Arquitectura para un Sistema de Ficheros Distribuido Orientado al Ámbito de las Aplicaciones Grid y Cloud.

Tesis de Máster (20 créditos).
Máster Oficial: Computación Paralela y Distribuida.

Autor: José Vicente Carrión Burguete.
Director: Vicente Hernández García.
Co-Director: Germán Moltó Martínez.

Marzo de 2011.

Tesis de Máster adscrita al Máster Oficial en Computación Paralela y Distribuida.

RESUMEN

Arquitectura para un Sistema Distribuido de Ficheros Orientado al Ámbito de las Tecnologías Grid y Cloud.

Desarrollo de un Prototipo Funcional para el Middleware Grid Fura en el Contexto del Proyecto ITECBAN (Infraestructura Tecnológica y Metodológica de Soporte para un Core Bancario).

Por José Vicente Carrión Burguete.

Actualmente las aplicaciones y los sistemas informáticos requieren la utilización de un conjunto extenso de recursos para poder llevar a cabo la funcionalidad por la que fueron diseñados.

Las tecnologías Grid y Cloud ofrecen una plataforma adecuada que puede ser aprovechada por aplicaciones de cualquier ámbito, para obtener prestaciones superiores a las que se podrían obtener si se ejecutasen haciendo uso de recursos individuales. En este sentido, los recursos requeridos por las aplicaciones que hacen uso del Grid no se refieren únicamente a disponer de gran capacidad de cálculo sino que cada vez con más importancia, las aplicaciones necesitan una capacidad de almacenamiento lo suficientemente grande para hacer persistente la gran cantidad de información que generan cuando son ejecutadas sobre una plataforma Grid o Cloud.

La mayoría de los sistemas de almacenamiento utilizados en los sistemas Grid y Cloud, adoptan una arquitectura centralizada de almacenamiento que origina problemas importantes como poca tolerancia a fallos o ineficiencia en el acceso a los ficheros. Por otra parte, el resto los sistemas de almacenamiento actuales utilizan una arquitectura parcialmente distribuida que consiste en almacenar los ficheros sobre un conjunto de servidores siguiendo una estructura jerárquica. Esta arquitectura solventa parcialmente los problemas de tolerancia a fallos pero introduce otros inconvenientes ya que no es posible asegurar en tiempo real la coherencia el sistema de ficheros.

Esta Tesina de Máster propone una arquitectura puramente distribuida para el almacenamiento de los ficheros generados por las aplicaciones que se ejecutan en los entornos Grid y Cloud. La arquitectura y el prototipo funcional que se ha desarrollado resuelven los problemas de almacenamiento con que cuentan este tipo de entornos y ofrecen un soporte de almacenamiento con características específicas de eficiencia, alta disponibilidad y tolerancia a fallos en tiempo real. El sistema de almacenamiento propuesto permite mantener la accesibilidad a los ficheros a pesar de que pudieran producirse fallos en alguno de sus componentes. Todas estas características hacen de la arquitectura propuesta una plataforma robusta y estable para aquellas aplicaciones que requieran un soporte estable y escalable de almacenamiento.

Se ha conseguido integrar y testear un prototipo funcional de la arquitectura sobre un sistema Grid en producción. Los resultados obtenidos demostraron que el sistema propuesto resolvió los problemas iniciales de almacenamiento con que contaba el software y además permitió aumentar su escalabilidad y tolerancia a fallos de manera significativa.

Índice temático de contenidos:

Capítulo 1 - Introducción.

Uno de los principales objetivos de las tecnologías Grid [1] es proporcionar a los usuarios y aplicaciones un conjunto extenso de recursos informáticos, ofreciendo así una capacidad de cálculo y de almacenamiento muy superior a la que se podría obtener de un único recurso.

Actualmente existe un gran número de aplicaciones que utilizan las tecnologías Grid con ámbitos de aplicación muy heterogéneos. Una característica común a todos estos sistemas, es la necesidad de requerir un soporte de almacenamiento lo suficientemente robusto para hacer persistente un volumen grande de ficheros. Una aplicación durante su ejecución en una plataforma Grid, requiere el acceso a un conjunto de ficheros de entrada y tras su procesamiento se genera un conjunto de ficheros de salida que requieren almacenarse para ser accedidos posteriormente.

La mayoría de infraestructuras Grid ofrecen un sistema de almacenamiento poco flexible y rudimentario consistente en ejecutar un proceso previo de obtención de los ficheros de entrada que serán procesados y almacenados en un determinado recurso del Grid. Tras el procesamiento de la tarea en el Grid, existe un proceso encargado de enviar los ficheros generados localmente al recurso, a un servidor centralizado de almacenamiento desde el que los usuarios pueden recuperar los resultados de sus ejecuciones.

El modelo de almacenamiento anterior introduce algunas carencias y desventajas como puede ser el hecho de no poder utilizar los ficheros de salida que van generando las aplicaciones hasta que no han sido almacenados sobre algún dispositivo bien conocido por el resto de usuarios del Grid.

Otra carencia con que cuentan los sistemas de almacenamiento Grid, es la no existencia de una capa de abstracción que permita a las aplicaciones acceder a la información sin la necesidad de conocer la ubicación física de los datos almacenados. Algunos sistemas emplean un servidor bien conocido por todas las aplicaciones del Grid para centralizar el almacenamiento de los ficheros. Este tipo de plataformas proporcionan un sistema poco escalable y muy poco tolerante a fallos, ya que un fallo en el servidor de almacenamiento impediría el acceso a los ficheros y por tanto, la inoperatividad temporal de los datos almacenados en el Grid.

Al contrario de esto y como alternativa, sólo unos cuantos sistemas utilizan un modelo distribuido para el almacenamiento de los datos de un Grid, pero la mayoría carece de mecanismos adecuados para acceder de forma eficiente a los ficheros, asegurar un acceso ininterrumpido e independiente a fallos en el sistema o mantener actualizadas todo el tiempo cada una de las réplicas de un fichero.

Los proyectos que existen actualmente han desarrollado herramientas que ofrecen el almacenamiento de ficheros utilizando mecanismos muy específicos de acuerdo a las necesidades propias de cada proyecto. Por eso, es difícil encontrar un sistema de almacenamiento que cumpla con requerimientos como tolerancia a fallos, transparencia en el acceso a los datos, utilización de algoritmos de replicación inteligentes que propicien un acceso eficiente, etc.

En cualquier caso, las características intrínsecas de las tecnologías Grid y el rápido desarrollo de las nuevas tecnologías de virtualización Cloud originan la necesidad de proporcionar una plataforma de almacenamiento robusta que permita almacenar toda la información que generan las aplicaciones y al mismo tiempo proporcionen accesibilidad a los ficheros, sin importar la ubicación del Grid desde donde se acceda a los datos.

Esta tesina de máster propone el diseño e implementación de un sistema distribuido que proporcionará soporte para el almacenamiento de los ficheros generados por las aplicaciones que se ejecutan en plataformas Grid o Cloud [2]. El sistema que se describe a continuación, cumple con los requerimientos que se especificaron anteriormente y añade algunas características adicionales motivo por el que sería factible la utilización de este sistema por aplicaciones con un ámbito de aplicación distinto al que inicialmente motivó su desarrollo y que más adelante se describirá.

1.1 Motivación.

El desarrollo de este trabajo de Máster surge motivado durante mi colaboración dentro del proyecto ITECBAN en el que ha participado el grupo de investigación GRyCAP[1], y que consistió en el desarrollo de ciertas funcionalidades del proyecto por medio de una colaboración con la empresa GridSystems [22].

ITECBAN (Infraestructura tecnológica y metodológica de soporte para un "Core" bancario) es uno de los 16 proyectos aprobados dentro de la primera convocatoria del programa CENIT promovido por el Ministerio de Industria en el año 2006. Su objetivo era desarrollar una plataforma de banca avanzada con tecnologías más modernas de las que están en uso ahora, para conducir a una renovación de la infraestructura tecnológica bancaria de largo alcance.

El proyecto se estructura por medio de la creación de un consorcio de empresas entre las que se encuentra GridSystems encargada de proveer su infraestructura Grid para el tratamiento de los procesos batch y el procesado distribuido de un volumen grande de datos.

En relación a este último aspecto, el grupo de investigación GRyCAP ha colaborado en distintas facetas de la ejecución del proyecto y por medio de un contrato de I+D colaborativa con la empresa GridSystems para desarrollar nuevos componentes para su software Fura Project [23]. Como consecuencia de este periodo de colaboración y del desarrollo de nuevos componentes aplicables a ámbitos diferentes al de propio proyecto inicial, ha surgido la motivación para el desarrollo de esta tesina de Máster.

Fura Project es un software de código abierto para la ejecución distribuida de procesos sobre plataformas heterogéneas. Por medio de un interface web sencillo, los usuarios pueden definir los procesos que se ejecutarán sobre un conjunto de recursos Grid. El planificador Fura se encarga de seleccionar los recursos más adecuados a las características de las tareas y proporciona todo tipo de herramientas para facilitar la recuperación de los resultados y la parametrización de las ejecuciones.

Uno de los componentes principales de Fura es su sistema de almacenamiento ya que es el encargado de almacenar los ficheros de entrada y salida que generan las tareas que se ejecutan en el sistema Grid. Además de los ficheros asociados a las tareas, el componente de almacenamiento de Fura también guarda la estructura de datos interna del software. Por este motivo es esencial que el servicio de almacenamiento de Fura esté accesible en todo momento.

Una característica importante del sistema de almacenamiento de Fura, es el hecho de que un mismo servidor es el encargado de planificar la ejecución de los procesos en los recursos Grid junto al almacenamiento de los datos que generan las ejecuciones. Teniendo en cuenta un análisis previo sobre el volumen de carga que suele manejar la herramienta Fura, se comprobó que para los despliegues Fura con mayores requerimientos de recursos, el tamaño máximo que puede ocupar el repositorio de almacenamiento podía llegar a ascender hasta el Terabyte de tamaño utilizado en disco. Del mismo modo el número de ficheros generados por las tareas, pueden oscilar en el peor de los casos hasta los 50.000.

De los datos recogidos en el análisis de requerimientos, se intuye fácilmente la existencia de un aspecto esencial a solucionar ya que actualmente el almacenamiento de los ficheros se realiza de modo centralizado a cargo de un único servidor Fura y esta circunstancia se vuelve inviable para volúmenes de carga importantes.

Después de analizar todos estos aspectos, se llegó a la conclusión de que era necesario disponer de un sistema de almacenamiento que fuera escalable en el tiempo y tolerante a los fallos que se produjeran en los componentes individuales de almacenamiento del sistema. Todos estos aspectos junto con los que se

[1] GRyCAP, Grupo de Grid y Computación de Altas Prestaciones.

describen en el apartado de objetivos generales son los que motivaron el diseño e implementación de un prototipo de sistema para el almacenamiento distribuido de ficheros.

1.2 Hipótesis de Trabajo.

La información que generan las aplicaciones dentro de los sistemas Grid y por extensión en las infraestructuras Cloud es cada vez más grande. Este aspecto se traduce en la necesidad de mantener para estos sistemas, un número de ficheros y un tamaño en bytes de información lo suficientemente grande como para ser tratado con las herramientas que actualmente existen.

Actualmente la mayoría de desarrollos dentro del ámbito de los sistemas Grid y Cloud, utilizan un modelo centralizado de almacenamiento muy desaconsejable para este tipo de entornos, ya que introducen puntos débiles en el funcionamiento del sistema, que podrían causar la inoperatividad del sistema tras concurrir determinadas circunstancias.

En comparación con los modelos centralizados, los sistemas distribuidos introducen características que permitirían dotar a los sistemas de almacenamiento Grid y Cloud de características ciertamente esenciales como son la escalabilidad, tolerancia a fallos y abstracción del usuario en el acceso a los datos.

Analizando toda esta problemática se ha llegado a la determinación que sustituir los modelos centralizados de almacenamiento por una arquitectura distribuida adecuada, solucionará las carencias que muestran las herramientas Grid como es el caso concreto de Fura y más ampliamente para cualquier sistema de almacenamiento integrado en una infraestructura Grid o Cloud.

1.3 Objetivos Generales.

Las principales características que debería poseer una herramienta de almacenamiento con soporte a las aplicaciones Grid y Cloud se enumeran a continuación:

- **Descentralización del sistema de almacenamiento**, para evitar que fallos en un componente central impidan el acceso al sistema y también la pérdida de los ficheros almacenados en él.

- **Sistema escalable en tiempo real**, para hacer posible aumentar o disminuir la capacidad total del sistema de almacenamiento dependiendo de los requerimientos y la carga en el sistema en cada momento.

- **Mecanismos de balanceo de carga**, ya que un sistema mal balanceado puede originar la denegación en el servicio de almacenamiento y por tanto una situación similar a la descrita para los sistemas centralizados.

- **Replicación de los ficheros contenidos en el sistema**, que permite implementar mecanismos de recuperación de ficheros y por tanto incrementar la tolerancia a fallos en el sistema.

- **Abstracción por parte del usuario en el acceso a sus ficheros**. El usuario final y las aplicaciones no deben conocer en ningún momento donde se encuentran almacenados sus ficheros.

  Una vez descritas las principales características que debería poseer un sistema de almacenamiento en Grid, se enumeran a continuación los principales objetivos que se han propuesto para la realización de esta Tesina de Máster.

- **Descripción de una arquitectura** para un sistema de almacenamiento de ficheros que ofrezca soporte a los requerimientos de almacenamiento de Fura. El sistema deberá poner solución a los principales puntos débiles y carencias del sistema actual y permitirá una capacidad de almacenamiento acorde a los requerimientos impuestos por el propio software.

- **Implementación de un prototipo funcional** y posterior integración con el software Fura. El prototipo implementado deberá poder ser evaluado por su funcionalidad no importando en esta primera aproximación su eficiencia. Se realizará una prueba de concepto para verificar el buen funcionamiento del prototipo con Fura.

  La arquitectura definida ofrecerá mecanismos de **tolerancia a fallos en tiempo real**, lo que significa que el sistema de almacenamiento deberá estar en un estado disponible en todo momento y a pesar de que pueda producirse fallos en los componentes de almacenamiento que lo forman. La tolerancia a fallos también debe ser contemplada desde el punto de vista de pérdida de ficheros por lo que deberá habilitarse algún mecanismo de replicación para proteger los ficheros ante cualquier eventualidad.

- El sistema de replicación implementado deberá optimizar las operaciones de transferencia de los ficheros. De este modo se deberá contemplar la opción de integrar diferentes **algoritmos de replicación** que permitan acercar los ficheros a las fuentes de datos (aplicaciones). Se evita así costosas transacciones en las transferencia de los ficheros mejorando el rendimiento global de las aplicaciones.

- Las réplicas almacenadas de un fichero se mantendrán actualizas en tiempo real, esto quiere decir que existirán **mecanismos de bloqueo sobre los ficheros** para impedir su actualización mientras estén siendo actualizados desde una ubicación diferente en el sistema.

- El sistema deberá proporcionar **transparencia en el acceso a los datos** independientemente de la ubicación donde se almacenan los ficheros. Se especificará un API (*Application Programing Interface*) que permita a los usuarios acceder a sus ficheros del mismo modo que se hace en un sistema local de almacenamiento.

1.4 Estructura de la Memoria Presentada.

La memoria presentada se organiza en 8 capítulos (Cap. 1 “Introducción”, Cap. 2 “Estado del Arte”, Cap. 3 “Introducción a Fura”, Cap. 4 “Propuesta de una Arquitectura Distribuida de Almacenamiento de Ficheros”, Cap. 5 “Diseño de Componentes”, Cap. 6 “Integración del Sistema con Fura”, Cap. 7 “Conclusiones”, Cap. 8 “Trabajo Futuro” y las referencias bibliográficas.)

Tras la introducción, el capítulo 2 realiza un análisis del estado del arte en la que se describe las herramientas de almacenamiento distribuido que existen actualmente.

El capítulo 3 introduce los conceptos más importantes del software Fura y también detalla su arquitectura de almacenamiento de ficheros, además se describirán algunas de las carencias y requerimientos más destacables que posee esta herramienta.

La sección 4 realiza una completa descripción de la arquitectura propuesta. Esta sección comienza planteando los requerimientos que debe aportar la nueva arquitectura así como la justificación para el diseño multicapa de la arquitectura. Por último la sección realizará un análisis detallado a cada uno de los componentes de la arquitectura.

Tras especificar la arquitectura del sistema, en el apartado 5 se realiza un estudio más detallado de cada uno de los módulos que componen el sistema diseñado, además se comentan aspectos relacionados con la funcionalidad y los diagramas de actividad entre los diferentes componentes.

En el capítulo 6 se detalla el proceso de integración del prototipo implementado con el software Fura, además se describirán las pruebas realizadas al prototipo. Para realizar estas pruebas se utilizará un número acotado de nodos de almacenamiento y un volumen de carga de trabajo limitada.

Por último, en el capítulo 7 y 8 se exponen las conclusiones finales sobre el trabajo realizado y se muestra un conjunto de posibles tareas que podrían desarrollarse en el futuro para mejorar y extender la arquitectura propuesta.

1.5 Aportaciones y Aplicaciones.

La principal aportación de esta Tesina de Máster es definir una arquitectura distribuida que proporcione soporte para el almacenamiento de ficheros a las aplicaciones de tipo Grid y Cloud. En relación a lo anterior, la arquitectura propuesta deberá además cumplir con ciertas características básicas impuestas por el contexto del sistema Fura en dónde debe integrarse. Las principales características que deberá aportar son tolerancia a fallos en tiempo real y un sistema de replicación inteligente que permita acercar los ficheros a sus posibles consumidores, evitando de este modo futuras transferencias innecesarias.

