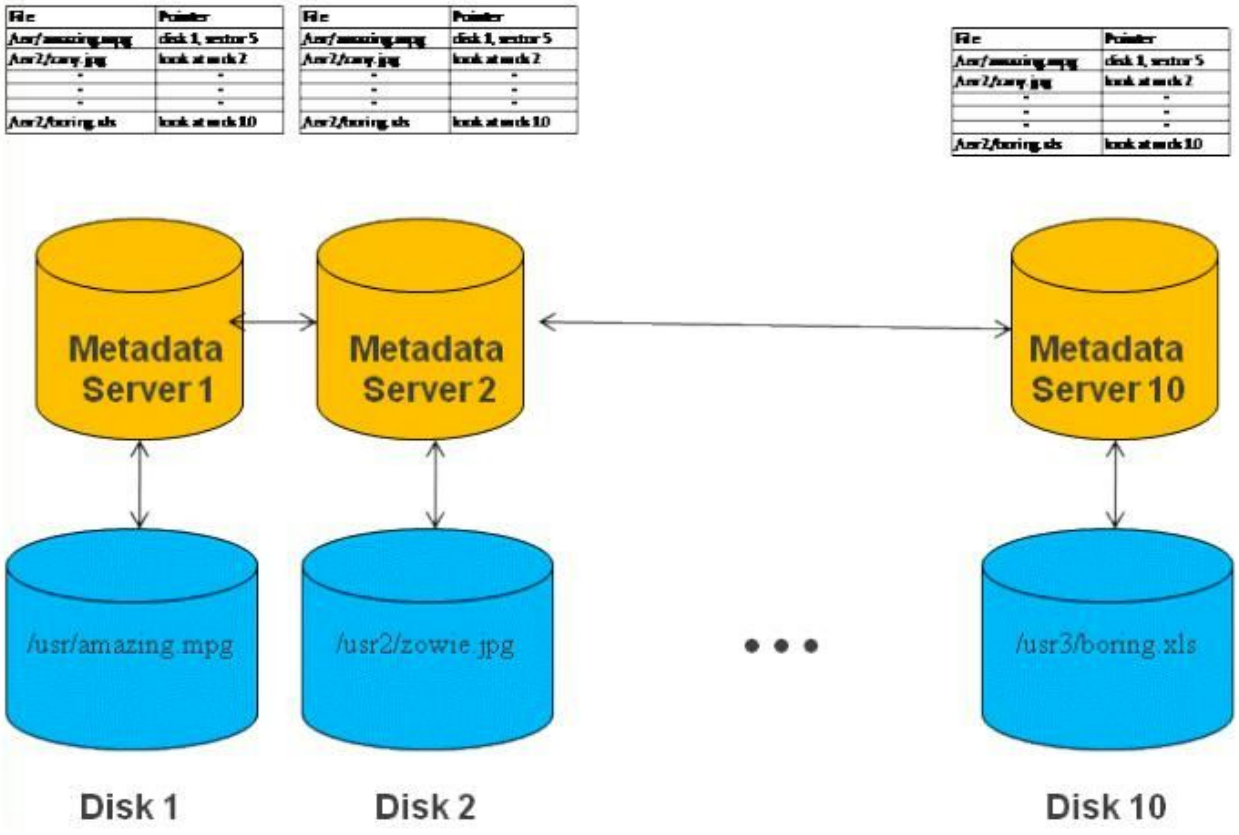


Figura A2.4: Esquema de una gestión distribuida de metadatos. Los metadatos se almacenan en un *cluster* de servidores. [19] **(Gluster Inc. 2011, p. 14)**

3.1.3. Distribución algorítmica

Para evitar los problemas que genera un índice de metadatos centralizado o un índice distribuido, Gluster ubica los metadatos junto con los archivos algorítmicamente. Esta opción no separa los datos de los metadatos de localización. Así, cualquier servidor tiene la inteligencia necesaria para saber en todo momento dónde se encuentra cada fichero sin tener que mirar en un índice o sin preguntar a otro servidor. Un servidor puede deducir dónde está un fichero si se le da la ruta y su nombre. Este sistema paraleliza completamente el acceso a los datos y garantiza un crecimiento lineal del rendimiento, ya que al no crear un cuello de botella con un índice centralizado, no es necesario incrementar la capacidad del servidor del índice a medida que crece el *cluster* añadiendo nuevos *bricks*.

El algoritmo responsable de la ubicación de los ficheros se llama «Algoritmo de resumen elástico» (*Elastic Hash Algorithm*). Las principales ventajas que aporta dicho algoritmo son:

- Calcular la dirección de un fichero de manera algorítmica es mucho más rápido que hacer una consulta a un índice.
- En sistemas grandes, nunca hay contiendas a la hora de acceder a una única instancia de metadatos.
- Gluster consigue presentar crecimiento lineal real, gracias a que cada nodo calcula algorítmicamente las ubicaciones que necesita y así no es necesario mantener un sistema de sincronización de metadatos.
- El enfoque algorítmico es más seguro en entornos de gran cantidad de datos ya que elimina todos los posibles riesgos derivados de un fallo de sincronización de los metadatos.

3.2. *Elastic Hash Algorithm*

Las ventajas de un reparto algorítmico ya han sido mencionadas. A continuación se explica cómo funciona una distribución de estas características y el porqué del nombre del algoritmo.

Una distribución algorítmica, como ya se ha señalado anteriormente, tiene como principal objetivo que cualquier elemento del sistema que recibe la ruta y nombre de un fichero, pueda calcular dónde está almacenado. Por ejemplo, una ordenación alfabética; tiene una serie de normas conocidas y cualquiera sabe que para buscar un fichero que empieza por *c* tiene que ir a buscar entre la *b* y la *d*.

El siguiente problema con el que hay que lidiar es que en un sistema de almacenamiento distribuido se desea que todos los miembros tengan una carga de trabajo semejante. En el ejemplo de la ordenación alfabética, hay letras que son mucho más comunes que otras y eso provocaría puntos calientes (*hotspots*) o zonas con un uso mucho más intenso que el resto. Es necesario uniformar de alguna manera el reparto. Para esto se emplean funciones resumen o *hash*. Estas funciones son deterministas y la distribución de sus resultados tiende a ser matemáticamente uniforme. Concretamente la función hash implementada en el *Elastic Hash Algorithm* está basada en un cifrador de bloque y es la Davies-Meyer: $H_i = E_{mi}(H_{i-1}) \oplus H_{i-1}$

Una vez que se tiene una distribución uniforme, pueden seguir apareciendo puntos calientes en el sistema debido por ejemplo a que la demanda de un determinado fichero es muy alta por algún motivo. Otro inconveniente es cómo hacer que el conjunto de todos los posibles valores que puede dar la función *hash* se adapte al conjunto de los *bricks* disponibles y también debe adaptarse a un número de *bricks* cambiante, ya que se pueden añadir y retirar dispositivos de almacenamiento y nodos al *cluster*.

Por este motivo al algoritmo se le llama elástico, porque tiene que funcionar para un número de *bricks* variable. Para resolverlo, el algoritmo gestiona un elevado número de volúmenes virtuales. Utiliza el algoritmo de *hash* para asignar los ficheros a los volúmenes virtuales y posteriormente asigna los volúmenes virtuales a múltiples dispositivos físicos usando un proceso independiente.

Los volúmenes virtuales están completamente abstraídos del *hardware* real que tienen por debajo y pueden crecer, encoger o migrar según sea necesario. Al incrementar la capacidad de almacenamiento total del sistema, los datos se recolocan automáticamente buscando un equilibrio. Durante este proceso, los datos están siempre disponibles. Se pueden hacer cambios en la configuración del sistema de ficheros en tiempo de ejecución. Estos cambios se propagan por el *cluster* dinámicamente.