Además de la anterior aportación, existen otras contribuciones secundarias que serán expuestas más adelante. Una de las aportaciones más destacadas, es quizás la de definir un diseño desacoplado que permite independizar los sistemas de búsqueda, catalogación y almacenamiento de ficheros. La utilización de estos sistemas de manera independiente, aporta numerosas ventajas respecto a otros sistemas de almacenamiento con un diseño compacto. Así, al independizar el sistema de almacenamiento físico respecto al sistema de catalogación, se hace posible que el sistema continúe operando con normalidad pese a producirse un fallo en alguno de los nodos ya que la información de tipo metadato asociada a cada fichero reside en un nodo diferente al de su almacenamiento físico. A este respecto, en caso de fallo múltiple en los nodos de catalogación, se proporciona procedimientos de recuperación que consiguen recuperar el sistema de catálogo a partir de los ficheros físicos almacenados en los dispositivos.

A pesar de que el principal objetivo de esta Tesina de Máster es la integración de un nuevo sistema de almacenamiento con el software Fura, la arquitectura propuesta es lo suficientemente flexible y robusta para ser utilizada en otros tipos de sistemas que impliquen el almacenamiento de ficheros con unos requerimientos distintos a los de Fura. No obstante, hay que tener en cuenta que el prototipo implementado se ha validado para proporcionar funcionalidad con una carga de trabajo acorde a los requerimientos de Fura.

Debido al diseño modular y a la utilización de las tecnologías distribuidas, la arquitectura propuesta permite escalar el tamaño del sistema de ficheros dependiendo de los requerimientos de carga de trabajo específicos que posea una organización. De este modo, es posible aumentar el número de nodos que componen el sistema con el objetivo de balancear la carga de almacenamiento para despliegues con gran volumen de almacenamiento de ficheros. Del mismo modo será posible utilizar un número de nodos de almacenamiento menor cuando el volumen de carga se reduzca, todo ello de un modo dinámico, en tiempo real y sin interferir en el funcionamiento del sistema de almacenamiento.

Aunque la primera aproximación del sistema propuesto esté pensada para una utilización en el ámbito de las tecnologías Grid y Cloud, la arquitectura ofrece una funcionalidad adecuada para ser utilizada con sistemas heterogéneos de propósito general.

A continuación se exponen algunos ejemplos de posibles aplicaciones de la arquitectura propuesta.

- **Almacenamiento de ficheros en el contexto de las aplicaciones Grid y Cloud**. Estos sistemas requieren una gran capacidad de almacenamiento y tolerancia a fallos. Un ejemplo de este tipo de sistemas es Fura.

- **Sistemas que requieran un soporte de almacenamiento robusto y tolerante a fallos**. Requieren una fiabilidad permanente en el acceso y recuperación de los datos almacenados. Este tipo de sistemas están siempre en uso y requieren que el sistema de almacenamiento esté continuamente operativo para un funcionamiento correcto. Un ejemplo de este tipo de sistemas son los orientados a dar soporte bancario y transaccional.

- **Sistemas que requieran características de ubicuidad en el acceso a los datos** por los usuarios. Puede tratarse de organizaciones que requieran que los datos almacenados estén disponibles independientemente de la ubicación dónde se encuentre el usuario. Las

infraestructuras Cloud de tipo *Software as a Service* (SaaS) [[3] pueden ser un ejemplo de este tipo de aplicaciones finales ya que requieren sistemas de almacenamiento que eliminen la necesidad de conocer dónde se almacenan los ficheros.

Capítulo 2 - Estado del Arte.

El almacenamiento de datos constituye un componente esencial en cualquier tipo de sistema para las aplicaciones, independientemente de la funcionalidad para la que estén desarrolladas.. Cualquier aplicación que maneje volúmenes de información elevados, requiere de una plataforma de almacenamiento que permita hacer persistente la información que consumen y generan. Por este motivo, la elección de un sistema de almacenamiento adecuado repercutirá en gran medida en la eficiencia global del sistema. Por el contrario una elección errónea puede suponer con el paso del tiempo una merma de las prestaciones que ofrece el sistema, llegando incluso a hacerlo inoperativo.

Existen en la actualidad multitud de herramientas y sistemas que permiten al almacenamiento de ficheros. Por ello es imprescindible realizar una clasificación atendiendo a las características que poseen las aplicaciones actuales y a los entornos de ejecución sobre los que pueden ser utilizadas.

2.1 Principales Sistemas de Almacenamiento Distribuido.

A continuación se enumera un conjunto de características deseables para cualquier aplicación que requiera almacenar sus ficheros. Este análisis permitirá realizar posteriormente una clasificación de los principales sistemas de almacenamiento que hay actualmente.

- **Virtualización del espacio de nombres de los ficheros**. Se requiere un mecanismo de acceso a los ficheros que abstraiga la necesidad de conocer la ubicación física en que se alojan los ficheros. Un espacio de nombres común para todo el sistema de almacenamiento permite asociar de forma única cada fichero con una etiqueta alfanumérica. De esta manera, el sistema de almacenamiento distribuido puede ser accedido del mismo modo a como se accedería en un sistema local de almacenamiento.

- **Balanceo del espacio de almacenamiento** entre todos los nodos que componen el sistema distribuido.

- **Replicación de ficheros**. Proporcionando redundancia de los ficheros se evita posibles pérdidas a la vez que se ofrece un acceso más rápido a los ficheros. Este aspecto, permite aumentar el ancho de banda que tienen las aplicaciones en el acceso a los datos.

- **Aproximación de los ficheros a las aplicaciones que finalmente los utilizarán**. Proporciona una mayor eficiencia global al sistema ya que las aplicaciones pueden acceder a un mismo fichero repetidas veces desde la ubicación más próxima dónde se encuentre almacenado.

- **Coherencia del sistema de replicación**. Un fichero en el sistema puede poseer múltiples réplicas, pero el acceso simultáneo a cada una de ellas desde distintas ubicaciones y durante todo el tiempo debe ser coherente. Esto implica el desarrollo de un sistema de actualización de réplicas en tiempo real.

- **Descentralización del sistema de catálogo**. Debe existir un sistema de catalogación que registre la información asociada al conjunto de ficheros almacenado en el sistema. Se debe evitar disponer de un único servidor que se ocupe de centralizar todo el sistema de catalogación puesto que un número elevado de peticiones podría provocar fallos en el acceso a la información.

- **Tolerancia a fallos**. El sistema de ficheros debería continuar funcionando a pesar de que se produzcan fallos en algunos nodos o tramos de red. La tolerancia a fallos puede ser entendida a diferentes niveles conceptuales pero en los sistemas de almacenamiento integrados en entornos Grid se precisa una tolerancia a fallos a tiempo real, lo que significa que cuando falla un

componente del sistema los ficheros deben permanecer accesibles y sin requerir una intervención manual por parte de los administradores del sistema.

- **Facilidad de acceso**. Una interfaz simple debe permitir realizar las operaciones básicas de escritura / lectura con los ficheros tal y como se operaría en un sistema de almacenamiento local.

Revisando el conjunto de herramientas que existen actualmente, se ha seleccionado las más significativas para comparar la funcionalidad que ofrecen.

A continuación se expone un listado de los principales sistemas de almacenamiento distribuido como pueden ser EGEE's[2] Replica Service [4], LCG[3] File Catalog [5] [6], RLS[4] de Globus y DataGrid [7] [8], Hadoop Distributed File System [9], Wuala [10], DropBox [11], Past [12] o Microsoft's Distributed File System [13]. Sin embargo, todos estos proyectos poseen algunas carencias importantes que han impedido que se consolidaran en el ámbito del almacenamiento de aplicaciones Grid.

- **LCG File Catalog**: es el servicio de replicación de ficheros desarrollado por el proyecto LCG para proporcionar una visión única del sistema de ficheros del Grid. Cada fichero posee un identificador único llamado GUI (Grid Unique Identifier) que puede ser referenciado por medio del identificador LFN (*Logical File Name*). Cada LFN puede tener asociadas una o múltiples réplicas físicas del fichero. A cada réplica almacenada en un dispositivo físico se le asocia un identificador llamado SURL (*Storage URL*). El acceso a los ficheros se realiza por medio de un conjunto de comandos similares a los utilizados en una máquina UNIX para acceder al sistema de almacenamiento local.

- **Globus Replica Location Service**: el servicio de localización de réplicas de Globus proporciona un mecanismo de mapeo entre nombres de ficheros lógicos y la localización de sus respectivas replicas. Por sí solo, no permite asegurar la coherencia de la información mantenida por los diferentes directorios de mapeo de réplicas que existen y se requiere que las aplicaciones que usan el servicio realicen esa tarea.

- **Hadoop**: se trata de un proyecto Apache en código abierto que agrupa un conjunto de herramientas con diferente funcionalidad como puede ser el almacenamiento distribuido de ficheros y el cálculo distribuido mediante técnicas de mapeo y reducción. En concreto cuenta con un componente de almacenamiento llamado HDFS (*Hadoop Distributed File System*) que ofrece redundancia en el almacenamiento de los ficheros por medio del envío de bloques entre varios nodos en el sistema.

  Cada fichero en *Hadoop* se divide en bloques de igual tamaño y cada uno de esos bloques es distribuido en nodos de almacenamiento diferentes. Además existe un sistema de replicación que permite realizar copias de cada bloque sobre diferentes dispositivos, por lo que la replicación se realiza a dos niveles (replica de fichero y bloque).

  La Fig. 1 muestra la arquitectura del sistema de almacenamiento HDFS *Hadoop*. Existe un único nodo de catálogo llamado *NameNode,* que contiene el listado completo de ficheros en el sistema junto con las referencias a la ubicación de cada uno de los bloques de datos que componen el fichero. Los nodos llamados *DataNode* son nodos de almacenamiento de bloques de ficheros y pueden agruparse en dominios de red (racks) dependiendo del segmento de red al que pertenezcan dentro de la organización. Agrupando de este modo los nodos y realizando una replicación sobre diferentes racks, se asegura que siempre existirá la posibilidad de recuperar un fichero aunque un nodo de almacenamiento esté inaccesible o un tramo de red en la organización esté cortado.

[2] EGEE, Enabling Grids for E-sciencE.
[3] LCG, Large hadron collider Computing Grid.
[4] RLS, Replica Location Service.

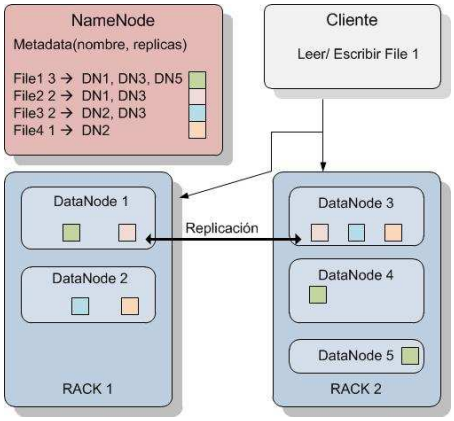

Fig. 1 Arquitectura HDFS.

Aunque el sistema ofrece un sistema de replicación por bloques de ficheros, HDFS tiene una arquitectura del tipo maestro/esclavo que obliga a las aplicaciones cliente a comunicarse siempre con un nodo servidor (*NameNode*) para recuperar o almacenar un fichero. Disponer de un sistema centralizado, proporciona un sistema poco tolerante a fallos ya que cualquier problema de acceso al nodo servidor, deja inaccesible el sistema de ficheros aunque posteriormente y de modo manual el administrador del sistema pueda volver a restaurar el acceso desde un servidor de respaldo.

- **Wuala**: se trata un software semipropietario[5] de almacenamiento online que permite a los usuarios subir ficheros desde su escritorio de trabajo y acceder a ellos posteriormente desde cualquier equipo con conexión a Internet. Los ficheros pueden ser compartidos entre una lista de contactos que mantiene el cliente.

  Wuala utiliza una arquitectura P2P (*peer to peer*) para distribuir y compartir los ficheros entre pares de nodos. Cada vez que un cliente envía los bloques de un fichero, estos se cifran por medio de una clave que es distinta para cada fichero y permanecen cifrados en el nodo destino. La lista de claves de los ficheros de un usuario se almacena también cifrada en un servidor perteneciente a Wuala y debe ser consultada cada vez que se quiere recuperar un fichero.

  Desde el punto de vista de la clasificación del sistema, se trata de una herramienta P2P estructurada con un grado de centralización híbrido, puesto que existe un servidor central encargado de almacenar el catálogo completo de ficheros junto con sus claves de encriptación. Una vez el cliente obtiene la referencia a los nodos de almacenamiento del fichero, la transferencia completa del fichero se realiza entre pares de nodos.

  Se trata de un modelo orientado a la compartición del espacio de almacenamiento ya que posee un mecanismo de incentivación, consistente en que los usuarios obtienen más capacidad de almacenamiento en el sistema a cambio de ofrecer el mismo espacio en su sistema local. También es posible obtener más capacidad de almacenamiento pagando una tarifa sin necesidad de compartir espacio en almacenamiento.

- **Dropbox**: básicamente posee la misma funcionalidad que Wuala ya que se trata de un sistema de almacenamiento y sincronización de ficheros online que proporciona también cifrado aunque únicamente a nivel de la transmisión de los ficheros.

- **PAST**: es una arquitectura de almacenamiento distribuido P2P estructurado y puramente descentralizado ya que pares de nodos almacenan y obtienen ficheros indistintamente sin

[5] Wuala implementa ciertos módulos en código abierto sin embargo protege bajo licencia otras partes de sus software.

necesidad de utilizar un servidor central donde localizar previamente los ficheros. Para obtener un fichero PAST utiliza en su capa inferior la arquitectura distribuida de localización de pares clave/valor llamada Pastry [14].

Pastry utiliza el modelo de tablas distribuidas DHT (*Distributed Hash Table*) [15] adoptando una topología en anillo para la interconexión de los nodos. A cada nodo del anillo se le asigna un identificador único que se obtiene aplicando una función de resumen sobre algún parámetro identificativo del nodo como su dirección IP o dirección MAC. Una vez asignados los identificadores cada nodo, se reordena en el anillo teniendo en cuenta que debe existir una correlación numérica entre los identificadores de los nodos vecinos. Así cada nodo siempre posee un nodo vecino directo con identificador estrictamente inferior a él y otro nodo vecino con identificador estrictamente superior. Una vez configurado el anillo de nodos, la localización de pares de valores es relativamente sencilla puesto que para buscar el valor de una clave se calcula previamente su código hash y se busca el nodo cuyo identificador esté más próximo.

Existen algoritmos de enrutamiento para realizar las búsquedas de claves con un número máximo de saltos de comunicación. El protocolo Pastry asegura un número máximo de saltos dado por la fórmula log N, siendo N el número de nodos activos en el sistema. Actualmente existe una implementación BSD de Pastry llamada FreePastry [16].

PAST hace uso del protocolo Pastry para localizar de modo determinista el nodo propietario de un determinado fichero. Una vez localizado el nodo que almacena un determinado fichero, la escritura o lectura es directa entre pares de nodos.

El sistema ofrece un mecanismo de replicación que permite almacenar cada fichero sobre los *k* nodos más cercanos al nodo encargado de almacenar un determinado fichero. Este sistema de replicación de ficheros posee una desventaja bastante importante ya que no es posible modelizar algoritmos inteligentes que permitan aproximar los ficheros a los nodos con mayor número de accesos.

- **Microsoft's DFS**: es un sistema de ficheros distribuido que ha sido implementado por Microsoft con el objetivo de proporcionar redundancia y transparencia en el acceso a los ficheros. Para conseguir esto, el sistema registra el conjunto de ficheros y directorios que comparten los usuarios en sus máquinas y los hace accesibles al resto de usuarios a través de un árbol de directorios virtual.

  DFS se ha diseñado para utilizar un servidor central encargado de catalogar los ficheros que gestiona el sistema y de realizar réplicas sobre otros servidores. El acceso a los ficheros hace uso de un sistema de nombres global lo que permite independizar la ubicación física dónde se encuentra el fichero.

  Lógicamente se trata de un sistema centralizado poco tolerante a fallos y con posibles problemas de inconsistencia del sistema de ficheros, además de estar orientado a un volumen de almacenamiento limitado.

2.2 Clasificación de los Sistemas de Almacenamiento Actuales.

Los principales sistemas de almacenamiento de ficheros pueden ser clasificados de acuerdo a los parámetros expuestos en la Tabla 1.

Sistema de Ficheros	Catálogo Distribuido	Soporte de Replicación	Tolerancia a Fallos	Integración en el Grid	Replicación Coherente	Búsqueda Determinista	P2P DHT	Replicación Inteligente
LCG	Parcial	SI	Medio	Alto	NO	NO	NO	NO
Globus	Parcial	SI	Medio	Alto	NO	NO	NO	NO
Hadoop	NO	SI	Bajo	Alto	SI	SI	NO	NO
Microsoft's DFS	NO	SI	Bajo	Medio	SI	SI	NO	NO
Wuala, Dropbox, Mesh, Omedo, etc.	SI	SI	Alto	Medio	SI	SI	SI	NO
PAST, CFS, Bunshin.	SI	SI	Alto	Bajo	SI	SI	SI	NO

Tabla 1: Algunas características de los principales sistemas de almacenamiento de ficheros.

El significado de las columnas de la tabla anterior se describe a continuación.

- **Catálogo Distribuido**: hace referencia a la existencia de un sistema de catálogo centralizado o distribuido. Por catálogo de ficheros se entiende a la estructura lógica que permite identificar un fichero en el sistema (espacio virtual de nombres) y obtener ciertos metadatos asociados al mismo. Existen ciertos sistemas de almacenamiento puramente distribuidos que permiten omitir el sistema de catalogación ya que utilizan la técnica de multidifusión de mensajes para determinar la existencia de un determinado fichero, no obstante son sistemas con una funcionalidad muy limitada y poco escalables debido a su ineficiencia cuando el número de ficheros es grande.

  En referencia a esta característica, existen pocos sistemas que utilicen un modelo íntegramente distribuido de catalogación de ficheros y la mayoría utilizan una aproximación consistente en utilizar unas pocas máquinas, interconectadas con una topología en árbol, para gestionar el catálogo del sistema.