Cuando se renombra un fichero, cambia la salida de la función *hash*. Esto hace que cambie el volumen lógico en el que se encuentra y puede que cambie el dispositivo de almacenamiento físico. Cuando esto ocurre, Gluster crea un puntero en la nueva ubicación lógica inmediatamente. Si algún cliente solicita el fichero con su nuevo nombre, se le dirige al nuevo volumen lógico donde encuentra el puntero y allí el puntero le redirige a su vez al viejo volumen lógico, que contiene realmente el fichero. Lo mismo pasa cuando hay que reasignar ficheros por motivos de rendimiento, cuando se crea un punto caliente o el rendimiento de un disco se

degrada excesivamente. Esto es así porque las operaciones de reescritura de ficheros no se hacen a tiempo real, sino que son un proceso que se lleva a cabo en segundo plano cuando se puede. La recolocación de un fichero grande puede ser una operación costosa que puede repercutir negativamente en el rendimiento del sistema.

Gluster puede replicar los volúmenes tantas veces como se le indique en su configuración, usando escrituras síncronas. Siempre se recomienda que se repliquen los datos dos o tres veces. Además de protección contra errores, se aumenta el rendimiento al hacer las lecturas distribuidas a través de todo el conjunto de réplicas de un mismo volumen (*mirror*). El fallo de una determinada réplica es completamente transparente para el usuario.

3.3. Tipos de volúmenes

Existen varios tipos de volúmenes en GlusterFS. A continuación se exponen sus principales características.

3.3.1. Volumen distribuido

Es el tipo de volumen que se crea por defecto en caso de no especificar otro. Cada fichero en este volumen se encuentra almacenado en un único *brick*, sin redundancia. Esta configuración no protege contra fallos que pueden originar pérdidas de datos. Se debe apoyar sobre un *hardware* que tenga implementados los mecanismos de protección adecuados.

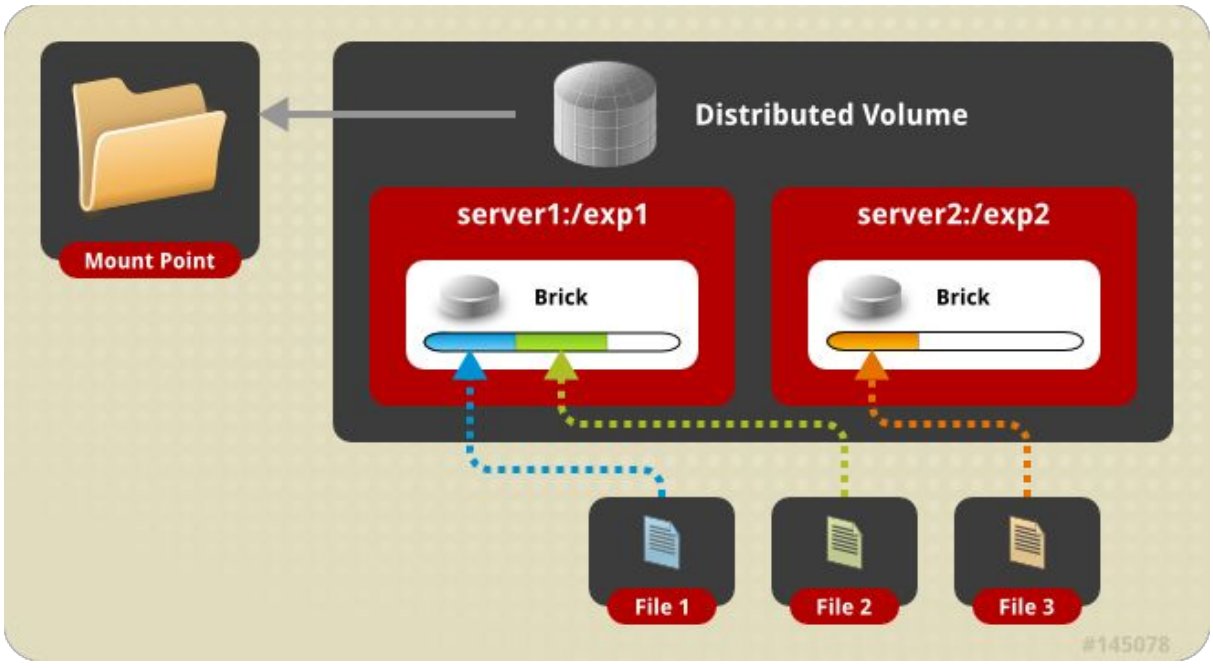


Figura A2.5: Volumen distribuido de GlusterFS. Cada archivo se almacena en un *brick* sin replicación. [20] **(Gluster Inc. 2015, Architecture)**

3.3.2. Volumen replicado

En este tipo de volumen cada *brick* tiene al menos una copia exacta en otro *brick*. Así se soluciona el problema de la ausencia de redundancia y por tanto a la exposición al riesgo de pérdida de datos. El número de copias es configurable.

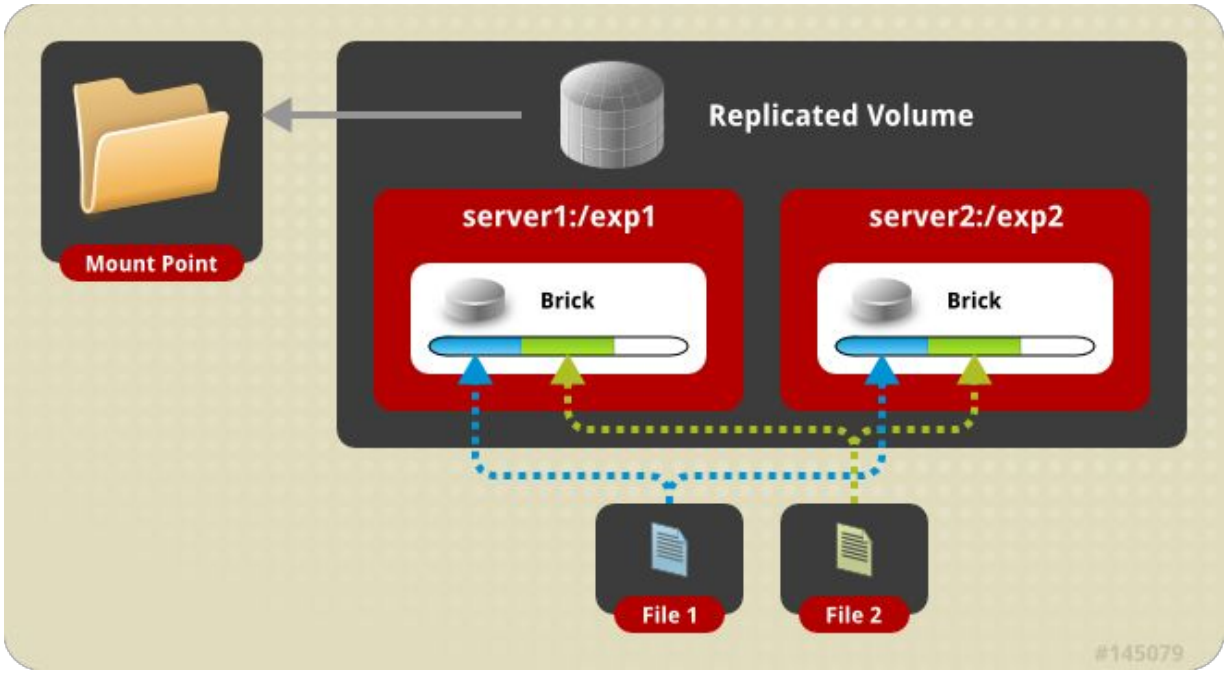


Figura A2.6: Volumen replicado de GlusterFS. Cada archivo se almacena en tantos *bricks* como se haya configurado. [21] **(Gluster Inc. 2015, Architecture)**

3.3.3. Volumen distribuido replicado

En este modelo los ficheros se distribuyen en grupos de *bricks* que se replican unos a otros. Está especialmente indicado para entornos en los que se necesita una elevada disponibilidad de los datos. También se reduce el riesgo de pérdida de datos, ya que si falla un *brick* quedan comprometidos solo los datos que almacena su grupo de *bricks*, no a los demás.
Estos volúmenes se identifican de la forma AxB, donde A es el número de grupos de *bricks* y B el número de *bricks* que compone cada grupo.

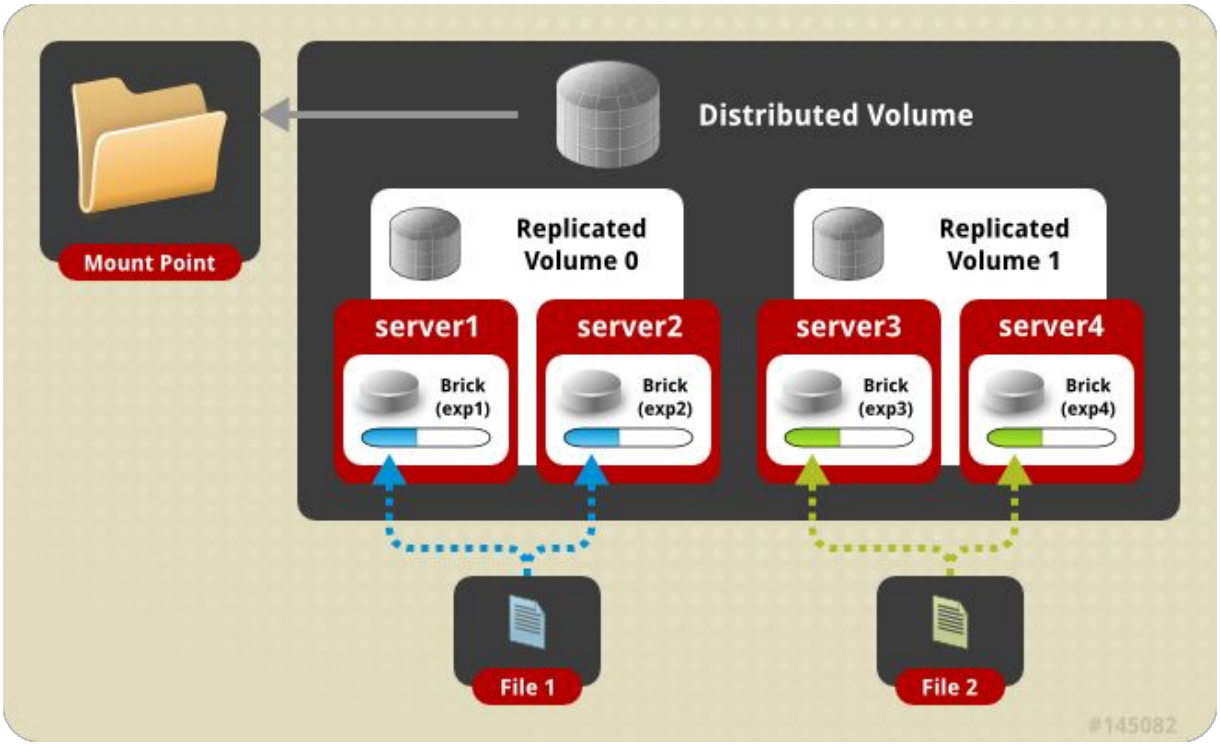


Figura A2.7: Volumen distribuido replicado de GlusterFS. Se crean grupos de bricks espejo y se denominan volumen replicado. [22] **(Gluster Inc. 2015, Architecture)**

3.3.4. Volumen fraccionado

Las peculiaridades de este volumen son que tiene un solo grupo de *bricks* y que los ficheros se fraccionan en tantos trozos como *bricks* haya para almacenar un fragmento en cada *brick*. De esta manera se mejora mucho el rendimiento del sistema si se tienen que almacenar grandes ficheros que tengan una elevada demanda. La desventaja es que no se permite redundancia en este tipo de volumen.

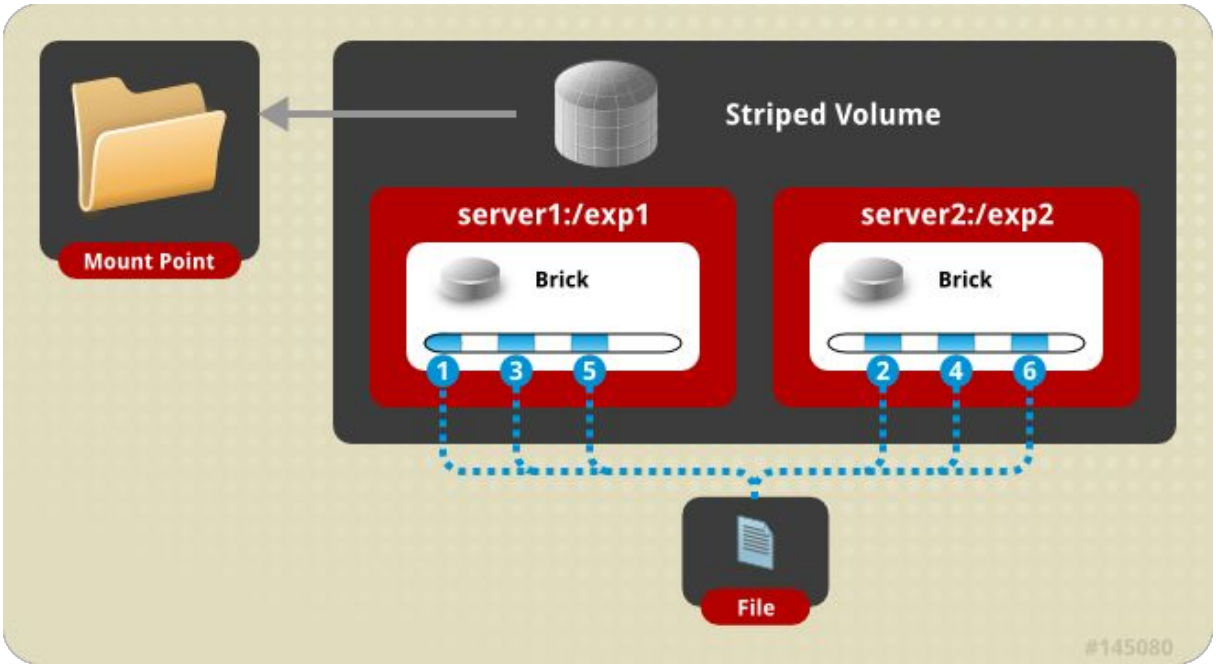


Figura A2.8: Volumen fraccionado de GlusterFS. Cada fichero almacenado en el *cluster* se fragmenta y estos fragmentos se reparten entre los *bricks* que haya. No hay redundancia. [23] **(Gluster Inc. 2015, Architecture)**

3.3.5. Volumen fraccionado distribuido

Similar al tipo anterior, salvo porque existe más de un grupo de *bricks*.