- **Soporte de Replicación**: es la capacidad de un sistema para realizar copias de un mismo fichero en diferentes nodos de almacenamiento. Por medio de la replicación se puede conseguir un mayor grado de tolerancia a fallos y un acceso más eficiente a los ficheros ya que si el sistema cuenta con un mecanismo de replicación adecuado, las copias pueden ser almacenadas estratégicamente permitiendo ser accedidas desde nodos cercanos a ellas.

  Todos los sistemas analizados cuentan con un mecanismo de replicación básico, sin embargo ninguno de ellos implementa un algoritmo inteligente de replicación que permita almacenar las copias atendiendo a determinadas políticas definidas por la organización propietaria del sistema.

  Algunas posibles políticas de replicación podrían ser las siguientes:

  Replicación atendiendo a la tipología de los ficheros. Una organización puede considerar importante realizar un número de réplicas determinado por el tipo de fichero a almacenar. Así sería posible aumentar o disminuir el número de copias que se mantienen en el sistema dependiendo a una determinada clasificación de los ficheros.

  Replicación en zonas seguras de red. Si todas las réplicas de un fichero se mantienen sobre nodos pertenecientes a un mismo segmento de red, es muy probable que un fallo en la red provoque la inaccesibilidad a cualquier copia del fichero. Una política de replicación sencilla

para evitar esto podría ser la de enviar cada una de las copias de un fichero a nodos de almacenamiento en redes independientes.

Replicación atendiendo a la proximidad de las fuentes de datos. Una organización en el que los clientes acceden de manera continuada a los ficheros, podría priorizar la eficiencia en el acceso a sus ficheros incorporando una política que seleccione de manera específica los nodos sobre los que debe replicarse un fichero. Si es posible conocer mediante una estimación estadística qué nodos son accedidos con más frecuencia para recuperar un determinado fichero, entonces sería adecuado mantener una copia del fichero sobre ese nodo con el objetivo de mejorar eficiencia en las transferencias.

- **Tolerancia a Fallos**: la tolerancia a fallos de un sistema de almacenamiento puede ser evaluada en tres niveles de efectividad dependiendo del tipo de mecanismo de resguardo de información que ofrezcan. Así, sistemas como *Hadoop* o *Microsoft's* DFS poseen servidores secundarios encargados de duplicar la información almacenada en el servidor principal. Se trata de un sistema de baja tolerancia a fallos ya que un fallo en el servidor principal supondría la inoperatividad del sistema, necesitando el tiempo suficiente para reconfigurar manualmente el sistema y poner el servidor secundario en funcionamiento. Por otra parte no se garantiza que todos los ficheros estén disponibles después del reinicio ya que los servidores secundarios sólo realizan operaciones de resguardo cada cierto intervalo tiempo.

  Otros sistemas como Globus y LCG implementan mecanismos que delegan sobre un nodo secundario el control del sistema de almacenamiento cuando un fallo ocurre en uno de sus nodos primarios. Debido a la estructura de almacenamiento con topología de árbol de dominios, es posible delegar sobre cada nodo hijo las operaciones de copia de seguridad. A pesar de lo anterior, las operaciones de respaldo se realizan con cierta demora en el tiempo y por esto no es posible garantizar una recuperación completa del sistema de fichero tras un fallo

  Generalmente la mayoría de sistemas estructurados y plenamente distribuidos, permiten ofrecer un alto grado de tolerancia ya que el catálogo de ficheros se distribuye entre todos los nodos del sistema, lo que permite arbitrar mecanismos de recuperación cooperativos cuando uno de los nodos falla. Aún así, es necesario destacar que muy pocos de estos sistemas ofrecen una tolerancia a fallos completa en tiempo real. Esto quiere decir, que existe acceso ininterrumpido al sistema aún cuando uno de los componentes del sistema falla y además el sistema puede restablecerse en un estado similar a antes de producirse el fallo.

- **Integración en el Grid**: este parámetro hace referencia al ámbito de aplicación para el que está concebida la herramienta de almacenamiento. La mayoría de los sistemas se diseñan para ofrecer soporte de almacenamiento en un ámbito muy particular, por ello es difícil integrar y adaptar estos sistemas para contextos más diversos de aplicación. Este parámetro mide el grado de adaptabilidad e integración de la herramienta de almacenamiento a una plataforma Grid.

- **Replicación Coherente**: en los sistemas con un catálogo de ficheros distribuido hay que tener presente que todas las réplicas de los ficheros deben estar actualizadas respecto al último cambio realizado. Por otra parte es necesario implementar mecanismos de bloque o que impidan la modificación de un fichero cuando el fichero ya esté siendo utilizado por otro usuario. Esta última característica es implementada por unos pocos de los sistemas analizados.

- **Búsquedas deterministas:** otra característica esencial en un sistema de almacenamiento distribuido, es que el resultado de una búsqueda de un fichero en el sistema debe ser determinista o lo que es lo mismo cuando un cliente solicita la presencia de un determinado fichero, el sistema debe arrojar un resultado positivo o negativo pero en ningún caso puede dejar la búsqueda indeterminada por fallos o un mal diseño de su sistema de catálogo. Si esta circunstancia ocurriese, el sistema de fichero podría volverse inconsistente porque podría suponer la pérdida de determinados ficheros que estuvieran previamente almacenados en el

sistema. En el caso de los sistemas de replicación de Globus y LCG, son las propias aplicaciones las encargadas de actualizar las referencias a las réplicas que se publican en los catálogos LRC (*Local Replica Catalog*) y por tanto realizar la búsqueda de un determinado fichero implica asumir la posibilidad de que el fichero ya no exista a pesar de estár publicado en alguno de los catálogos de replicación.

- **Sistemas P2P estructurados**: dentro de los sistemas distribuidos existen determinados soportes de almacenamiento que utilizan un modelo *peer to peer* (P2P) para compartir los ficheros almacenados. Cuando un sistema P2P utiliza cierta topología para localizar determinados nodos sin requerir de un componente central que actúe de catálogo se dice que es un sistema P2P estructurado; un ejemplo de este tipo de sistemas son las *Distributed Hash Tables* (DHT), un modelo distribuido de localización clave/valor que básicamente consiste en asignar a cada nodo del sistema un rango de valores (ficheros) que es posible localizar de forma determinista.

- **Replicación Inteligente:** este parámetro se refiere a la existencia o no de algún mecanismo para decidir la localización del nodo que debe almacenar una copia del fichero. Muchos de estos sistemas incluyen la posibilidad de decidir el número de réplicas que ha de realizarse de un determinado fichero pero no la localización de las mismas.

  Como se puede observar de la Tabla 1, ninguno de los sistemas analizados posee una política de replicación que permita aproximar los ficheros almacenados a los nodos que permitirán posteriormente su recuperación por las aplicaciones. En un entorno Grid esta característica es sumamente importante ya que las aplicaciones se ejecutan sobre recursos asignados dinámicamente y por tanto es importante para mejorar la eficiencia de las transmisiones, determinar cuál de los nodos más cercanos al recurso Grid contendrá una réplica del fichero.

2.3 Requerimientos del Software Fura respecto al Almacenamiento Distribuido de Ficheros.

Fura es un software propietario desarrollado con el objetivo de servir de plataforma Grid para la ejecución de aplicaciones sobre recursos computacionales heterogéneos. Este apartado se limita a describir los principales requerimientos del software Fura en cuanto a almacenamiento se refiere, así pues, no se hará referencia al resto de características y funcionalidades del software que serán brevemente analizadas en el capítulo siguiente.

La arquitectura de Fura ha sido diseñada en forma modular por medio de la implementación de conectores (o plugins) que ofrecen diferente funcionalidad al sistema. Así por ejemplo, existe un plugin denominado IXOS que se encarga de resolver las tareas de almacenamiento y gestión de los ficheros dentro del sistema Fura.

IXOS está implementado para ser utilizado de un modo centralizado, esto supone, que el almacenamiento y mantenimiento de los ficheros se realiza sobre una única máquina que es utilizada además como servidor principal del sistema Fura.

El volumen de ficheros almacenados sobre un servidor Fura puede oscilar enormemente dependiendo de la tipología del Grid sobre el que se despliega la herramienta. Por este motivo y para estimar la capacidad de almacenamiento que debería soportar el sistema propuesto, se analizaron los requerimientos de los principales sistemas Grid que hacen uso de Fura.

En los entornos Grid los recursos son compartidos atendiendo a ciertas políticas de seguridad que controlan el acceso y disponibilidad de los mismos. Una Organización Virtual (OV) [24] es un grupo que comparte los mismos recursos computacionales bajo ciertas políticas bien conocidas por ellos

La Tabla 1 representa los umbrales de requerimientos que se han observado en los principales despliegues realizados sobre el software Fura.

Los valores siguientes representan una estimación de la capacidad de almacenamiento requerida por las principales OV's con un tamaño de actividad pequeño, medio y grande. Además de esto, para medir el nivel de actividad de la organización, se ha tenido en cuenta tanto el tamaño como el número de ficheros que han debido almacenarse después de un periodo largo de producción. Los resultados obtenidos se representan a continuación.

	Tamaño total de ficheros previsiblemente requerido por la OV	Número de ficheros previsiblemente requerido por la OV
OV pequeña	100 MB	1.000
OV mediana	30 GB	10.000
OV grande	200 GB	35.000
OV extra grande	1 TB	50.000

Tabla 1. Requerimientos de almacenamiento Fura para distintas OV.

Los resultados anteriores reflejan la necesidad de poseer un sistema de almacenamiento que consiga balancear la carga entre varios nodos. Para organizaciones de tamaño grande y medio quizás un único servidor puede ser suficiente para hacerse cargo del almacenamiento de los ficheros Fura, pero cuando las organizaciones poseen un tamaño de actividad más grande la utilización de modelos centralizados provoca ciertas limitaciones de escalabilidad, además de obtener un sistema menos tolerante a fallos.

Capítulo 3 – Introducción a Fura.

3.1 Descripción General del Sistema.

Fura [19] es un planificador Grid que permite asignar la ejecución de tareas a un conjunto de recursos computacionales que pueden funcionar sobre plataformas heterogéneas. Desde el punto de vista del usuario, el proceso de planificación y ejecución de las tareas es transparente ya que el usuario sólo necesita introducir ciertos datos de su aplicación, por medio de una interface web, y esperar a que se ejecuten en el Grid.

En éste apartado se describirá brevemente los principales componentes del software Fura con el propósito de entender mejor la arquitectura de almacenamiento Fura y por tanto descubrir las principales deficiencias resueltas por la arquitectura propuesta en esta tesina.

Básicamente existen tres componentes principales en la arquitectura Fura:

- **Portal**: el usuario interactúa con un interfaz web definiendo las características de las tareas que necesita ejecutar. Por medio del portal de usuario es posible especificar parámetros como ficheros de entrada, ejecutable, parámetros de ejecución, requerimientos mínimos, etc. El componente portal de Fura utiliza la tecnología SOAP [25] para realizar las comunicaciones con el servidor. El portal permite monitorizar las tareas lanzadas al Grid y ofrece una interfaz para que el usuario pueda recuperar los ficheros generados tras la finalización de la ejecución de las tareas.

- **Servidor**: es el componente central de Fura y se encarga de la totalidad de las actividades relacionadas con las planificación, ejecución, monitorización y gestión de los recursos que integran el Grid. Cada servidor Fura posee un repositorio local que se utiliza para almacenar de modo estructurado toda la información relacionada con las tareas que se ejecutan en el Grid. Esta información se almacena en forma de ficheros y puede incluir ficheros de entrada necesarios para la ejecución de la tarea, ficheros generados por cada una de las ejecuciones y ficheros de control que introduce Fura para gestionar la ejecución de una tarea.

- **Agentes**: se denominan así a los recursos computacionales encargados de ejecutar las tareas ofrecidas por el servidor Fura. Un servidor Fura gestiona un conjunto de agentes montados sobre plataformas diferentes proporcionando mayor escalabilidad al sistema, ya que es posible ejecutar una misma tarea sobre diferentes plataformas computacionales de un modo parcialmente transparente al usuario.

  Los agentes Fura se registran en el servidor cuando se inician y posteriormente solicitan al servidor la ejecución de nuevas tareas. Hay que destacar que se trata de un modelo de funcionamiento bajo demanda muy eficiente, ya que permite liberar al servidor de la tarea de monitorizar la actividad de los agentes cuando tienen un nivel de carga lo suficientemente bajo. Del mismo modo este mecanismo de petición de tareas a cargo de los clientes posibilita la idea central del desarrollo de Fura, es decir el aprovechamiento de los ciclos perdidos de computación cuando el recurso se encuentra en un estado ocioso[6].

Una vez se han ejecutado las tareas sobre los agentes Fura, los ficheros generados son almacenados directamente sobre el servidor utilizando los protocolos de paso de mensajes y transferencia

[6] Para medir esta característica sobre un recurso, Fura evalúa la carga del sistema en un periodo de tiempo y considera al recurso en un estado ocioso cuando la carga total del sistema no supera un determinado valor.

transferencias SOAP o FTP. La Fig. 2 muestra un esquema global del modelo de ejecución de tareas en Fura.

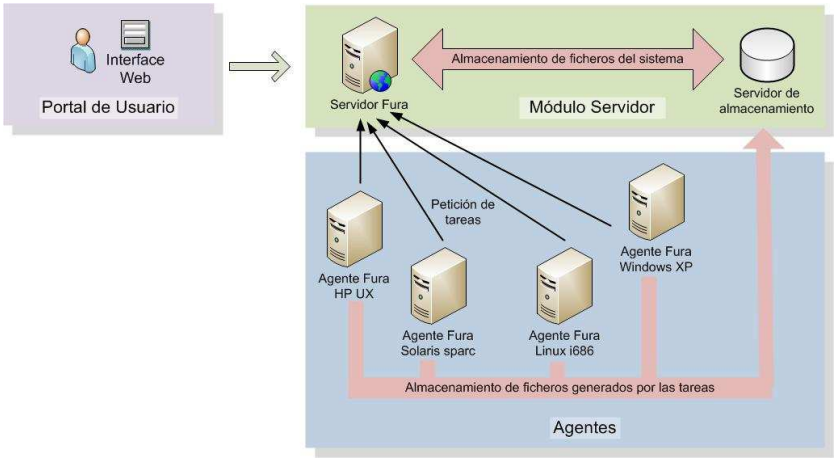

Fig. 2 Esquema general de la ejecución de tareas en Fura.

A continuación se explica brevemente los pasos que ocurren en un sistema Fura desde que el usuario solicita la ejecución de una tarea y se retornan los resultados de dicha ejecución.

El usuario accede por medio de una interfaz web a la herramienta cliente Fura. Después, el usuario debe autentificarse por medio de un mecanismo de usuario/contraseña que le permite utilizar las funcionalidades del servidor Fura. El portal de usuario permite acceder a distinta funcionalidad de Fura dependiendo del rol asignado a cada usuario, de este modo, los usuarios pueden usar diferentes funciones de Fura dependiendo de si son administradores, desarrolladores o simplemente usuarios que requieren ejecutar tareas ya definidas previamente.

El proceso de ejecución de tareas en Fura comienza cuando un usuario de tipo administrador o desarrollador, define un módulo de tarea que describe las propiedades más genéricas de sus tareas. Por medio de un módulo de tarea es posible definir posteriormente tareas con rangos de ejecución diferentes.

La Fig. 3 muestra la pantalla de configuración de un módulo de tarea. El usuario debe especificar el nombre del módulo, los requerimientos físicos de las tareas (memoria, disco, cpu, etc.) y un fichero binario que se ejecutará para cada una de las plataformas que se especifique. Fura admite un conjunto muy amplio de las principales plataformas de sistemas computacionales sobre las que puede ejecutar tareas. Como se describirá más adelante, el planificador asigna tareas a los agentes Fura independientemente del tipo de plataforma que posean. Los agentes Fura solicitan la ejecución de tareas al servidor cuando quedan libres y el servidor proporciona a los agentes el binario adecuado a la plataforma sobre la que se ejecutará la tarea.

Dentro de la pantalla de módulo de tareas el usuario puede especificar el rango de ejecución sobre el que una tarea va a ser ejecutada. Existen diferentes modos de especificar rangos sobre Fura pero para simplificar la explicación se expondrá un ejemplo representativo:

Sleep.sh 1 100 1
Ejecutable: sleep.sh
init: I
end: F
step: S

El ejemplo anterior permite iterar un ejecutable en un rango que será especificado posteriormente por medio de una interfaz web.

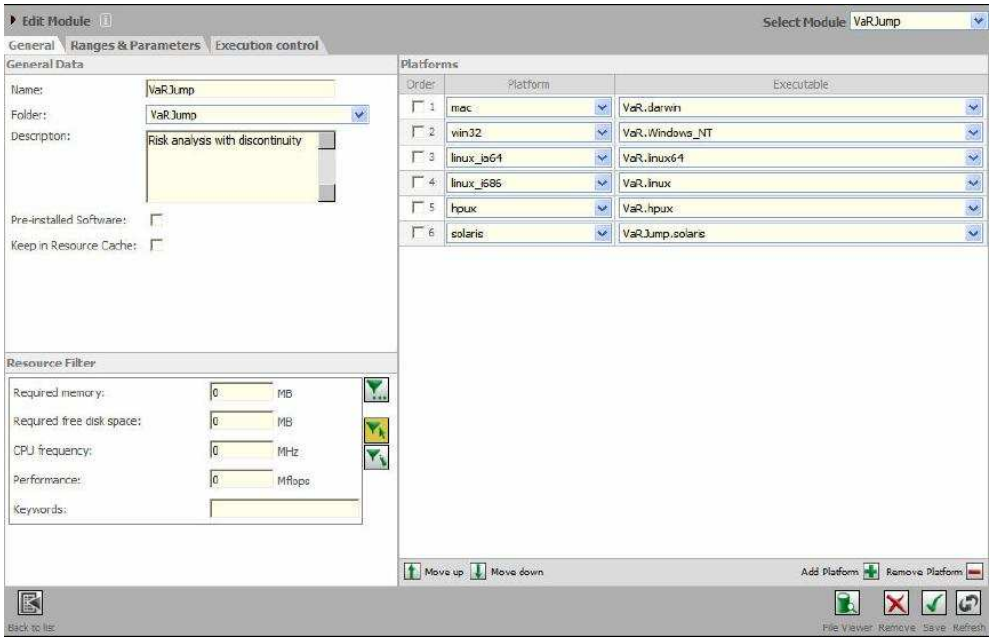

Fig. 3 Interface de usuario para especificar un módulo de tarea Fura.