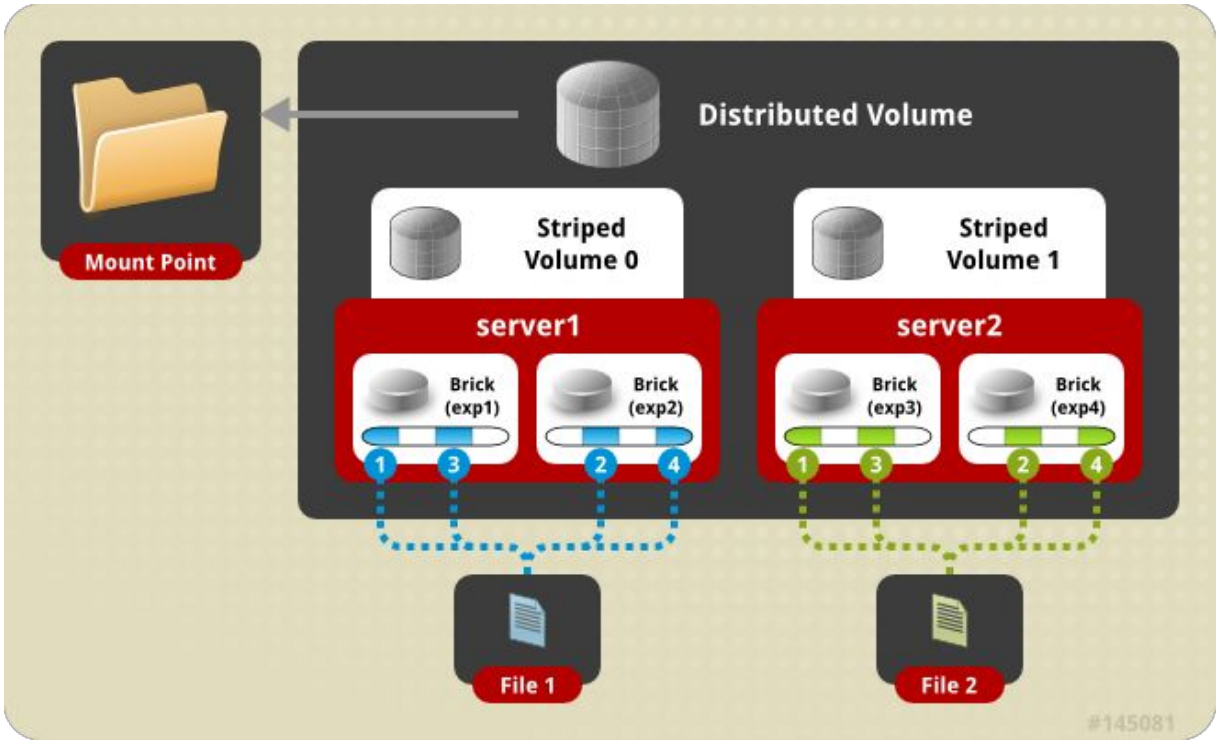


Figura A2.9: Volumen fraccionado distribuido de GlusterFS. En este caso los ficheros se trocean y se reparten entre los *bricks* de un volumen fragmentado. [24] **(Gluster Inc. 2015, Architecture)**

Anexo 3: LizardFS

1. ¿Qué es LizardFS?

LizardFS es un sistema de ficheros distribuido que deriva de MooseFS. Se ha diseñado para ser fiable, escalable, para tener una alta disponibilidad, ser seguro y eficiente.

Está preparado para funcionar en *hardware* convencional, puede añadir y retirar nodos de almacenamiento sin necesidad de dejar de dar servicio, gestiona los fallos de *hardware* de manera automática, dispone de una interfaz web para la gestión, tiene una función de papelera de reciclaje, es posible limitar el número de entradas y salidas por usuario y tiene la posibilidad de hacer *snapshots*. El rendimiento del sistema crece linealmente a medida que se añaden nodos al *cluster*. Utiliza FUSE para su montaje en el cliente.

Se basa en una tecnología solvente y asentada en el panorama de los sistemas de ficheros distribuidos. El único inconveniente que tiene es la escasísima documentación de su funcionamiento a bajo nivel.

Funciona sobre Ubuntu LTS, RHEL, CentOS y Debian, entre otros. Lizard dispone de un cliente para windows con instalador.

2. Tipos de almacenamiento soportados

Lizard almacena únicamente ficheros pero es capaz de manejar ficheros especiales como dispositivos de bloque o archivos de dispositivo.

3. Arquitectura

Para comprender cómo funciona LizardFS primero es necesario presentar a todas las entidades que forman parte de un *cluster* de Lizard:

- *Chunkservers*: Son los servidores donde Lizard almacena los datos de los ficheros que los clientes guardan en el sistema. Cada uno de estos nodos puede tener más de 16 discos duros. No es necesario que tengan un hardware especialmente potente en términos de RAM o CPU para funcionar. Lizard trocea los ficheros que se quieren guardar en el sistema en pedazos (*chunks*) de hasta 64 MB. Es necesario contar con al menos dos *chunkservers* para que haya redundancia.

- *Master Servers*: Son los servidores que gestionan el *cluster*. Son necesarios al menos dos y funcionan en modo maestro-esclavo. Se recomienda que estén instalados en hardware potente. Se estima que necesitan 300 MB de RAM por cada millón de ficheros almacenados en los *chunkservers*. El servidor esclavo recibe el nombre de *Shadow*.
- *Metaloggers*: Son máquinas que almacenan una copia de seguridad de los metadatos de los *Master Servers*. Están pensados para aliviar el trabajo de estos y evitar que necesiten máquinas aún más potentes para funcionar.
- Clientes: Son los que montan el sistema de ficheros de LizardFS para acceder a los datos.

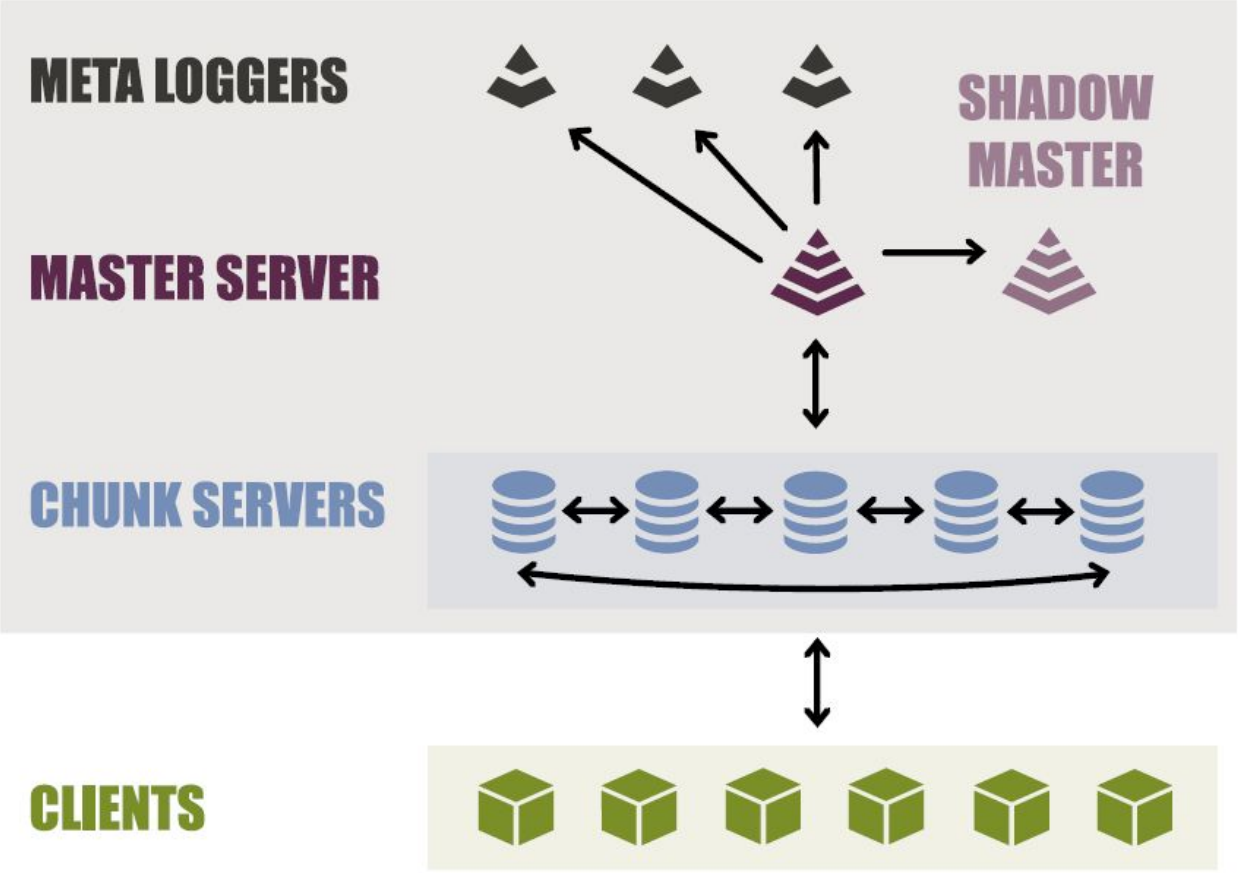


Figura A3.1: Principales entidades de un *cluster* de LizardFS. [25] **(Skytechnology 2015, p. 3)**

3.1. Gestión de los metadatos

Los metadatos que almacena y maneja LizardFS son: nombres de ficheros y directorios, los atributos POSIX estándar, atributos extendidos, ACL (listas de control de acceso), información sobre el nivel de replicación e información sobre los fragmentos del fichero.

Todos los metadatos se almacenan en el *Master Server*, que es un servidor de metadatos y que controla toda la instalación. Los demás servidores de metadatos del *cluster* funcionan como respaldo, manteniéndose sincronizados con el *Master Server* y reciben el nombre de *Shadow Masters*. Todos los cambios que se hacen en el principal son casi inmediatamente replicados en los servidores esclavos o *Shadow*. Estos servidores secundarios se pueden retirar o añadir en cualquier momento de manera transparente para el usuario.

El propósito principal de los *Shadow Masters* es dotar a LizardFS de una alta disponibilidad. Cuando un *Master Server* falla, inmediatamente uno de los *Shadow* toma el papel de principal y todo el *cluster* comienza a trabajar con él. Cuando es necesario hacer alguna modificación que requiera la parada del servidor principal, se puede iniciar este proceso manualmente.

Los Meta Loggers son solo copias de los metadatos del servidor principal. Reciben la misma información de sincronización que los servidores Shadow pero por sí mismos no pueden funcionar como Master, al contrario que los Shadow. Son una versión aligerada de los servidores de metadatos. El objetivo de tener Meta Loggers es poder restaurar el sistema en caso de desastre. Para necesitar una restauración así tendrían que destruirse o corromperse todos los metadatos de todos los servidores de metadatos, Master y Shadow simultáneamente. Este caso puede llegar a ser tremendamente improbable si la distribución de las máquinas del *cluster* es adecuada.

3.2. Replicación de los datos

LizardFS permite configurar distintos perfiles de replicación (*goals*). Cada archivo o carpeta puede tener asignado su propio nivel de replicación. Estos niveles indican cuántas veces se debe replicar el archivo en el *cluster* y opcionalmente, en qué servidores concretos. Nunca se almacenan dos copias de un fichero en el mismo *chunkserver*.

Los datos se distribuyen uniformemente por todo el *cluster*, no existen servidores dedicados a ser espejo de otros servidores. Los archivos se fragmentan como ya se ha explicado anteriormente y cada fragmento y sus réplicas son enviadas a *chunkservers* aleatorios.

Si un *chunkserver* falla o no está accesible, se replican los fragmentos que contenía para mantener siempre activas el número de copias que el perfil de almacenamiento determine.

Existen dos maneras de replicar los datos: la normal y la XOR.
En el primer caso se trata simplemente de copiar un fragmento el número de veces que tenga asignado su fichero. En el caso de la replicación XOR, existe un parámetro que configura el nivel de este tipo de replicación (xor level) y que sirve para fragmentar un *chunk* en tantos trozos como indique más uno. El fragmento adicional sirve para controles de paridad. Si se pierde uno de esos fragmentos es posible recuperarlo gracias al fragmento de paridad.

3.3. Papelera

Cada vez que se elimina un archivo almacenado en el sistema, se mueve a una ubicación especial llamada papelera, donde permanece un cierto tiempo que viene determinado por un parámetro configurable.

Anexo 4: Creación de una máquina virtual Ubuntu 14.04.2 LTE en VMware Workstation 11

Los pasos que hay que seguir para crear un nodo son los siguientes:

- Arrancar VMware y pulsar en «Create a New Virtual Machine».

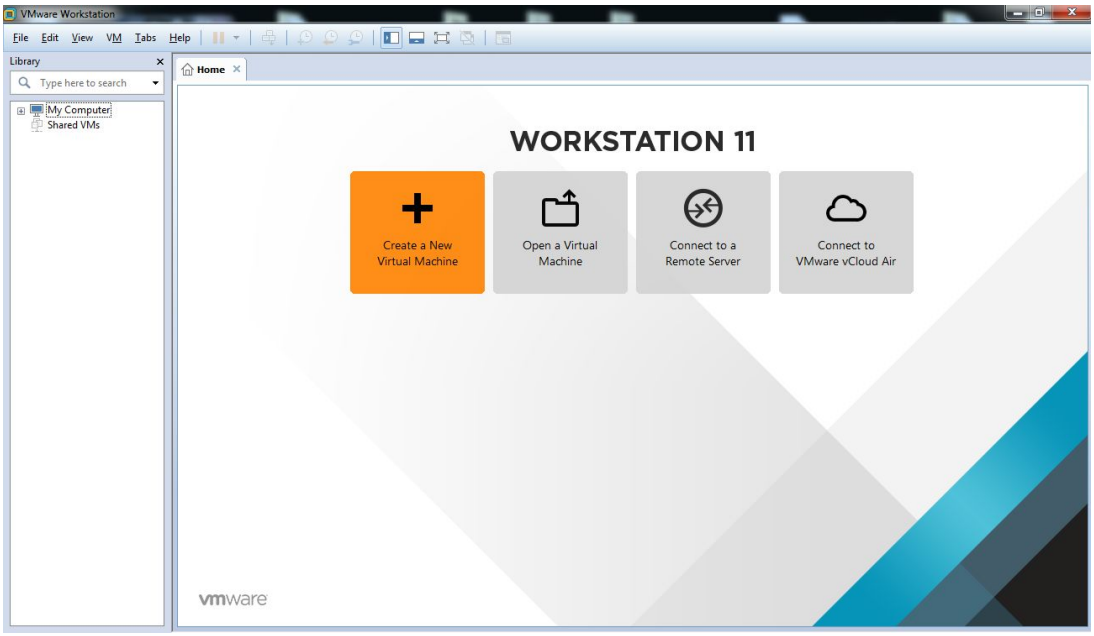


Figura A4.1: Pestaña inicial de VMware Workstation 11.

- Marcar «Custom» y hacer *click* en «Next».

Figura A4.2: Inicio del asistente para crear una nueva máquina virtual en VMware Workstation 11.

- *Click* en «Next» en la siguiente ventana. Así la máquina virtual contará con las características de la última versión de VMware sin importar la compatibilidad con versiones anteriores.
- Seleccionar la imagen del sistema operativo que se quiere instalar.