El siguiente paso es definir para este módulo el conjunto de ejecuciones que el usuario debe lanzar al Grid. La Fig. 4 muestra un ejemplo de especificación de una tarea multiparamétrica. La ejecución de cada una de las combinaciones posibles de los argumentos es definida por Fura como microtarea y de modo general cada agente Fura recibe una microtarea para su ejecución.

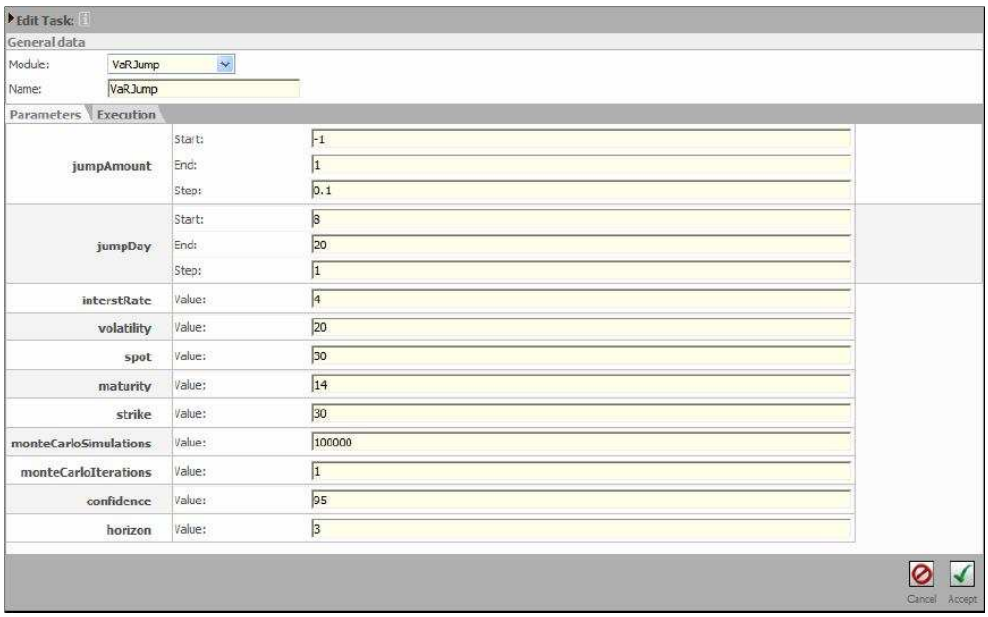

Fig. 4 Pantalla de especificación de tareas.

Finalmente el planificador asigna, dependiendo de la disponibilidad de los recursos, cada microtarea a un agente Fura. La Fig. 5 muestra la ejecución de un conjunto de tareas pertenecientes a diferentes módulos. Por medio de esta interfaz es posible monitorizar el estado de ejecución de las tareas y realizar diferentes tareas como reiniciarlas o pararla. La Fig. 6 muestra la pantalla de información de monitorización para tareas que permite, entre otras cosas, obtener información asociada a los recursos sobre los que se está ejecutando cada microtarea.

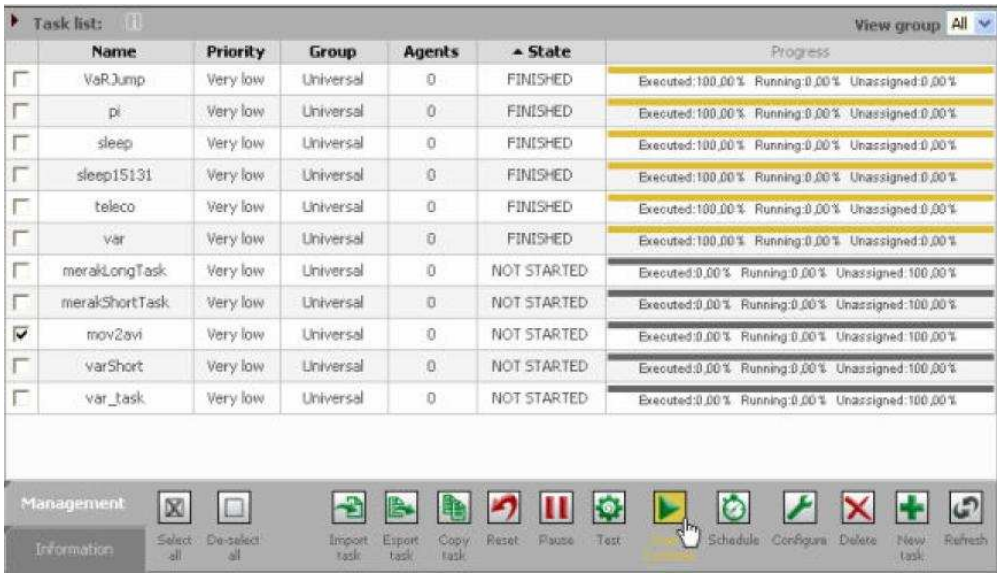

Fig. 5 Pantalla de monitorización de tareas en Fura.

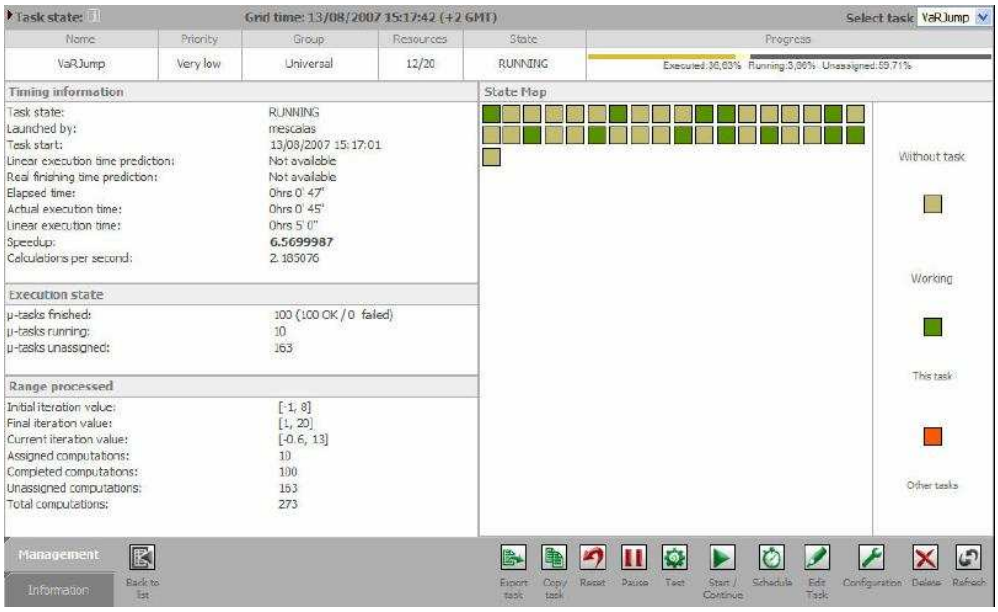

Fig. 6 Estado de ejecución de las microtareas asignadas a una tarea.

3.2 Sistema de Almacenamiento de Ficheros en Fura.

Tal y como se comentó anteriormente, el sistema Fura almacena los ficheros asociados a las ejecuciones de tareas en una ubicación local al servidor. Para llevar a cabo esta tarea, el sistema de almacenamiento Fura se ha diseñado a modo de servicio Web lo que permite a los desarrolladores y al resto de componentes del software interactuar fácilmente con el sistema de almacenamiento.

El módulo encargado de prestar el acceso a la gestión de ficheros en Fura se llama IXOS y se subdivide en dos API que prestan diferente funcionalidad al sistema de ficheros.

Por una parte Fura expone el *ApiFileSystem* que publica un conjunto de operaciones con ficheros similares a la de cualquier sistema de almacenamiento local. Este API es utilizado por el propio servidor para realizar operaciones de mantenimiento de ficheros a nivel local (copiar, listar, crear, borrar, etc.) y casi siempre son ejecutadas como respuesta a una petición del usuario por medio del interface Web.

Fura expone una segunda API referida como *ApiFileSystemIO* que utilizan los agentes Fura para realizar transferencias de ficheros desde una máquina remota al servidor. La Tabla 2 y la Tabla 3 recogen un listado de todas las operaciones que implementan estas dos librerías.

getHome()
chmod(String path, FilePermissions[] permissions)
chmodAdd(String path, FilePermissions[] permissions, boolean recursive)
chmodRemove(String path, FilePermissions[] permissions, boolean recursive)
chmodTree(String path, FilePermissions[] permissions)
getAvailableRepositorySpace()
copy(String src, String dst, boolean overwrite)
createZip(String[] paths)
decompressZip(String pathZip, String pathDest)
exists(String path)
removeFile(String path)
eraseLock(String path)
ls(String path)
lsACL(String path)
lsInfo(String path)
mkdir(String directory)
mkdirs(String directory)
readDir(String directory)
readDirACL(String directory)
readDirInfo(String directory)
rename(String oldName, String newName)
rmdir(String directory)
rmTree(String directory)
touch(String path)
readDirInfoStart(String directory, int orderBy, int orderDirection, boolean block)
readDirInfoGet(long id, int from, int count)
readDirInfoEnd(long id)
readDirInfoCount(long id)

Tabla 2 Api IXOS (ApiFileSystem) operaciones locales en el servidor Fura.

openFile(String path, int mode, boolean createSecFile)
closeFile(FileHandler f)
readFile(FileHandler f, int count)
readAttach(FileHandler f, int count)
downloadFile(String fileName)
skipFile(FileHandler f, int count)
seekFile(FileHandler f, long pos)
markFile(FileHandler f)
resetFile(FileHandler f)
writeFile(FileHandler f, byte[] buffer)
writeAttach(FileHandler f, DataHandler data)
truncateFile(FileHandler f, long size)
uploadFile(String fileName, DataHandler dataHandler, boolean createSecFile)

Tabla 3 Api IXOS (ApiFileSystemIO) transferencia de ficheros con el servidor Fura.

A pesar de que los plugins de almacenamiento de Fura exponen un conjunto extenso de operaciones, existen dos operaciones que se invocan un número de veces muy superior respecto al resto. La operación *download* y *upload* permiten a los nodos agente descargar o almacenar respectivamente un fichero desde el servidor Fura.

La Fig. 7 permite observar la interacción entre los distintos componentes de Fura para el almacenamiento de ficheros. Los agentes Fura utilizan las operaciones *uploadFile* y *downloadFile* para subir y bajar ficheros al servidor. Por su parte, el usuario por medio de la interface web puede realizar operaciones básicas con los ficheros y directorios del sistema.

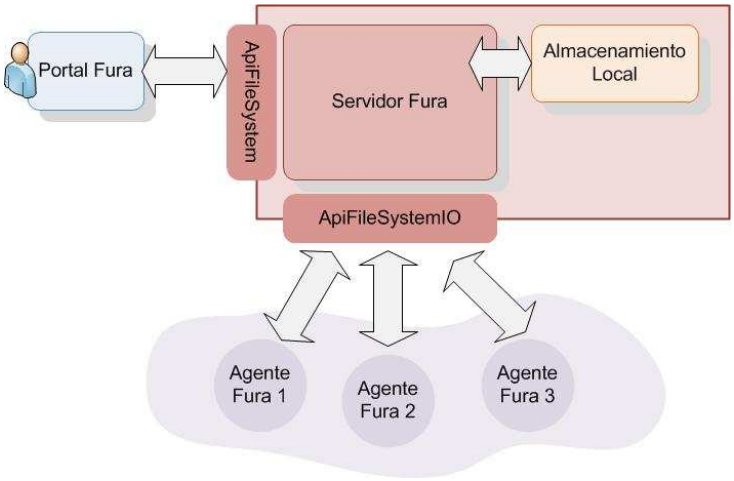

Fig. 7 Esquema de almacenamiento Fura orientado a servicios Web.

Capítulo 4 – Propuesta de una Arquitectura Distribuida de Almacenamiento de Ficheros (DFS).

4.1 Introducción.

En el capítulo de estado del arte, se proporcionó una visión general a los principales sistemas de almacenamiento de ficheros que actualmente se utilizan en diferentes entornos Grid y distribuidos. En este sentido y analizando el caso de los entornos Grid, se ha producido una gran evolución respecto a los requerimientos de almacenamiento demandados por los sistemas actuales.

Inicialmente las aplicaciones Grid se orientaron hacia el aprovechamiento de la capacidad de cálculo que ofrecía la compartición de recursos computacionales. Sin embargo, en la actualidad el volumen de información que generan las aplicaciones Grid es cada vez mayor y los recursos de almacenamiento son limitados. Por ello, es importante disponer de un sistema de almacenamiento distribuido que permita absorber el gran volumen de información que generan las aplicaciones y que es susceptible de ser almacenada.

De este modo, la necesidad de disponer de sistemas que proporcionen herramientas para el almacenamiento distribuido supone actualmente un elemento de igual o mayor importancia a la de disponer de gran capacidad de cálculo.

La principal motivación para la implementación de los sistemas de almacenamiento anteriormente descritos, es ofrecer soporte para el almacenamiento y recuperación de grandes volúmenes de datos. Sin embargo la mayoría de estos sistemas poseen un diseño muy centralizado que les limita dotándoles de características poco deseables en cuanto a escalabilidad y tolerancia a fallos.

En los sistemas de almacenamiento centralizados o jerarquizados, el acceso a los datos se realiza por medio de uno o varios servidores conectados a un dispositivo local de gran capacidad de almacenamiento. Este modelo permite a los usuarios del sistema tener acceso a los ficheros de un modo simple. Sin embargo, cuando la cantidad de usuarios que hacen uso del sistema se incrementa de modo considerable, pueden aparecer problemas de saturación en el acceso a los ficheros quedando el sistema temporalmente inaccesible.

Una de las características más importantes que debe proporcionar un sistema de almacenamiento distribuido es que la accesibilidad a los ficheros debe ser permanente en el tiempo. Cuando un servidor de almacenamiento falla (o los dispositivos de almacenamiento asociados a él) se requiere un cierto tiempo para poder recuperar los ficheros y dejar estable de nuevo al sistema. Por eso, un modelo centralizado no puede ser utilizado para entornos grandes de usuarios y/o volumen de datos extenso como es el caso de los entornos Grid.

Los sistemas Grid y en general cualquier sistema que opere con volúmenes grandes de ficheros y usuarios, necesita disponer de herramientas de almacenamiento de ficheros que ofrezcan características como tolerancia a fallos, alta disponibilidad, cercanía de los datos a las aplicaciones que los utilizan, rendimiento, seguridad en el acceso a los datos y escalabilidad.

Los sistemas que existen actualmente proporcionan alguna de estas características pero carecen del resto principalmente en lo que se refiere a la tolerancia a fallos de sus componentes o por ejemplo la elección adecuada de la ubicación de los ficheros. Precisamente, esta última característica es uno de los puntos que determinan el buen rendimiento del sistema en el acceso a los ficheros previamente almacenados. Es importante contar con un sistema que tenga en cuenta la localización de los ficheros almacenados ya que el tiempo invertido por las aplicaciones para acceder a los ficheros determinará el rendimiento global del sistema.

A pesar de que existen herramientas que ofrecen modelos de replicación de ficheros y permiten su restauración en caso de fallo, el almacenamiento de los ficheros se realiza aleatoriamente entre los componentes del sistema sin considerar la cercanía de los datos a las aplicaciones clientes.

Este documento describe la arquitectura de un sistema de almacenamiento distribuido que ofrece soporte para los principales entornos Grid y Cloud. La arquitectura propuesta añade nuevas características respecto a los principales sistemas de almacenamiento y por tanto trata de superar las limitaciones que imponen estos sistemas .

4.1.1 Planteamiento de una Arquitectura Adecuada a los Requerimientos de Almacenamiento de los entornos Grid y Cloud.

El sistema que se propone en este documento tiene como objetivo proporcionar un sistema de almacenamiento distribuido de ficheros con características como tolerancia a fallos, escalabilidad, acceso continuado y consistente al sistema de ficheros y replicación inteligente y adaptable por medio de funciones heurísticas.

Como se comentó anteriormente, en los sistemas con una arquitectura centralizada, una máquina es la encargada de ofrecer el servicio de almacenamiento de los ficheros y por tanto un posible fallo del dispositivo de almacenamiento del servidor puede impedir que los ficheros estén accesibles durante bastante tiempo.

Muchos de los sistemas que implementan este modelo poseen servidores de respaldo que permiten recuperar los ficheros desde copias de seguridad secundarias; sin embargo este proceso no es inmediato y requiere la intervención manual del administrador del sistema impidiendo el acceso al sistema de ficheros durante un tiempo indeterminado. Además de lo anterior, mantener copias desactualizadas en servidores secundarios puede ocasionar inconsistencias en el sistema de ficheros una vez finaliza el proceso de restauración.

Este trabajo plantea el diseño de un sistema de almacenamiento de alta disponibilidad en el acceso a los ficheros ya que el fallo en alguno de los componentes de almacenamiento no influirá de ningún modo en la disponibilidad ni la integridad del sistema de ficheros.