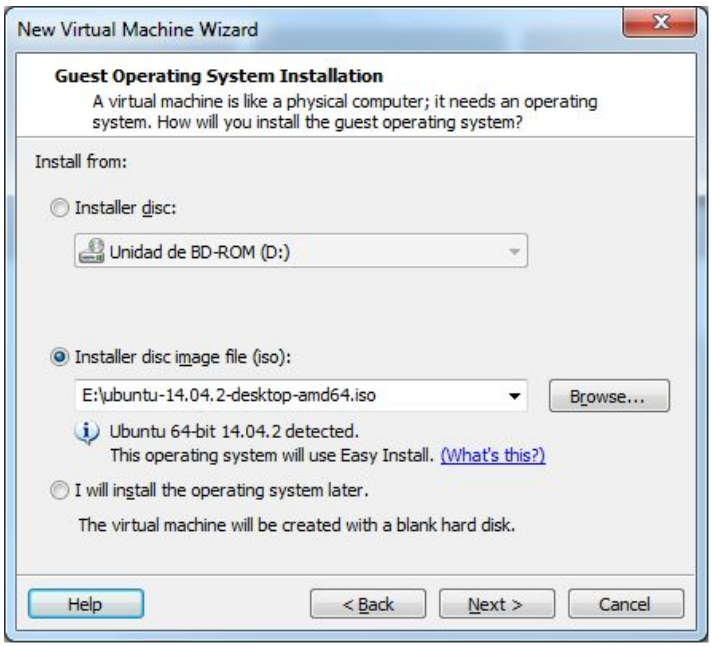


Figura A4.3: Creación de una máquina virtual desde una imagen ISO.

- Se introduce el nombre de la máquina, el nombre del administrador y la contraseña.
- Después se selecciona el nombre de la máquina virtual para VMware y su ubicación.
- Se selecciona el número de procesadores y de núcleos.
- Se fija la cantidad de memoria de la que puede disponer la máquina. VMware sugiere una determinada cantidad según qué sistema se vaya a instalar.

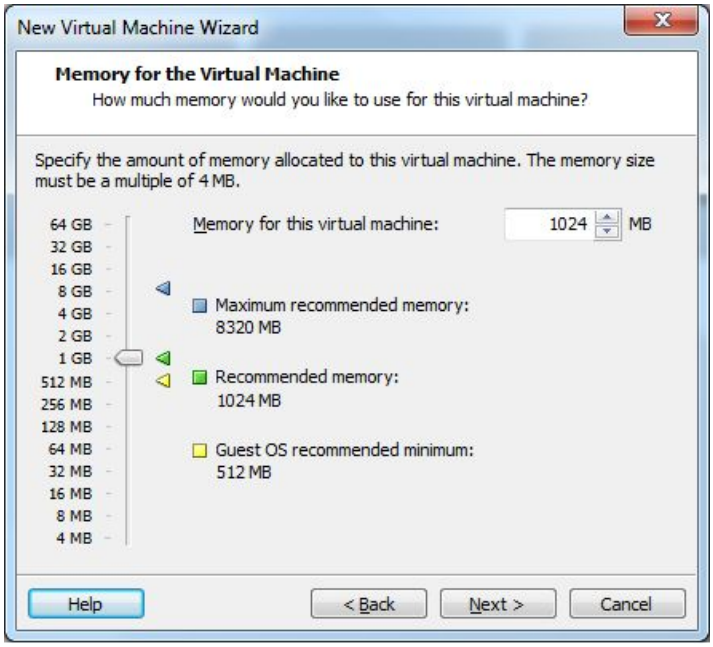


Figura A4.4: Configurar cantidad de memoria para la máquina virtual.

- Se configura la red a la que se quiere conectar la máquina virtual. Se puede escoger cualquier opción y configurarlo más adelante si se desea. Lo mejor es escoger la opción «Host Only».
- A continuación se elige la opción por defecto para el tipo de controladora SCSI.
- Se deja seleccionada la opción recomendada, SCSI.
- Se elige «Create a new virtual disk».
- Se selecciona el tamaño del disco duro virtual y si se quiere fraccionado o completo.

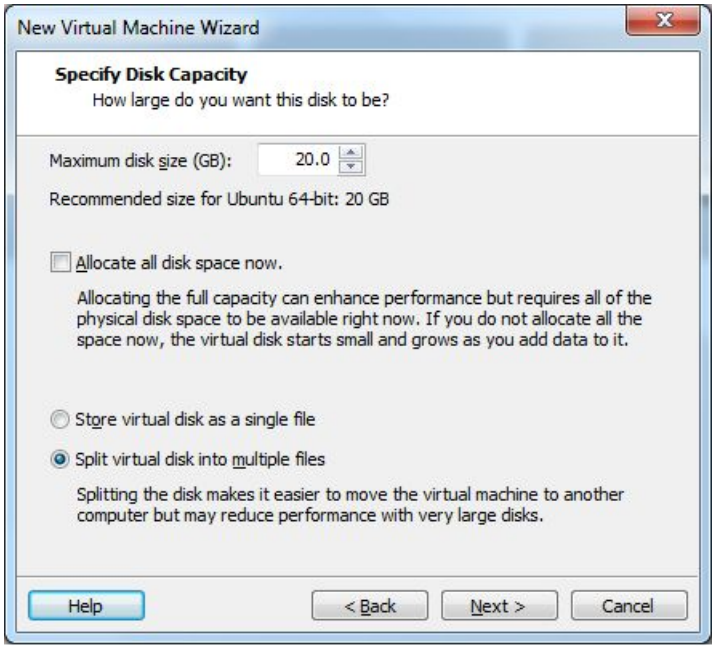


Figura A4.5: Configurar tamaño del disco duro y la creación del mismo en ficheros separados.

- Se introduce el nombre del fichero del disco duro virtual.
- Por último se revisa que todo esté correcto y se hace *click* en «Finish».

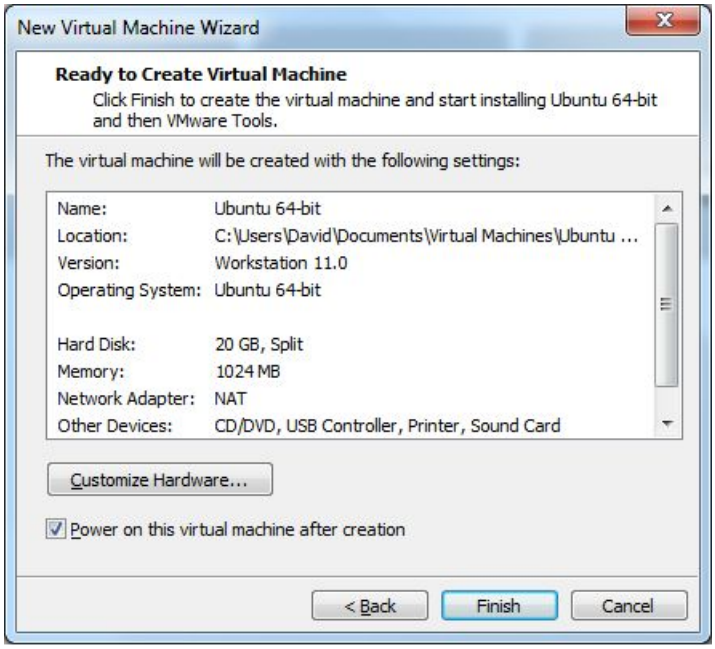


Figura A4.6: Ventana resumen con todos los datos de la máquina virtual.

Una vez que se tienen todos los nodos, se configuran sus direcciones IP y se conectan todos a la misma red virtual.