La arquitectura propuesta permite que los ficheros sean accesibles desde cualquier punto del sistema distribuido proporcionando al usuario una visión similar a la de un sistema de almacenamiento local. Para posibilitar esto, la arquitectura define una serie de operaciones básicas que puede utilizar el usuario para realizar las transferencias de los ficheros.

Se trata de un sistema adaptativo al volumen de datos soportado ya que es fácilmente escalable con la supresión o incorporación de nuevos componentes de almacenamiento sin que esto afecte al normal funcionamiento del sistema.

Una de las características principales de la arquitectura propuesta es la de abstracción en el acceso a los datos. Muchos de los sistemas de almacenamiento distribuido requieren que el usuario o la aplicación que accede al sistema deba conocer la ubicación física dónde se encuentra el fichero que se quiere recuperar; por otra parte y tratándose de sistemas distribuidos se necesita conocer la identificación del fichero que casi siempre estará relacionada con la dirección física del nodo que almacena el fichero.

Para facilitar el acceso a las aplicaciones se propone un sistema de almacenamiento que permita acceder a los ficheros como si de un sistema de almacenamiento local se tratase. Para ello se define un sistema de identificación de ficheros único y global que permite a las aplicaciones clientes identificar un fichero dentro del sistema sin conocer la ubicación física del mismo.

Con el objetivo de proporcionar tolerancia a fallos en los componentes de almacenamiento del sistema, los ficheros serán replicados en localizaciones concretas teniendo en cuenta cierta función heurística

definida por el propio sistema de almacenamiento; no obstante el usuario podrá seleccionar el número de copias del fichero que se almacenarán en el sistema. Por medio de la replicación de ficheros se consigue proporcionar al sistema de un medio de recuperación tras un fallo de alguno de sus nodos, además el modelo de replicación es utilizado para seleccionar y almacenar las copias de los ficheros en localizaciones estratégicas que permitan a las aplicaciones recuperarlos de la forma más eficiente posible.

A diferencia de otras herramientas de almacenamiento distribuido, la arquitectura plantea un módulo encargado de decidir cada una de las ubicaciones físicas dónde se almacenan las réplicas de un fichero. Las réplicas de los ficheros son almacenadas cerca de sus fuentes de datos lo que quiere decir que se seleccionarán aquellas localizaciones de almacenamiento que más accesos al fichero han tenido durante la actividad del sistema. Para contabilizar los accesos a los ficheros, un módulo estadístico registrará la información histórica de accesos a los ficheros y esta información se utilizará posteriormente para la seleccionar la ubicación de una réplica.

La replicación de ficheros en un sistema distribuido requiere que las copias del fichero se mantengan actualizadas en todo momento y tras cada operación de escritura. El sistema resuelve este problema aplicando un modelo de coherencia de ficheros que permite obtener siempre una copia actualizada de un fichero independientemente del punto de acceso al sistema.

El sistema se completa con un mecanismo de bloqueo para operaciones de escritura que impide el acceso a los ficheros que están siendo actualizados manteniendo así la consistencia el sistema de ficheros.

Como resumen de lo anterior, este trabajo especifica una arquitectura para un sistema de almacenamiento de ficheros completamente tolerante a fallos, con replicación coordinada y con un ámbito de aplicación muy heterogéneo como puede ser una organización Grid/Cloud o un sistema independiente de almacenamiento distribuido. La arquitectura propuesta en esta tesina será referida a partir de aquí por medio de las siglas DFS (***D****istributed* ***F****ile* ***S****ystem*).

4.2 Descripción de los Componentes de la Arquitectura DFS.

Tal y como se describió en capítulos anteriores, utilizar una arquitectura centralizada para el almacenamiento de ficheros puede ser desaconsejable en sistemas con un tamaño de carga medio o grande. Utilizar un modelo plenamente distribuido, permite balancear la carga global del sistema sobre los componentes que lo integran, evitando así situaciones de denegación de servicio o inaccesibilidad temporal al sistema de ficheros.

La arquitectura DFS utiliza un modelo distribuido *P2P,* donde las unidades de almacenamiento que componen el sistema actúan indistintamente sirviendo o almacenando ficheros.

Dentro de los sistemas distribuidos *P2P* se pueden distinguir los modelos estructurados y los no estructurados. En los modelos estructurados cada par de nodos conoce su relación respecto al resto de componentes del sistema y por tanto la coordinación en el intercambio de información entre pares de nodos se realiza sin mediar ningún otro componente. Por el contrario, en los modelos no estructurados se requiere de un componente mediador que permita poner en contacto a cada par de nodos.

La arquitectura DFS utiliza un sistema estructurado *P2P* para el diseño del sistema de localización de ficheros ya que permite realizar búsquedas de ficheros de modo determinista.

La característica más importante de este tipo de arquitecturas *P2P* estructuradas, es que los ficheros permanecen diseminados por todos los componentes del sistema sin que exista un nodo central encargado de la gestión o la coordinación del resto de nodos. Al contrario de esto, cada nodo tiene la función de administrar una porción del espacio global de ficheros que están almacenados y la suma de cada conjunto de ficheros asociados a cada nodo forma lo que se denomina el catálogo global del sistema de ficheros.

De este modo, la arquitectura propuesta distingue por una parte un sistema de catálogo encargado de almacenar la información de metadatos asociada a los ficheros, y por otra el sistema de almacenamiento encargado de salvaguardar el contenido físico del fichero.

En relación a lo anterior, la posibilidad de que los ficheros se almacenen en recursos de almacenamiento físicamente independientes a los nodos de gestión DFS, permite que un posible fallo de un nodo del sistema de catalogación no afecte de ningún modo a los ficheros ya almacenados. Del mismo modo, la inaccesibilidad a un recurso de almacenamiento no impedirá que el sistema de catalogación pueda recuperar los ficheros desde otras posibles ubicaciones.

La arquitectura propuesta permite que los nodos DFS puedan unirse y salir del sistema de un modo dinámico sin que esto suponga una pérdida de las prestaciones del sistema. Precisamente esta característica permite proporcionar al sistema una capacidad de almacenamiento variable y escalable dependiendo de las necesidades puntuales de la organización que lo utiliza.

De modo general en las arquitecturas *P2P,* la supresión de nodos del sistema (por fallo o por cese de su actividad) suele ser habitual en el tiempo. Sin embargo, la arquitectura DFS implementa mecanismos de tolerancia a fallos que impiden situaciones de inestabilidad en el sistema. Para evitar estas situaciones, cuando un nodo falla en el sistema DFS la gestión de los ficheros que gestionaba es asumida por el resto de nodos activos en el sistema.

La Fig. 8 muestra una visión general del funcionamiento del sistema. Las aplicaciones acceden al sistema por medio de la conexión con algún nodo conocido y solicitan una operación de transferencia de ficheros. Posteriormente, el sistema DFS se encarga de transferir el fichero hacia los recursos de almacenamiento más adecuados.

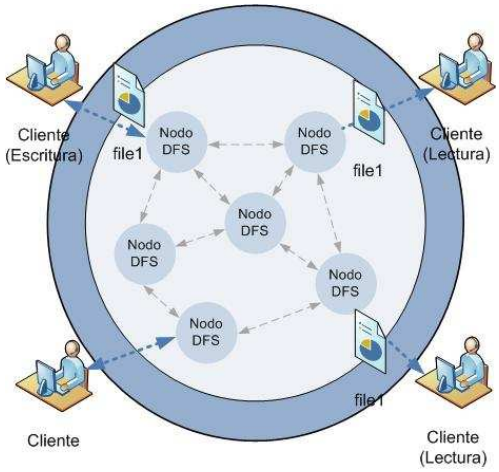

Fig. 8 Visión global del sistema DFS.

Como se observa en la Fig. 8, los nodos DFS interaccionan dentro del sistema asumiendo el rol de cliente o servidor en función de si solicitan o sirven peticiones de transferencia de ficheros. No obstante, la figura anterior representa un esquema meramente orientativo y no debe interpretarse de manera errónea, ya que aunque los nodos se representen de un modo desestructurado, la arquitectura DFS establece cierta estructura lógica en la disposición de los nodos dentro del sistema.

Esta sección describe cada uno de los componentes que integran el sistema DFS. La arquitectura está formada por las tres capas de componentes que se describen a continuación.

- **Nivel de Interfaz**: especifica un conjunto de operaciones que permiten a las aplicaciones realizar transferencias de ficheros dentro del sistema de almacenamiento. El nivel de interfaz define un sistema de nombres de ficheros único y global que permite abstraer a las aplicaciones clientes de la ubicación física del fichero almacenado.

- **Nivel Lógico**: existe un modelo inteligente de replicación de ficheros que almacena las copias de los ficheros cerca de sus futuras fuentes de datos. El sistema de catalogación de los ficheros se mantiene distribuido en el sistema permitiendo independizar su localización física de la información de tipo metadato asociada al fichero. Por medio del nombre de un fichero es posible descubrir, si es que existe, el nodo que almacena su información lógica. Esta información de control se almacena por medio de un sistema de backup lo que permite asegurar la estabilidad del sistema ante fallos de sus componentes. Por último, el sistema permite la entrada y/o salida de nuevos nodos de almacenamiento sin afectar significativamente al rendimiento del sistema ni comprometer la estabilidad del mismo.

- **Nivel de Almacenamiento**: ofrece soporte para almacenar físicamente los ficheros en los nodos del sistema. El nivel de almacenamiento está formado por una serie de plugins que permiten utilizar protocolos de transferencia como FTP, HTTP, GridFTP, sockets, etc. Por medio de esta capa, es posible individualizar el modelo de transferencia que cada componente tiene instalado.

4.2.1 Nivel de Interfaz.

Existen cuatro operaciones básicas que permiten a una aplicación almacenar, recuperar, eliminar y listar un conjunto de ficheros pertenecientes al sistema de almacenamiento.

Además de estas operaciones, se definen un grupo de operaciones internas al sistema que son accesibles sólo por los nodos de almacenamiento para realizar tareas de gestión y administración. Se detalla a continuación la funcionalidad de cada una de las operaciones agrupadas dependiendo de su nivel de acceso.

Operaciones accesibles por parte del cliente (cliente - DFS):

- **put**(*String localName, String virtualName, integer nReplicas*): realiza una transferencia de un fichero desde el sistema de local de ficheros del cliente hacia el sistema de almacenamiento distribuido. La operación es invocada por la aplicación cliente y los parámetros requeridos son *localName*, es la ruta del fichero que está almacenado localmente; *virtualName*, nombre que recibirá el fichero en el espacio de nombres del sistema de almacenamiento; *nReplicas*, indica el número de copias del fichero que deberán permanecer en el sistema de ficheros.

- **get**(*String localName, String virtualName*): la aplicación cliente invoca esta operación para recuperar un fichero desde el sistema de almacenamiento. Los parámetros requeridos son *localName*, ruta local a la aplicación cliente desde dónde se obtiene el fichero; *virtualName*, nombre del fichero a recuperar que debe existir dentro del espacio de nombres global.

- **del**(*String virtualName*): el fichero se elimina del sistema de almacenamiento distribuido. Se necesita identificar al fichero con su nombre en el espacio de nombres DFS.

- **show**(*String pattern, int depth*): permite a las aplicaciones solicitar información acerca de cualquier fichero en el sistema que concuerde con un determinado patrón de búsqueda. La búsqueda es determinista, lo que significa que si existe algún fichero que concuerde con el filtro aplicado por el cliente, entonces el fichero existe en el sistema. En caso contrario significa que el fichero no está almacenado en el sistema DFS. Los parámetros de entrada son *pattern*, filtro de búsqueda; *depth*, para sistemas con un espacio de nombres jerarquizado en directorios, indica el nivel máximo de búsqueda considerando '/' como marca de directorio. Un ejemplo de búsqueda dentro de una jerarquía de una estructura de directorios sería, show("/dir1/*",1) devolviendo la lista de nombres de fichero que pertenecen al directorio "/dir1" y que además no poseen subdirectorios.

Operaciones internas entre nodos (DFS - DFS):

Operaciones básicas	
putDFS(localName, virtualName, nReplicas)	Transferencia interna de almacenamiento de ficheros entre nodos DFS. Al igual que en una operación *put* el nodo cliente indica el número de copias del fichero.
getDFS(virtualName)	Transferencia interna para la recuperación de un fichero entre dos nodos DFS.
delDFS(virtualName)	Eliminación de un fichero del sistema.
Operaciones de búsqueda:	
getOwner(key)	Devuelve el identificador del nodo encargado de gestionar un determinado fichero.
joinRing(idNewNode, ipNewNode)	Un nuevo nodo DFS solicita a otro nodo la unión al sistema indicando su identificador global y su dirección IP.
neighborOperation(type, nodeIP, recursive)	Realiza una operación de mantenimiento del sistema de búsqueda. Existen tres posibles operaciones, añadir un nuevo nodo conocido; cambiar la referencia al nodo vecino inferior o superior; Las operación de añadir un nuevo nodo al sistema implica que los k vecinos inferiores deban conocer la existencia de este nuevo nodo. El parámetro recursive propaga la operación sobre los k nodos vecinos inferior.
statusRing(ipFirstNode)	Devuelve la lista de los nodos que forman el sistema anillo de búsqueda y monitoriza la conexión entre ellos. En caso de existir un

	problema de conectividad entre nodos la operación devuelve el último nodo conectado al anillo.
manageRing(type, recursive)	Cambia el estado de un nodo del anillo de búsqueda permitiendo retransmitir la operación a otros nodos recursivamente. La operación permite establecer los estados de *starting, building, online, recovery y shutdown*. Dependiendo del estado en que se encuentre algunas operaciones en el nodo son deshabilitadas.
notifyNode(ipFailedNode)	Indica que un nodo está fuera del sistema y debe eliminarse cualquier referencia a él.
keepAlive()	El nodo vecino con identificador superior invoca esta operación cada cierto tiempo para indicar que sigue activo. Si tras un tiempo determinado no se recibe una operación *KeepAlive,* el nodo determina un fallo de conectividad o funcionamiento de su vecino e inicia el protocolo de reestablecimiento del anillo de búsqueda.
Operaciones de catalogación:	
backupCatalog(blockFiles, delOldCatalog)	Obtiene la información de metadatos de un conjunto de ficheros que proceden de un nodo vecino y lo almacena en el repositorio de backup. El segundo parámetro indica si ha de eliminarse previamente los ficheros almacenados en el repositorio de backup.
rebuild(blockFiles)	Reconstruye el sistema de catálogo con un nuevo bloque de ficheros. Esta operación permite iniciar un proceso de actualización del catálogo descartando ficheros que pertenecen a la administración de un nodo diferente.
Operaciones de Transferencia:	
transfer(type, virtualName)	Permite obtener los parámetros de conexión con el dispositivo que almacena el fichero que se indica como argumento. Los nodos DFS transfieren o recuperan el fichero directamente a partir de la cadena de conexión que se obtiene.

4.2.2 Nivel de Almacenamiento.

El nivel de almacenamiento define un conjunto de componentes que permite a las aplicaciones y a los nodos DFS abstraer los mecanismos internos de transferencia de los ficheros; además los componentes poseen un diseño modular que permite ampliar el conjunto de protocolos de transferencia con los que es posible enviar y/o recibir ficheros.

Cuando se necesita transferir un fichero entre dos nodos se produce una comunicación previa entre los módulos de almacenamiento de ambos nodos. El nodo que inicia la comunicación solicita los parámetros de transferencia al nodo destino y este último nodo, responde enviando el tipo de protocolo que se deberá utilizar para transferir el fichero. Para realizar la transferencia el nodo destino devuelve los parámetros de conexión que permiten al nodo origen obtener el fichero directamente desde un dispositivo de almacenamiento. La cadena de conexión depende del tipo de protocolo que se utiliza, por lo que no es necesario que el nivel de almacenamiento deba conocer los detalles de implementación del protocolo de transferencia a utilizar.

La Fig. 9 representa la comunicación que se establece a nivel de almacenamiento en una operación de transferencia de un fichero. El nodo 1 solicita una operación de transferencia de fichero al nodo 2. En respuesta, el nodo 2 devuelve el tipo de protocolo que se debe utilizar para enviar el fichero y además se envía la cadena con los parámetros de conexión requeridos para que el nodo 1 pueda transferir el fichero.

En el caso del ejemplo mostrado por la Fig. 9Fig. 9, el protocolo utilizado es FTP por lo que debe utilizarse un módulo de transferencia tipo FTP para realizar la conexión. Este módulo obtiene los parámetros de conexión para conectar con el dispositivo de almacenamiento y transferir el fichero.

Es importante resaltar que la operación de transferencia se realiza directamente con el dispositivo de almacenamiento (servidor FTP) que almacena el fichero involucrado en la operación y no interviene ningún elemento adicional en el sistema.

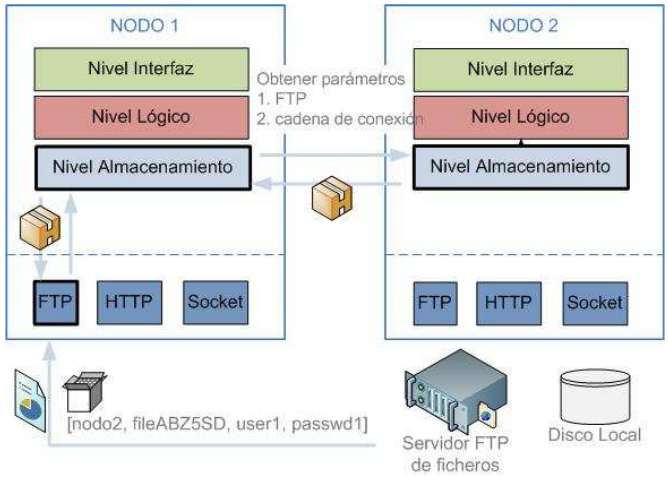

Fig. 9 Comunicación en el nivel de almacenamiento.

4.2.3 Nivel Lógico.

Los componentes del nivel lógico forman el núcleo central del sistema DFS y proporcionan la funcionalidad al nivel de interfaz para administrar el almacenamiento de ficheros en el sistema. A continuación se detalla brevemente cada uno de los módulos que componen el nivel lógico en un nodo DFS.

- **Servidor/Instanciador**: es el elemento que permanece activo en el sistema atendiendo operaciones de transferencia. Cada petición de operación de transferencia es atendida en paralelo por un proceso diferente.

- **Kernel**: coordina los módulos del nivel lógico para llevar a cabo toda la lógica del sistema. Establece los estados de actividad del nodo y habilita o deshabilita la ejecución de determinadas operaciones dependiendo del estado en que se encuentre el nodo. El componente kernel es el encargado de monitorizar la actividad en un nodo DFS y llevar las acciones oportunas para mantener la estabilidad en el nodo ante un posible fallo o reconfiguración del sistema.

- **Módulo de Operaciones**: implementa la lógica de cada una de las operaciones que dan funcionalidad al sistema. Como se vio anteriormente las operaciones tienen un carácter interno o externo dependiendo de si han de ser invocadas por los propios nodos del sistema o por una aplicación externa.

- **Módulo de Comunicaciones**: proporciona un conjunto de operaciones que permiten realizar la comunicación entre nodos por medio del envío de mensajes de control.

- **Sistema de Catálogo**: el sistema de catálogo almacena toda la información de tipo metadato asociada a los ficheros que un nodo administra. Esta información de control permite establecer la ubicación exacta de las réplicas de cada fichero además de su información de contexto. En los siguientes apartados se explicará el modelo de datos del catálogo.

- **Sistema de Búsqueda**: el módulo de búsqueda permite determinar qué nodo puede contener información de tipo metadato relacionada con el fichero buscado. También es utilizado para determinar el rango de ficheros que debe gestionar cada nodo. Es importante resaltar que el sistema de búsqueda únicamente se encarga de localizar información acerca de los ficheros pero no es el encargado de la gestión del catálogo ni el almacenamiento físico de los ficheros.

- **Módulo de Heurística/Replicación**: este componente se encarga de seleccionar la identidad del nodo encargado de gestionar una determinada réplica de fichero. Este módulo hace uso del componente de estadística que le proporciona los datos históricos de acceso al fichero.

- **Módulo de Estadísticas**: recoge información histórica sobre el tipo de acceso que se realiza sobre los ficheros y posteriormente se utiliza esta información para seleccionar las réplicas.

- **Módulo de Persistencia**: cuando un nodo pasa a un estado de inactividad toda la información es almacenada de modo persistente. En un posterior reinicio del nodo toda la información de catalogación, control, estadística, etc. se integra nuevamente al sistema tras realizar un proceso de revisión y reconstrucción con la información ya existente en el sistema.

- **Módulo de Backup**: este módulo se encarga de resguardar toda la información de catalogación que gestiona un nodo vecino. Cada nodo del sistema se encarga de realizar copias de seguridad del nodo vecino con identificador inmediatamente superior y restablecer esta información cuando el nodo tiene un fallo de funcionamiento.

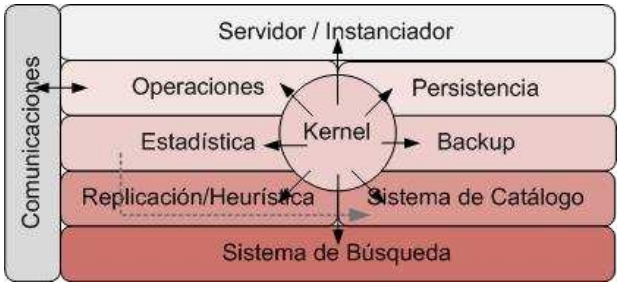

Fig. 10 Arquitectura de capas DFS.

A continuación se hace una breve exposición de los tres módulos principales del sistema DFS: sistema de replicación, sistema de catálogo y sistema de búsqueda. En la siguiente sección se describirá a nivel de diseño cada uno de estos componentes.

4.2.3.1 Sistema de Replicación.

El sistema de replicación de ficheros DFS cumple una doble finalidad, por una parte es un mecanismo esencial para asegurar la tolerancia a los fallos que se producen en los recursos de almacenamiento; también permite reducir el tiempo que invierten las aplicaciones para acceder a los ficheros almacenados.

El sistema de replicación DFS se coordina con dos módulos adicionales con el objetivo de seleccionar los nodos encargados de almacenar las copias de un fichero. La aplicación cliente es la encargada de especificar el número de copias del fichero que deben mantenerse, sin embargo es el propio sistema quien selecciona los nodos donde se envían las copias.

La política de replicación consiste en seleccionar los nodos que tienen mayor probabilidad de ser accedidos posteriormente por las aplicaciones para recuperar ficheros. La arquitectura propone un diseño modular que permite configurar al sistema de modo que sea posible utilizar otra heurística de replicación diferente dependiendo de las necesidades de las aplicaciones.

El módulo de heurística y estadística son utilizados conjuntamente por el sistema DFS para seleccionar los nodos más adecuados para realizar la replicación. El módulo de estadística almacena información histórica acerca de los accesos producidos sobre cada fichero. Esta información se utiliza para determinar qué nodo ha realizado mayores operaciones de lectura sobre el fichero y por tanto con más probabilidades de acceder al fichero en el futuro. En el caso de que no existiera información estadística sobre un fichero, se utiliza la información de accesos por nodo conocido para determinar el nodo de replicación.

Tal y como se verá en el apartado siguiente, el sistema de replicación se asemeja al mecanismo de caches de un sistema multiprocesador donde cada procesador accede a un bloque de información a través de su cache asociada. Del mismo modo DFS permite a las aplicaciones clientes acceder a los ficheros directamente desde los nodos que poseen réplicas válidas del fichero, ya que cada nodo DFS actúa como una cache de ficheros recientemente accedidos y por tanto no se requiere ninguna comunicación adicional a la propia transferencia del fichero.

El almacenamiento de varias réplicas de un mismo fichero en distintas ubicaciones del sistema, requiere que las copias se mantengan actualizadas en tiempo real. La mayoría de los sistemas actuales de almacenamiento distribuido optan por actualizar sus réplicas gradualmente en el tiempo o establecer un único punto de acceso al fichero a pesar de disponer de varias copias. Por el contrario, DFS garantiza que las aplicaciones pueden acceder a cualquier copia del fichero desde cualquier punto del sistema, además las copias se actualizan en tiempo real tras una operación de escritura.

Mantener más de una copia de un fichero en el sistema de almacenamiento puede dar lugar a situaciones de inconsistencia cuando la actualización de todas las copias no se realiza en tiempo real. Para evitar esto, DFS utiliza un modelo de coherencia de ficheros basado en técnicas de coherencia de caches, que permite mantener la integridad de las copias en todo momento; además se aplica un algoritmo de bloqueo sobre las operaciones de escritura que evita solapamientos en la escritura de varias copias de un mismo fichero. A continuación se explica con más detalle cada uno de estos modelos.

Modelo de integridad de ficheros basado en coherencia de caches:

Un sistema distribuido puede asemejarse a un sistema multiprocesador tal y como muestra la Fig. 11. Los procesadores corresponderían a los nodos DFS del sistema, las memorias cache serían los dispositivos de almacenamiento locales a cada nodo, la red puede entenderse como el Grid, los bloques de datos de la memoria serían los ficheros y la memoria principal puede asimilarse al sistema de nombres global.

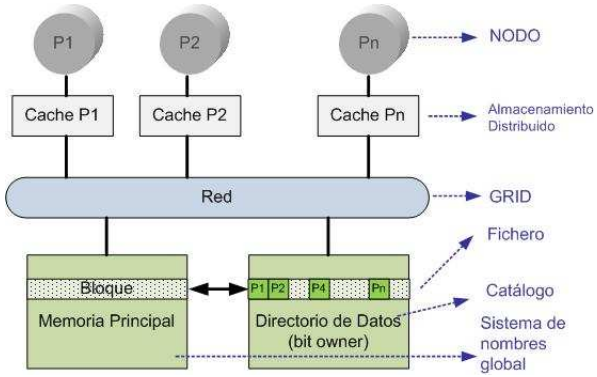

Fig. 11 Similitud entre un sistema multiprocesador y un sistema distribuido de ficheros.

Al igual que en un sistema distribuido de ficheros, el problema de la incoherencia de cache surge cuando los bloques de datos almacenados en cada una de las caches de los procesadores pueden verse desactualizados por escrituras simultáneas de un mismo bloque en más de un procesador. Para abordar este problema existen múltiples soluciones definidas en la literatura, DFS utiliza un modelo de coherencia de ficheros basada en directorios [20].

Básicamente el esquema de coherencia de directorio funciona invalidando las copias de un fichero cuando se realiza una operación de escritura sobre él. Una operación de escritura invalidará todas las copias de un fichero que están distribuidas en el sistema. Además para evitar realizar operaciones de invalidación innecesarias, el protocolo de coherencia introduce un estado llamado "*Dirty*" que permite realizar escrituras repetidas de un mismo fichero invalidando sólo una vez las copias tras la primera escritura del fichero ya que un nodo declarado "*Dirty*" posee la única copia válida y actualizada en el sistema de ficheros.

Cuando una aplicación intenta recuperar desde un nodo un fichero inválido, DFS encuentra la copia correcta del fichero y lo envía al nodo que inicia la petición. Del mismo modo, las operaciones de lectura desde nodos dónde no existe una réplica del fichero implica la creación de una nueva réplica en ese nodo, de este modo las posteriores lecturas del fichero sobre ese nodo son inmediatas.

Escritura simultánea de ficheros:

El sistema permite acceder a los ficheros desde cualquier nodo DFS y esto implica que puede producirse inconsistencias en el sistema de ficheros al tratar de escribir simultáneamente desde dos nodos diferentes al mismo fichero. Para evitar esto, DFS utiliza un mecanismo de bloqueo de ficheros que consiste en no permitir escribir una copia de un fichero mientras haya una operación de escritura en curso. Las operaciones de lectura de un fichero siempre están permitidas excepto el caso en que una operación de escritura terminara correctamente, en tal caso cualquier operación de lectura que estuviera en marcha se abortaría.

La Fig. 12 muestra los posibles casos de uso en una operación de escritura de un fichero.

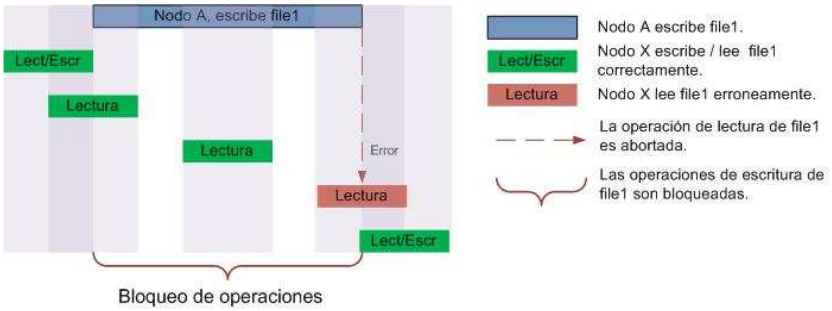

Fig. 12 Acceso concurrente a un fichero en una operación de escritura.

Tal y como se muestra, mientras se produce una operación de escritura sobre el nodo A, las operaciones de escritura son bloqueadas y las operaciones de lectura siempre son permitidas excepto en el caso que la operación de escritura terminase correctamente y hubiese alguna operación de lectura pendiente de terminar.

Mediante este modelo se priorizan las operaciones de lectura sobre las de escritura ya que supone que un fichero es válido sólo cuando finaliza correctamente su escritura. No obstante, existen otros modelos de bloqueo de acceso concurrente a ficheros en los que operaciones de escritura bloquean tanto escrituras como lecturas.

4.2.3.2 Sistemas de Búsqueda y Catálogo.

La arquitectura propuesta está estructurada por medio de un diseño multicapa que permite diferenciar e independizar los sistemas de búsqueda, catálogo y almacenamiento. Cada uno de estos tres sistemas ofrece su propia interfaz por la que se comunican los diferentes componentes de cada nivel.

Para entender mejor la funcionalidad que prestan cada uno de los anteriores sistemas, y poniendo como ejemplo la recuperación de un fichero en el sistema llamado "fichero1", es conveniente plantearse y responder a las siguientes preguntas.

¿Qué nodo posee "información" acerca del "fichero1"? ‣ El sistema de búsqueda permite localizar el nodo que puede contener la información de tipo metadato asociada al fichero de ejemplo. A este nodo se le denominará "propietario del fichero1".

Una vez conocida la identidad del nodo que gestiona la información de control de un fichero, se debe plantear la siguiente pregunta sobre el nodo propietario.

¿Qué nodos almacenan una réplica del fichero buscado? ‣ El sistema de catalogación almacena información acerca de las propiedades de un fichero incluyendo localización de nodos de réplica, flag indicando la existencia de un nodo dirty, etc.

Una vez obtenida la dirección del nodo encargado de almacenar una réplica del fichero, el cliente puede recuperar directamente la copia del fichero. El sistema de almacenamiento está compuesto por nodos encargados de almacenar y recuperar ficheros por medio de diferentes protocolos de transmisión (socket o ftp)..

A continuación se explican en profundidad los sistemas de búsqueda y catalogación.

4.2.3.2.1 Sistema de Búsqueda.

El sistema de búsqueda DFS permite localizar un fichero dentro del sistema usando únicamente de su nombre virtual asignado. En los sistemas de almacenamiento centralizados la búsqueda de los ficheros resulta trivial al existir un único servidor que gestiona la localización de los ficheros.

En los sistemas distribuidos, la búsqueda de un fichero puede resultar más compleja si se tiene en cuenta que el catálogo de ficheros es mantenido por todos los componentes del sistema de almacenamiento. A este respecto, un punto esencial para el buen funcionamiento del sistema es asegurar que las búsquedas de los ficheros obtienen un resultado determinista. Esto significa que debe existir un mecanismo tolerante a fallos que permita interrogar al sistema acerca de la existencia o no de un determinado fichero. Un sistema de búsqueda no determinista podría originar que el sistema de almacenamiento se volviera inconsistente por la existencia de ficheros duplicados.

DFS utiliza un mecanismo de tablas hash distribuidas consistente en asignar a cada nodo del sistema de búsqueda, un rango de ficheros que debe gestionar de forma determinista. En un sistema de tablas distribuidas cada nodo perteneciente al sistema está encargado de almacenar el valor de un conjunto de claves delimitadas entre dos índices conocidos por cada nodo.

Las arquitecturas P2P estructuradas requieren implementar sistemas de búsqueda deterministas ya que cada componente en el sistema tiene asignado de modo predeterminado la gestión de una parte del rango de claves de búsqueda.

Existen diversos modelos e implementaciones de arquitecturas P2P de búsquedas en sistemas distribuidos como por ejemplo, Pastry [14], Chord [17], Tapestry [18], etc. Todos estos sistemas se basan el modelo de tablas distribuidas DHT (*Distributed Hash Table*) que consiste en asignar un identificador único a cada nodo del sistema y delegar la gestión de un rango de pares clave/valor a cada uno de ellos, dependiendo del identificador que tengan asignado.

DFS implementa una adaptación simplificada del modelo Pastry para desarrollar su sistema de búsqueda. Pastry organiza los nodos del sistema con una topología anillo de modo que cada componente posee un vecino superior e inferior directo con el que conectar. Esta disposición de nodos es idónea para realizar búsquedas deterministas, ya que cada nodo conoce que rango de ficheros ha de gestionar y a la vez proporciona estabilidad al sistema en el caso de fallos de red o inactividad de alguno de los nodos.

La especificación completa de Pastry puede ser encontrada en referencias como [14], esta tesina de Máster se centrará en describir la adaptación realizada al protocolo Pastry y su implementación sobre el sistema DFS.

Básicamente el sistema de búsqueda puede ser visto como un gran anillo dónde cada nodo tiene un identificador único. El identificador de cada nodo es generado por medio de una función hash en base a la dirección IP, MAC o nombre de máquina DNS.

Del mismo modo, cualquier fichero en el sistema tiene asociado un identificador único que se calcula por medio de su nombre virtual y la misma función hash empleada para calcular los identificadores de nodo.

Los nodos son ordenados en el anillo de menor a mayor dependiendo de sus identificadores, de esta forma cada nodo dentro del anillo conoce a sus vecinos más próximos (con identificador superior e inferior). Además de conocer a su vecino inferior y superior cada nodo también posee referencia a los *k* vecinos superiores más próximos. Esta información es necesaria para permitir reconectar el anillo en caso de fallo en un nodo vecino y también para hacer posible el enrutamiento de la búsqueda de un fichero en el anillo.

Del mismo modo que los nodos, cada fichero posee un identificador que es generado utilizando la misma función de hash que emplean los nodos DFS. De este modo, se asocia cada fichero con el nodo vecino inferior que posea el identificador más cercano al del fichero. El nodo asociado con cada fichero es el nodo denominado "propietario" y tal como se ha comentado se encargará de gestionar la información de tipo metadato asociada al fichero.

Cuando se pretende encontrar el nodo propietario de un fichero, el nodo que inicia la búsqueda, en la Fig. 13 el nodo A, genera el identificador de fichero (utilizando la función hash y el nombre virtual) y comprueba que no se trata del nodo "propietario". En caso de que no sea nodo "propietario" del fichero (File1) la búsqueda continua por medio del vecino superior conocido más lejano del nodo A, y así continua el proceso hasta encontrar con el nodo cuyo identificador se corresponda con el del nodo más próximo por debajo al identificador de File1. Si este nodo (nodo B) contiene una entrada en su catálogo al fichero buscado, la búsqueda finaliza devolviendo la referencia del nodo B al nodo iniciador (nodo A), en otro caso el fichero no existe en el sistema.