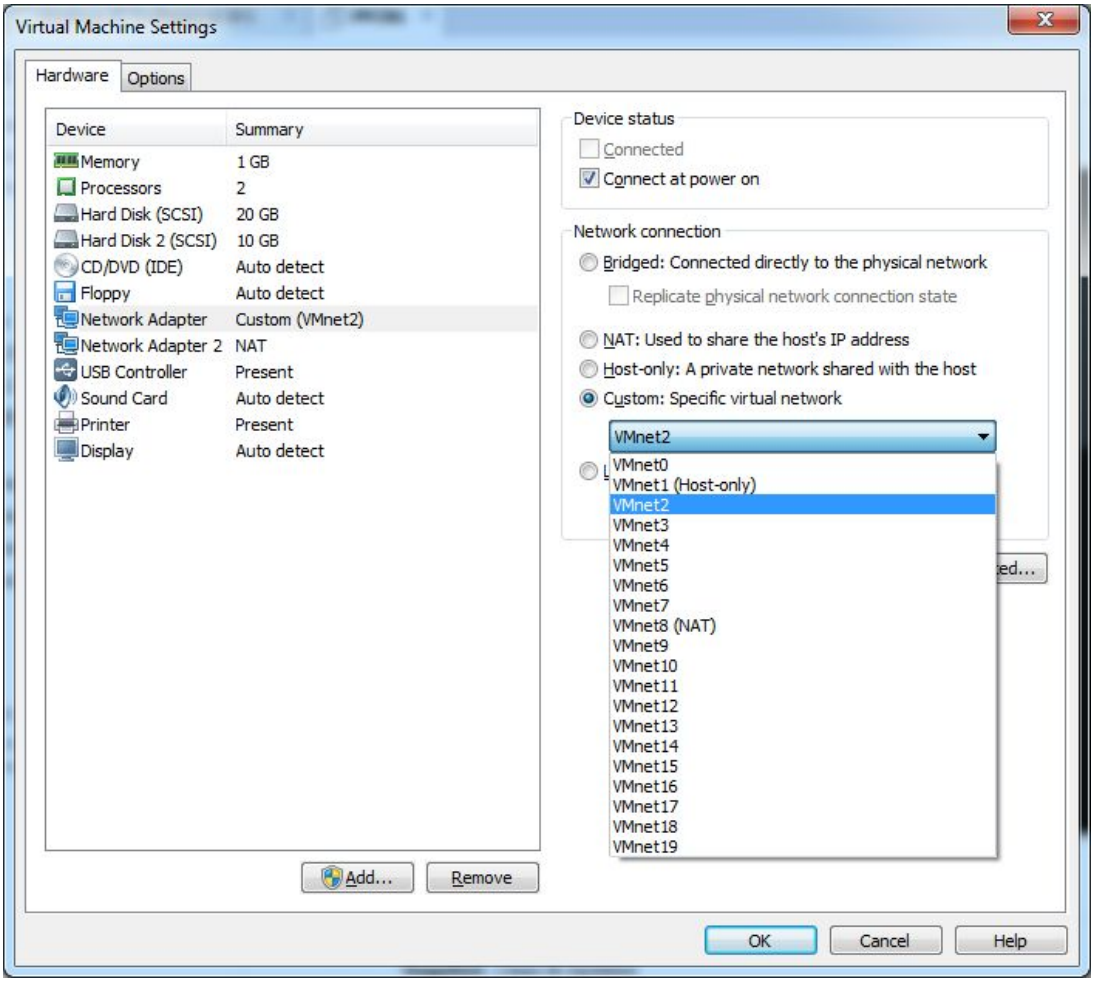


Figura A4.7: Configuración de la red virtual.

Es muy recomendable tomar una imagen (*snapshot*) del conjunto de las máquinas virtuales para tener un punto de restauración en un estado completamente limpio y configurado.

Anexo 5: Marco regulador

Al tratarse este estudio de un sistema de almacenamiento, la ley que hay que asegurar que se cumpla es la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal [31].

En esta ley se establece qué derechos tienen las personas cuyos datos han sido almacenados, qué obligaciones tienen las personas físicas o jurídicas que poseen un fichero donde almacenan datos, qué es la Agencia de Protección de Datos y las sanciones derivadas de la vulneración de los derechos de los titulares de los datos, del incumplimiento de los procedimientos establecidos para el tratamiento de los datos y del incumplimiento de las obligaciones del titular del fichero, entre otras cosas.

Los responsables del uso del sistema de almacenamiento elegido serán quienes deban determinar si esta ley es aplicable o no, ya que depende por completo del uso que le vaya a dar el grupo de investigación a dicho sistema. En principio, durante las reuniones no se ha mencionado que se tenga la intención de almacenar datos de carácter personal.

Se debe recordar que, según esta ley, los datos de carácter personal son «cualquier información concerniente a personas físicas identificadas o identificables».

Anexo 6: Nuevas personalidades de Filebench

Se escogió Filebench como una de las herramientas de medición para este estudio por su sencillez de uso, gracias a sus personalidades ya creadas y por su potencia a la hora de configurar las pruebas, debido a su lenguaje WML.

Para poder probar con Filebench el rendimiento de las soluciones escogidas cuando se les da un uso acorde con los perfiles de almacenamiento correspondientes al control de versiones y al almacenamiento de discos duros virtuales (RS-09 y RS-11, respectivamente) fue necesario escribir nuevas personalidades porque Filebench carece de personalidades predefinidas que se parezcan a esas necesidades de almacenamiento.

A continuación se incluye el código de esas nuevas personalidades: Versions.f y VM.f.

Versions.f:

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
```

```
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#


# This file has been created with academical purposes only.
# Universidad Carlos III de Madrid. Author: David Ventas Sierra.


set $dir=/tmp
set $filesize=3k
set $iosize=4k

define fileset
name=svnfileset,path=$dir,entries=70,dirwidth=3,dirgamma=0,size=$filesize,filesizegamma=1600,prealloc,paralloc

define process name=Versions,instances=4
{
 thread name=sec-read,memsize=5m,instances=3
 {
   flowop readwholefile name=sec-read1, filesetname=svnfileset, iosize=$iosize
   flowop readwholefile name=sec-read2, filesetname=svnfileset, iosize=$iosize
   flowop readwholefile name=sec-read3, filesetname=svnfileset, iosize=$iosize
   flowop readwholefile name=sec-read4, filesetname=svnfileset, iosize=$iosize
```

```
    flowop readwholefile name=sec-read5, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read6, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read7, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read8, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read9, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read10, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read11, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read12, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read13, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read14, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read15, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read16, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read17, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read18, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read19, filesetname=svnfileset, iosize=$iosize
    flowop readwholefile name=sec-read20, filesetname=svnfileset, iosize=$iosize

    flowop writewholefile name=sec-write1, filesetname=svnfileset, iosize=$iosize
  }
}

echo  "Repository Version 1.0 personality successfully loaded"
usage "Usage: set \$dir=<dir>"
usage "       set \$filesize=<size>    defaults to $filesize"
usage "       set \$iosize=<size>  defaults to $iosize"
usage "       run runtime (e.g. run 60)"
```

VM.f:

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# This file has been created with academical purposes only.
# Universidad Carlos III de Madrid. Author: David Ventas Sierra.

set $dir=/tmp
set $filesize=4g
set $iosize=4k
```

```
define file name=vmfile,path=$dir,size=$filesize,prealloc,reuse,paralloc

define process name=VMSW,instances=1
{
  thread name=rand-read,memsize=5m,instances=3
  {
    flowop read name=rand-read1, filename=vmfile, iosize=$iosize, random
  }
  thread name=rand-read,memsize=5m,instances=1
  {
    flowop write name=rand-write1, filename=vmfile, iosize=$iosize, random
  }
}



echo  "VM Software Version 1.0 personality successfully loaded"
usage "Usage: set \$dir=<dir>"
usage "       set \$filesize=<size>    defaults to $filesize"
usage "       set \$iosize=<size>  defaults to $iosize"
usage "       run runtime (e.g. run 60)"
```

Anexo 7: Bibliografía

1 JIMÉNEZ CAPEL, M. Bases de datos relacionales y modelado de datos. ISBN 978-84-16109-76-0. Málaga. IC Editorial. 2014. Disponible en: <https://books.google.com/books?id=uhHmCQAAQBAJ&pg=PT1&lpg=PT1&dq=bases+de+datos+relacionales+y+modelado+de+datos+yolanda&source=bl&ots=B54bviXiqL&sig=SkBkoOnmZaVIwxNAwcTYW2kfDW0>

2 FOWLER, M. y SADALAGE, P. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. ISBN 978-0321826626. Addison-Wesley. 2013.