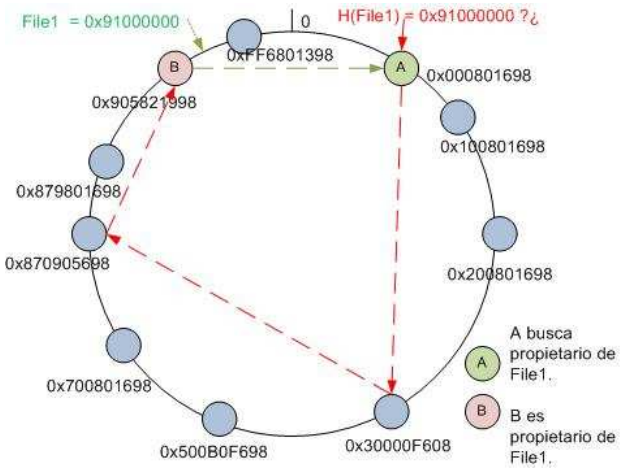

Fig. 13 Sistema de búsqueda utilizando la topología en anillo. Cada nodo posee un identificador único calculado a partir de su dirección IP y una función hash.

4.2.3.2.2 Sistema de Catálogo.

Cada nodo DFS gestiona la información de tipo metadato sobre una parte del catálogo global del sistema, de este modo cada fichero se asocia a un nodo llamado "propietario" que almacena únicamente la información de tipo metadato del fichero.

Los metadatos de un fichero contienen información referente a la localización física de las réplicas de un fichero, la información de control de copias invalidadas, los accesos producidos sobre un fichero y toda aquella información necesaria para mantener consistente el fichero dentro del sistema de almacenamiento. En apartados posteriores se mostrará cómo se estructura internamente esta información.

Para acceder a la información metadato de un fichero con el fin de localizar sus réplicas, es necesario localizar en primer lugar el nodo "propietario" y para ello se requiere de un sistema de búsqueda de ficheros determinista.

Esto implica que cuándo un nodo requiere localizar al nodo "propietario" de un fichero, el sistema de búsqueda devolverá siempre una referencia al nodo "propietario" si el fichero existe. Es necesario adoptar este modelo determinista de búsqueda dentro de un sistema distribuido como DFS ya que cualquier otro sistema no determinista podría mantener copias incoherentes de un mismo fichero produciendo la inestabilidad del sistema completo.

Una vez que el sistema de búsqueda ha resuelto la referencia al nodo "propietario" de un fichero, es posible obtener directamente la información metadato del fichero y por tanto la localización de sus réplicas.

Es importante remarcar que el sistema de búsqueda y catálogo son independientes del sistema de almacenamiento ya que las réplicas de un fichero son almacenadas en ubicaciones separadas de los nodos que mantienen la información metadato, es decir, de los nodos "propietarios". La Fig. 14 muestra la interacción entre los sistemas de búsqueda, catalogación y almacenamiento para una operación de lectura.

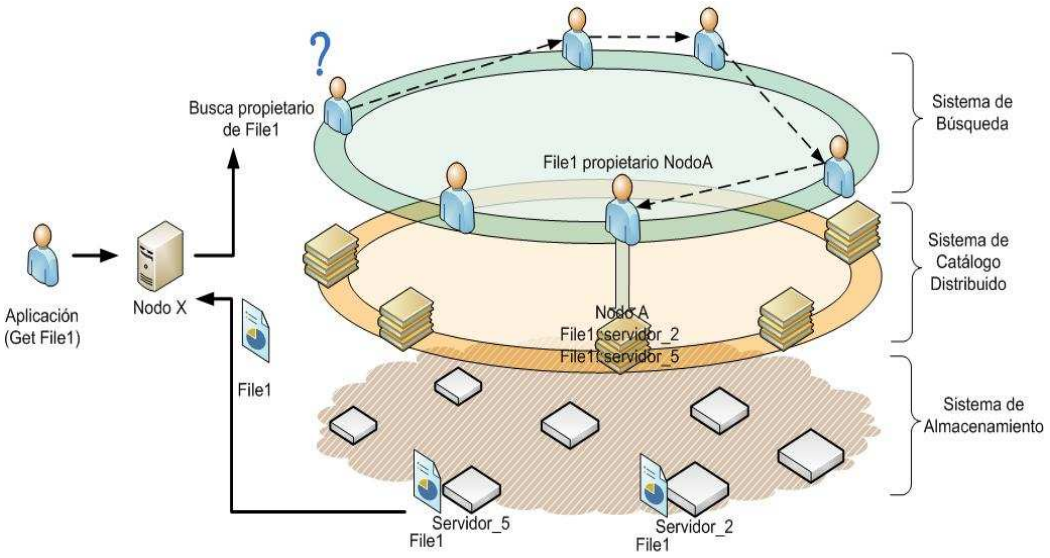

Fig. 14 Esquema General del Sistema de Catálogo y Búsqueda.

Capítulo 5 - Diseño de Componentes.

En el capítulo anterior se ofreció una visión general de la arquitectura de componentes del sistema DFS. En este capítulo se presenta el diseño de los componentes además de otros detalles de implementación.

5.1 Diagrama de Componentes.

DFS posee un diseño modular que permite agrupar los componentes de la arquitectura tal y como describe el diagrama de componentes de la Fig. 15. Cada nodo en el sistema actúa como una entidad independiente que recibe y emite peticiones de transferencia.

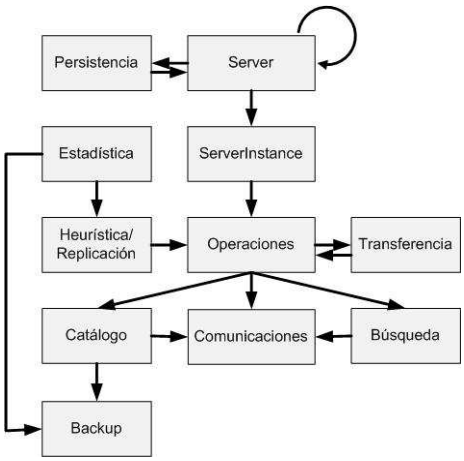

Fig. 15 Diagrama de componentes del sistema DFS.

Del esquema anterior, se puede deducir que existe un componente servidor que permanece activo en el sistema a la espera de recibir nuevas peticiones de transferencias de ficheros. Por cada operación que se recibe, el servidor crea una instancia del proceso principal encargada de desplegar la funcionalidad de la operación solicitada.

Los módulos de operación pueden necesitar interaccionar con otros componentes del sistema DFS dependiendo de la funcionalidad que implementen. Así para una operación de escritura (PUT) intervienen los módulos de estadística y replicación para seleccionar las replicas y también el sistema de catálogo y búsqueda; para una operación de lectura (GET) y borrado (DEL) los módulos que intervienen son los sistemas catálogo y búsqueda.

El módulo de comunicación es utilizado por la mayoría de módulos para enviar mensajes de control entre otros nodos DFS del sistema; el módulo de transferencia se encarga de realizar las acciones necesarias para transmitir los ficheros por medio del sistema de almacenamiento; por último el módulo de backup proporciona funcionalidad para almacenar la información de catálogo y de estadística del nodo vecino al actual.

5.2 Servidor.

El componente servidor es el primer proceso que se pone en ejecución al arrancar un nodo DFS y su función es inicializar las variables de entorno del sistema además de coordinar y monitorizar la actividad del nodo. A continuación se detallan las operaciones que suceden en orden de ejecución.

Creación de la estructura de repositorios:

El primer paso es la creación de la estructura de directorios que utilizará DFS dentro del sistema local de almacenamiento. En el inicio del arranque el proceso servidor comprobará la existencia de los directorios (creando uno nuevo en caso de no existir) que se listan a continuación.

tmp: se almacenan de modo temporal los ficheros que las aplicaciones han de transferir al sistema independientemente de que finalmente se utilice un protocolo de transferencia específico.

persistence: almacena en tres ficheros binarios toda información que el nodo debe mantener para poder conservar el estado en el siguiente reinicio. El fichero "data_a" almacena la información del catálogo, "data_b" guarda la información de backup del nodo actual y el fichero "data_c" almacena la información del sistema de búsqueda.

repos: almacenan los ficheros en el sistema local cuando el mecanismo de transferencia utilizado es por medio de *sockets*.

Aparte de estos directorios se inicializa el fichero "DFS.log" que mantendrá la información de error y depuración durante la actividad del nodo.

Inicialización de las variables configuración:

El siguiente paso en la inicialización del nodo es la lectura de los parámetros de configuración del sistema desde el fichero "DFS.cfg". A continuación se listan los parámetros por defecto de este fichero y su función dentro del sistema.

Parámetro	Descripción	Valor Defecto
port	Número de puerto en el que escucha el servidor.	10000
sendaliveinterval	Intervalo de tiempo que debe esperar un nodo antes de enviar un mensaje para indicar a su nodo vecino que sigue activo.	20 seg..
checkaliveinterval	Intervalo de tiempo que debe esperar un nodo antes de comprobar si su nodo vecino sigue activo.	40 seg.
sendbackupinterval	Intervalo de tiempo que debe esperar un nodo antes de enviar el bloque de ficheros que ha de resguardar su nodo vecino.	30 seg.
startinginterval	En el caso de tratarse del primer nodo que se inicia en el sistema, indica el tiempo durante el cual el resto de nodos que estaban activos antes de parar el sistema pueden unirse de nuevo, restableciéndose así la información metadato que existía antes de la parada del sistema.	60 seg.
transfertype	Especifica el protocolo de transferencia que el nodo utiliza para transferir los ficheros.	Socket
friendnode	Cuando un nodo se une al sistema debe conocer al menos un nodo activo en el sistema, por medio de este parámetro es posible indicar la dirección IP del nodo de contacto. Además, el sistema mantiene una lista dinámica de los nodos que integran el sistema por lo que en caso de no especificar un nodo "friend" el nodo intenta contactar con alguno de los registrados en el fichero "nodes.txt"	
nodespath	Ruta donde se almacena el fichero "nodes.txt".	
tmppath	Ruta al directorio temporal.	
repositorypath	Ruta al directorio repositorio.	
persistencepath	Ruta al directorio persistente.	

Inicialización de los módulos:

El siguiente paso antes de poner al proceso servidor en modo escucha es la inicialización de cada uno de los módulos que prestarán funcionalidad al sistema DFS. En concreto la instanciación de componentes se realiza en el siguiente orden:

Estadística → Replicación → Búsqueda → Catálogo → Transferencia

Activación del nodo:

Tras el proceso de inicialización anterior, el nodo se pone en estado de espera escuchando peticiones de operación en el puerto especificado. Durante el periodo de actividad normal dentro del sistema, los nodos pueden cambiar de estado (Fig. 16) afectando al tipo de operaciones que es posible invocar.

Starting: el primer nodo que se inicia en el sistema se pone en estado "starting" durante un periodo de tiempo esperando que nuevos nodos se unan al sistema. Durante este estado no están permitidas las operaciones externas de transferencia de ficheros y únicamente se permiten operaciones de control. Finalizado el periodo de tiempo, los nodos en estado "starting" cambian al siguiente estado "building".

Building: cada uno de los nodos en este estado realiza dos tareas, por una parte recuperan del módulo de persistencia toda la información de catálogo que poseen desde la última vez que el nodo estuvo activo, además se inicia un proceso de revisión del catálogo recuperado para descartar aquellos ficheros que ya no son gestionados por el nodo (por la unión de un nuevo nodo vecino en el sistema) y deben pasar a ser administrados por otro nodo. Durante este estado también se detectan nodos que no están activos ya en el sistema y de los que se guarda un backup de su catálogo de ficheros. Las operaciones permitidas en el estado de "building" son las mismas que el estado de inicialización y por tanto no es posible transferir ficheros ni tampoco la unión de nuevos nodos en el sistema.

Online: una vez se ha reconstruido y actualizado los catálogos de ficheros de todos los nodos que se iniciaron en la primera fase de inicialización, cada nodo pasa al estado "online" indicando que el nodo está activo y totalmente funcional a la espera de recibir operaciones de transferencia.

Recovery: un nodo en estado "online" puede pasar al estado de recuperación cuando detecta que su nodo vecino está inactivo y por tanto debe hacerse cargo del catálogo de ficheros que administraba su vecino. Una vez se ha integrado toda la información de backup dentro del catálogo de ficheros, el nodo vuelve a pasar al estado online permitiendo realizar nuevas operaciones de transferencia.

Shutdown: un nodo en el sistema puede pasar a estado de inactividad voluntariamente sin que se produzca un error. Durante este estado el nodo hace persistente la información de catálogo y se deshabilitan cualquier tipo de operación de transferencia con los ficheros.

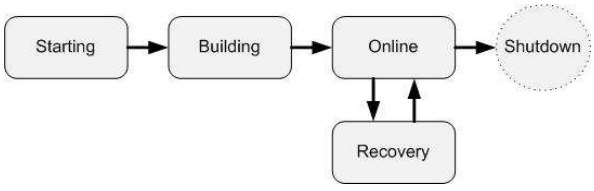

Fig. 16 Diagrama de estados en un nodo DFS.

5.3 Instanciador de Operaciones.

En apartados anteriores se describió el conjunto de operaciones que los nodos DFS pueden ejecutar. Cuando una solicitud de operación llega al servidor se procesa creando un nuevo hilo de ejecución que se asocia a un tipo concreto de operación junto con los parámetros de entrada requeridos. Debido a que existen múltiples operaciones y no todas pueden ser resueltas en función del estado en que se encuentra el nodo, existe un componente encargado de distinguir el tipo de operación que ha de ejecutarse y si es posible realizar la ejecución para el estado en que se encuentra el nodo. Tras recibir los parámetros de

operación desde el servidor, el componente instanciador creará componente de operación adecuado y encargado de interaccionar con el cliente.

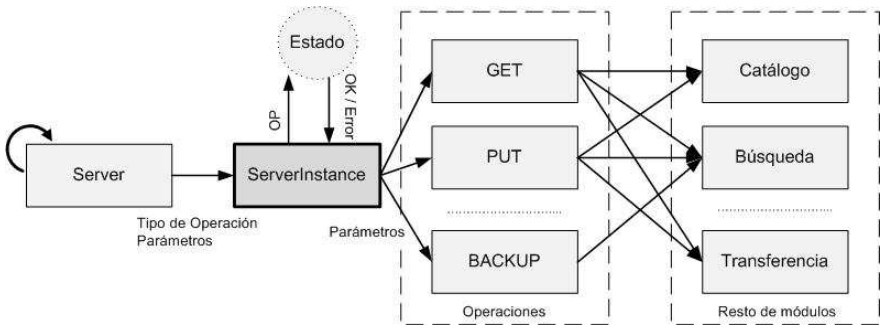

Fig. 17 El módulo de instancia recibe los parámetros de operación y transfiere el control dependiendo del tipo de operación.

El siguiente listado muestra las operaciones y sus posibles estados de ejecución dentro del nodo.

Operaciones de transferencia cliente - nodo:		
Operación	Parámetros	Estados hábiles
put	String localname String virtualName int nReplicas	ONLINE
get	String virtualName	ONLINE
del	String virtualName	ONLINE

Operaciones de transferencia nodo - nodo:		
Operación	Parámetros	Estados hábiles
putDFS	String localname String virtualName int nReplicas	ONLINE
getDFS	String virtualName	ONLINE
delDFS	String virtualName	ONLINE
Operaciones de búsqueda:		
getOwner	long key	ONLINE, STARTING, BUILDING, RECOVERY
joinRing	long idNewNode String ipNewNode	ONLINE, STARTING
neighborOperation	int type String virtualName Boolean recursive	ONLINE, RECOVERY, STARTING
statusRing	String ipFirstNode	ONLINE, RECOVERY
manageRing	int type Boolean recursive	ONLINE, STARTING, BUILDING, RECOVERY, SHUTDOWN
notifyNode	String ipFailedNode	ONLINE, RECOVERY, BUILDING
keepAlive		ONLINE, STARTING, BUILDING, RECOVERY, SHUTDOWN
Operaciones de catalogación:		
backupCatalog	List blockFiles Boolean delOldCataloge	ONLINE, RECOVERY, BUILDING
rebuild	Array[dataFile] blockFiles	ONLINE, STARTING
Operaciones de Transferencia:		
transfer	Boolean type	ONLINE

	String virtualName	

En el siguiente apartado se explica el modelo de comunicación entre los nodos DFS y las aplicaciones; también se describirá el mecanismo de empaquetamiento de mensajes que permite seleccionar que operaciones han de realizarse y los parámetros de entrada necesarios.

5.4 Módulo de Comunicaciones.

El componente comunicaciones permite a las aplicaciones y a los nodos DFS enviar información de control para realizar una determinada operación sobre un nodo. Se debe diferenciar el componente comunicación, que sólo es utilizado para solicitar la ejecución de operaciones remotas, del sistema de transferencia que ofrece un servicio completo de transferencia de ficheros.

Las comunicaciones entre aplicaciones y nodos DFS se realizan por medio del envío de mensajes de control remotos. Esto quiere decir, que cada vez que se requiere realizar una operación sobre un nodo la aplicación cliente deberá construir un objeto mensaje del tipo de la operación que se solicita y además se agregará los argumentos de entrada específicos para esa operación. De este modo existen tantos tipos de mensajes de control como operaciones existen. Además existe un mensaje especial de respuesta que será utilizado por el nodo receptor para informar del resultado de la operación o devolver la información que el nodo emisor solicita. El siguiente listado muestra los tipos de mensajes de operación que se utilizan para cada operación.