3 WIKIPEDIA THE FREE ENCYCLOPEDIA. NoSQL - Wikipedia, la enciclopedia libre. 4 de agosto de 2015. Disponible en: <https://es.wikipedia.org/wiki/NoSQL>

4 FOWLER, M. NoSQL Definition. 9 de Enero de 2012. Disponible en: <http://martinfowler.com/bliki/NosqlDefinition.html>

5 WEBSITE OPTIMIZATION LLC. Average Web Page Breaks 1600K. 8 Julio de 2014. Disponible en: <http://www.websiteoptimization.com/speed/tweak/average-web-page/>

6 SUN MICROSYSTEMS INC. Java Code Conventions 2007. Disponible en: <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

7 AYUNTAMIENTO DE MADRID. INFORMÁTICA DEL AYUNTAMIENTO DE MADRID. Anexo a la guía de estándares. Normas de codificación. 2011. Disponible en: <http://www.madrid.es/UnidadesDescentralizadas/UDCIAM/Documentaci%C3%B3n%20t%C3%A9cnica/Gu%C3%ADaEstandares/Ficheros/Anexos/Anexo%207.%20Normas%20de%20c%C3%B3digo.doc>

8 TSCHABITSCHER, H. What Is the Average Size of an Email Message? – About.com. Agosto de 2012 Disponible en: <http://email.about.com/od/emailstatistics/f/What_is_the_Average_Size_of_an_Email_Message.htm>

9 INKTANK STORAGE INC., Project Overview, Ceph Dashboard. Enero de 2014. Disponible en: <http://metrics.ceph.com/>

10 KILLIAN, E. SoftLayer - Tutorial Number Six - Part 1 [videograbación]. Episodio 6. Introducing Object Storage. 8 de septiembre de 2014. Disponible en: <https://www.youtube.com/watch?v=m8a_nqR7zJA&ab_channel=EamonnKillian>

11 WIKIPEDIA THE FREE ENCYCLOPEDIA, Object Storage [imagen digital]. 9 de julio de 2015. Disponible en: <https://upload.wikimedia.org/wikipedia/commons/9/97/High_level_object_storage_architecture.png>

12 INKTANK STORAGE INC., Ceph Documentation, Architecture [imagen digital]. 20 de Junio de 2015. Disponible en: <http://ceph.com/docs/master/_images/stack.png>

13 BRANDT, S., LEUNG, A., MALTZAHN, C. y WEIL, S. RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters. 11 de Noviembre de 2007. Disponible en: <http://www.pdsw.org/pdsw07/resources/p35-weil.pdf>

14 INKTANK STORAGE INC., Ceph Documentation, Architecture, Ceph Protocol [imagen digital]. 20 de Junio de 2015. Disponible en: <http://ceph.com/docs/master/_images/ditaa-1a91351293f441ce0238c21f2c432331a0f5a9d3.png>

15 INKTANK STORAGE INC., Ceph Documentation, Ceph Object Gateway [imagen digital]. 20 de Junio de 2015. Disponible en: <http://ceph.com/docs/master/_images/ditaa-50d12451eb76c5c72c4574b08f0320b39a42e5f1.png>

16 INKTANK STORAGE INC., Ceph Documentation, Ceph Block Device [imagen digital]. 20 de Junio de 2015. Disponible en: <http://ceph.com/docs/master/_images/ditaa-dc9f80d771b55f2daa5cbbfdb2dd0d3e6dfc17c0.png>

17 INKTANK STORAGE INC., Ceph Documentation, Architecture, Ceph Filesystem [imagen digital]. 20 de Junio de 2015. Disponible en: <http://ceph.com/docs/master/_images/ditaa-1cae553f9d207d72257429d572673632afbd108c.png>

18 BLACK, D. Red Hat Storage Server Internals & Architecture [imagen digital]. 11 de Septiembre de 2014. Disponible en: <http://people.redhat.com/dblack/reveal.js/rhss_internals/index.html?print-pdf#/>

19 GLUSTER INC. Cloud Storage for the Modern Data Center. An Introduction to Gluster Architecture [imagen digital]. 2014. Disponible en: <http://moo.nac.uci.edu/~hjm/fs/An_Introduction_To_Gluster_ArchitectureV7_110708.pdf>

20 GLUSTER INC. Getting started with GlusterFS, Architecture [imagen digital]. 2015. Disponible en: <https://cloud.githubusercontent.com/assets/10970993/7412364/ac0a300c-ef5f-11e4-8599-e7d06de1165c.png>

21 GLUSTER INC. Getting started with GlusterFS, Architecture [imagen digital]. 2015. Disponible en: <https://cloud.githubusercontent.com/assets/10970993/7412379/d75272a6-ef5f-11e4-869a-c355e8505747.png>

22 GLUSTER INC. Getting started with GlusterFS, Architecture [imagen digital]. 2015. Disponible en: <https://cloud.githubusercontent.com/assets/10970993/7412402/23a17eae-ef60-11e4-8813-a40a2384c5c2.png>

23 GLUSTER INC. Getting started with GlusterFS, Architecture [imagen digital]. 2015. Disponible en: <https://cloud.githubusercontent.com/assets/10970993/7412387/f411fa56-ef5f-11e4-8e78-a0896a47625a.png>

24 GLUSTER INC. Getting started with GlusterFS, Architecture [imagen digital]. 2015. Disponible en: <https://cloud.githubusercontent.com/assets/10970993/7412394/0ce267d2-ef60-11e4-9959-43465a2a25f7.png>

25 SKYTECHNOLOGY LizardFS White Paper, Architecture [imagen digital]. 2015. Disponible en: <http://lizardfs.com/wp-content/uploads/2015/07/lizardfs-whitepaper.pdf>

26 GOBIERNO DE ESPAÑA. MINISTERIO DE LA PRESIDENCIA. AGENCIA ESTATAL BOLETÍN OFICIAL DEL ESTADO. Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. Nº 298. 13 de Diciembre de 1999. Disponible en : <http://www.boe.es/buscar/pdf/1999/BOE-A-1999-23750-consolidado.pdf>

Anexo 8: Otros recursos

I "Amazon Web Services Simple Monthly Calculator." 2007. 9 Jul. 2015 <http://calculator.s3.amazonaws.com/calc5.html>

II "yUML." 2009. 3 Sep. 2015 <http://yuml.me/>

III "Tom's Planner: Planificador de proyectos | Diagrama de ..." 2013. 3 Sep. 2015 <http://www.tomsplanner.es/>

IV "GridFS — MongoDB Manual 3.0.4." 2013. 26 Jul. 2015 <http://docs.mongodb.org/manual/core/gridfs>

V "mikejs/gridfs-fuse · GitHub." 2010. 26 Jul. 2015 <https://github.com/mikejs/gridfs-fuse>