- messagePut
- messageGet
- messageDel
- messageGetDFS
- messagePutDFS
- messageDelDFS
- messageDiscoveryGetOwner
- messageDiscoveryJoinRing
- messageDiscoveryNeighborOperation
- messageDiscoveryStatusRing
- messageDiscoveryManageRing
- messageDiscoveryNotifyNode
- messageDiscoveryKeepAlive
- messageCatalogBackup
- messageCatalogShowCatalog
- messageCatalogRebuild
- messageTransfer
- messageReturn

La Fig. 18 muestra la estructura para un mensaje asociado a una operación “Show” que permite obtener el listado de los ficheros que coinciden con determinado patrón. La Fig. 19 hace referencia a la estructura de mensaje para una operación de retorno. Esta estructura la utilizan todas las operaciones para indicar entre otras cosas el código de terminación de la operación y devolver los resultados si existieran en alguno de los tipos de argumentos disponibles.

```
public class MessageCatalogShowCatalog implements Serializable {
private static final long serialVersionUID = 1L;
      // Tipo de operación a realizar (0 mostrar catálogo global, 1
      mostrar sólo catálogo del nodo completo)
      public int operationType;
      // Primer nodo en consultar.
      public boolean firstRequest;
      // Consultar catálogo de ficheros propietarios o no.
      public boolean onlyOwner;
      // Patron de fichero a recuperar.
```

```
        public String pattern;
        // Nivel de profundidad.
        public int deepthLevel;
        // Información adicional sobre el mensaje.
        public String extraData;
}
```

Fig. 18 Ejemplo de mensaje de control para la operación Show.

```
public class MessageReturn implements Serializable
{
        private static final long serialVersionUID = 1L;
        // Código de terminación.
        public int code;
        // Cadena un código de error.
        public String codeDescription;
        // Cadena un sólo resultado.
        public String result;
        // Cadena vector de resultados.
        public Vector<String> vResults;
        // Cadena vector de resultados auxilar.
        public Vector<DataCatalogFile> vResultsFiles1;
        // Cadena vector de resultados auxilar.
        public Vector<DataCatalogFile> vResultsFiles2;
        // Objeto genérico asociado al mensaje de respuesta.
        public objectPropertiesFile resultObject;
        // Tabla de objetos genéricos asociados a las propiedades del
        fichero.
        public HashMap<String, objectPropertiesFile> hResultsObject;
        // Resultado entero 1
        public int iResult1;
        // Resultado entero 2
        public int iResult2;
}
```

Fig. 19 Ejemplo de mensaje de control para la operación de respuesta.

Los mensajes de control son construidos por medio de objetos que se serializan en el momento del envío y se transfieren utilizando los sockets del sistema operativo. Existe un módulo que permite automatizar este proceso y puede ser utilizado por los nodos DFS cada vez que requieren ejecutar una operación sobre otro nodo.

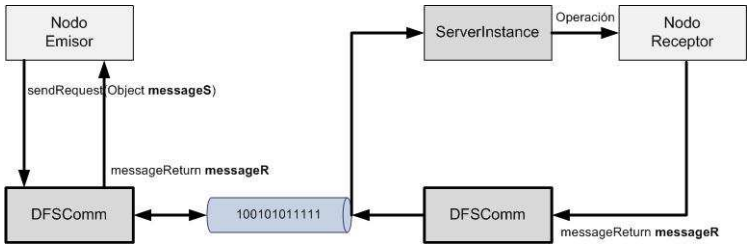

Fig. 20 Serialización de mensajes de control de operaciones.

5.5 Sistema de Búsqueda.

En el capítulo anterior se explicó brevemente la arquitectura del sistema de búsqueda DFS, en este apartado se profundizará en el diseño e implementación de un sistema que ofrezca los siguientes servicios.

Estabilidad del sistema: el sistema de búsqueda debe garantizar que el sistema de almacenamiento permanezca activo tras un fallo en alguno de sus nodos o la inserción de nuevos nodos. Este servicio se ha de prestar de un modo independiente a cualquier otro sistema DFS.

Búsquedas deterministas: el sistema permitirá resolver de modo determinista qué nodo del sistema es el propietario de un determinado fichero. El sistema de catálogo utilizará esta información para determinar la existencia o no del fichero.

El sistema de búsqueda y el sistema de catálogo están fuertemente relacionados ya que para encontrar un fichero en DFS es necesario conocer qué nodo gestiona la información de tipo metadato asociada al fichero. Sin embargo la implementación del sistema de búsqueda debe ser independiente de cualquier otro sistema dentro de DFS. Por ello el sistema se diseña como un módulo que ofrece los servicios de búsqueda de claves genéricas[7]. En este sentido, DFS utiliza una implementación simplificada del modelo de búsquedas distribuidas Pastry basado en una cache de nodos conocidos que permite acelerar las búsquedas y reducir la complejidad de enrutamiento en el sistema. La topología de comunicación de Pastry especifica un anillo de nodos en el que cada nodo se comunica con su vecino superior e inferior, además por medio de una tabla de enrutamiento y cierto algoritmo de búsqueda es posible localizar el nodo que se hace cargo de la gestión de una determinada clave.

5.5.1 Modelo de Datos del Sistema de Búsqueda.

El sistema de búsqueda posee una estructura de datos similar a la presentada en la Fig. 21; cada nodo DFS almacena las referencias IP a su vecino superior directo, su vecino inferior directo y a sus k vecinos superiores más próximos. De este modo en todo momento se garantiza la conectividad de un nodo con sus k vecinos superiores más próximos lo que permite recuperar fácilmente la estabilidad del sistema tras el fallo de uno de estos k nodos.

Además de mantener una referencia constante a los nodos vecinos más próximos, cada nodo DFS mantiene una tabla con las referencias IP de los nodos que va descubriendo en el sistema tras realizar una operación de búsqueda. Esta tabla funciona a modo de cache dinámica y permite enrutar las solicitudes de búsqueda que no pueden ser resueltas por el propio nodo; además DFS implementa esta cache a modo de árbol binario de búsqueda lo que permite realizar búsquedas de claves rápidamente.

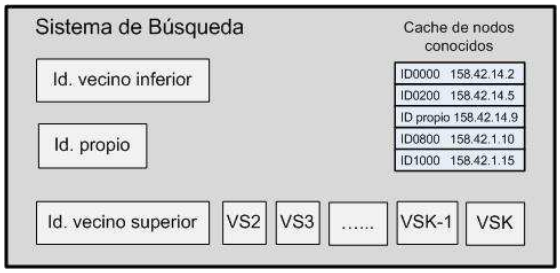

Fig. 21 Estructura de datos del sistema de búsqueda DFS.

5.5.2 Inserción y Eliminación de Nodos en el Sistema.

Inserción de un nuevo nodo en el sistema de almacenamiento DFS:

La inserción de un nuevo nodo en el sistema sigue una serie de pasos que se detallan a continuación.

1ª Fase: cuando un nuevo nodo quiere integrarse dentro del sistema DFS genera su identificador a partir de su dirección IP. Posteriormente se comunica con un nodo conocido y activo en el sistema para iniciar la búsqueda del nodo "padre" que será su futuro nodo inferior cuando concluya la fase de inserción.

[7] La clave de un fichero se determina en DFS aplicando una función hash MD5 al nombre del fichero.

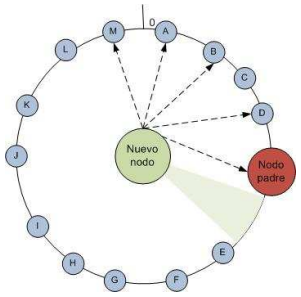

Fig. 22 Búsqueda del nodo "padre" de un nuevo nodo a insertar en el sistema.

2ª Fase: el nuevo nodo contacta con su nodo "padre" que le devuelve la referencia a su vecino superior más próximo y la lista de sus nuevos k vecinos superiores directos (**¡Error! No se encuentra el origen de la referencia.**6).

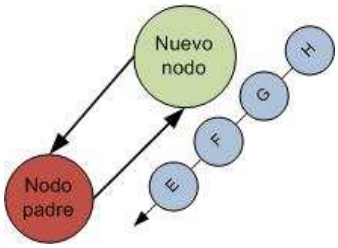

Fig. 23 El nodo "padre" devuelve las referencias a los nodos vecino superior e inferior.

3ª Fase: el nuevo nodo contacta con su vecino superior inmediato y crea un primer enlace con él Fig. 24.

4ª Fase: el nuevo nodo confirma la operación de inserción con su nodo "padre" que deshace el enlace que tenía con su anterior vecino superior actualizándolo al del nuevo nodo Fig. 24.

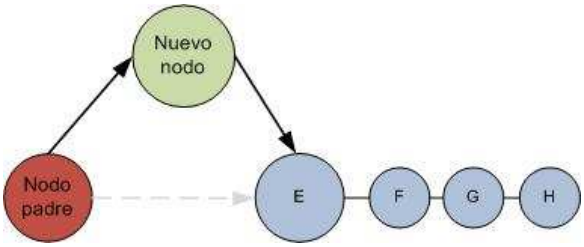

Fig. 24 El nuevo nodo se inserta en el sistema y pasa a estar activo.

Desconexión de un nuevo nodo del sistema de almacenamiento DFS:

Tal y como se ha comentado anteriormente, el anillo de búsqueda debe permanecer en todo momento consistente y por tanto se debe diseñar un protocolo de reconexión ante un fallo en algún enlace entre sus nodos.

DFS implementa un mecanismo de monitorización de la actividad del nodo vecino superior que funciona con el envío de un mensaje de control denominado "*keepalive*" que es enviado cada cierto tiempo al nodo vecino inferior. A la vez, cada nodo monitoriza cada cierto tiempo los mensajes de "*keepalive*" enviados por su nodo vecino superior, así que cuando un nodo detecta que su vecino deja de enviar estos mensajes considera que se ha producido un fallo en el enlace con su vecino superior e inicia el protocolo de reconexión del anillo.

El protocolo de reconexión del anillo de búsqueda implementado por DFS tiene 2 fases que se detallan a continuación:

1ª Fase: uno de los nodos detecta un fallo de conexión con su vecino superior por lo que intenta comunicar por orden ascendente con sus k vecinos superiores más próximos. Si se produce la comunicación con alguno de los k vecinos, el nodo actualiza la referencia de su vecino superior.

En la Fig. 25 el nodo A inicia el proceso intentando la conexión con sus vecinos superiores, el nodo C se convierte en el nuevo vecino superior de A por lo que el sistema de búsqueda se restablece nuevamente.

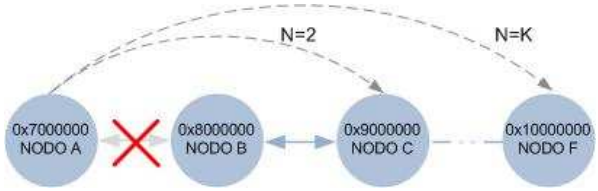

Fig. 25 Paso 1, reconexión con alguno de los k vecinos superiores.

2ª Fase: si ninguno de los k vecinos superiores de un nodo responde a una petición de reconexión entonces el nodo que detecta el fallo inicia un protocolo de reconfiguración que implica averiguar el último nodo accesible. En la Fig. 26 el nodo A detecta que no hay conexión con sus vecinos superiores B, C, D y E, por lo tanto envía un mensaje de reconfiguración en sentido inverso a las agujas del reloj. El mensaje se retransmite de nodo a nodo hasta que el último nodo F no puede comunicar con su vecino E. El nodo F devuelve al A su identificador y de este modo se consigue reconfigurar nuevamente el anillo.

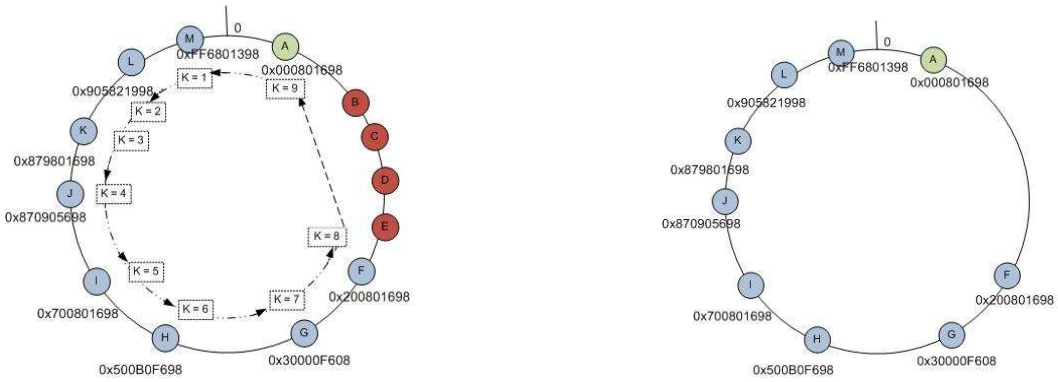

Fig. 26 Fase 2 Reconexión del anillo tras un fallo en múltiples nodos.

El caso más complejo puede darse cuando se produce simultáneamente fallos en distintos enlaces del anillo produciendo así dos subredes de nodos que podrían ocasionar inconsistencia en el sistema de ficheros del sistema.

La Fig. 27 muestra el caso para un fallo simultáneo en dos de los enlaces de comunicación que existen en el anillo (enlace A-B y enlace F-E). En esta situación el protocolo de reconfiguración determinaría que existen dos subanillos de búsqueda activos se inicia un proceso para determinar cuál de los anillos debe permanecer activo.

Para el caso de la Fig. 27 el sistema DFS establece que el anillo que deberá permanecer activo será el que mayor número de nodos posea respecto al anillo inicial antes de la desconexión[8]. Para este caso, el nodo A determina que el anillo 1 está formado por 9 nodos mientras que el nodo E obtiene un número 4 nodos en su anillo 2, entonces aplicando la regla n > (N/2) + 1 el nodo A establece que su anillo debe de quedar activo mientras que el nodo E debe desactivar el anillo 2.

[8] Un nodo determina si un anillo debe quedar activo si se cumple que n > (N / 2) +1, siendo n el número de nodos del subanillo y N el número de nodos totales antes de producirse el fallo.

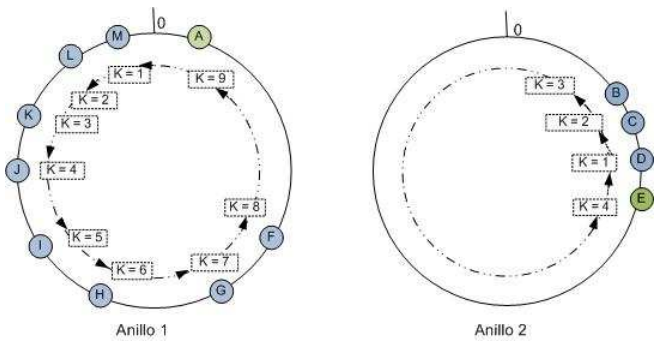

Fig. 27 Formación de 2 anillos tras un fallo múltiple de conexión. El anillo 2 queda desactivado hasta que se produzca la reconexión.

Para todas estas situaciones de reconexión el sistema activa el estado "RECOVERY" en los nodos afectados no permitiendo realizar ninguna operación de transferencia de ficheros hasta que el anillo de búsqueda recupera su conectividad.

A parte de las operaciones propias de recuperación del sistema de búsqueda, la inserción y desconexión de un nodo en el anillo de búsqueda implica una serie de actualizaciones en el sistema de catálogo de los nodos afectados que serán explicadas en el siguiente apartado.

5.6 Sistema de Catálogo.

Tal y como se comentó en la sección anterior, cada fichero en el sistema DFS está asociado a un nodo activo en el sistema. Esto quiere decir que el fichero puede existir en múltiples ubicaciones pero su información metadato sólo se almacena en uno de los nodos DFS y puede ser recuperada desde cualquier punto del sistema de un modo determinista. Es importante aclarar a que se refiere esta información metadato por eso en el siguiente apartado se definirá la estructura exacta de datos de fichero.

A pesar de que cada fichero dentro del sistema de almacenamiento está catalogado por un nodo DFS, desde el punto de vista de las aplicaciones clientes, los ficheros son accedidos del mismo modo que si existieran en un dispositivo de almacenamiento local. Para ello, DFS define un espacio de nombres común a todo el sistema que permite que cualquier fichero pueda ser identificado sin hacer referencia a su ubicación física, además para cada organización que haga uso de DFS es posible utilizar una sintaxis sin restricciones identificando ficheros como por ejemplo: /dir1/dir2/fichero1 ó dir1.dir2.fichero1 ó fichero1.

5.6.1 Modelo de Datos del Catálogo.

Cada nodo DFS almacena información metadato para dos tipos de ficheros:

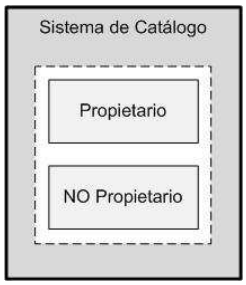

Fig. 28 Modelo de catálogo para un nodo DFS.

Ficheros propietarios: son los ficheros que están bajo el dominio de administración directo del propio nodo y por tanto se posee toda la información necesaria para realizar tareas como localizar ubicaciones

de réplicas, invalidar las réplicas de un fichero en una operación de escritura o bloquear accesos de escritura para todas las réplicas del fichero. Para cada fichero en el sistema existe un único nodo que posee toda la información que a continuación se describe en su tabla de ficheros propietarios.

Estructura de datos para el catálogo de ficheros **PROPIETARIOS**						
Nombre	Bloqueo	Inválido	Dirty	DirtyNode	Replicas	Propiedades
String	Boolean	Boolean	Boolean	String	List	Objeto

Nombre: nombre del fichero en el sistema de nombres global DFS.

Bloqueo: flag que habilita o deshabilita las operaciones de escrituras sobre un fichero. Si existe una operación de escritura en curso entonces el flag aparece habilitado impidiendo que cualquier réplica pueda ser escrita simultáneamente. Finalizada la operación de escritura el flag vuelve a deshabilitarse.

Inválido: si se habilita este campo significa que el fichero no es accesible directamente desde este nodo pero sigue pudiendo ser accedido en el caso que exista una réplica del mismo.

Dirty: habilitado indica que el fichero tiene una única copia válida cuya ubicación se indica en el campo "DirtyNode". Cuando existen ficheros con este flag activado el acceso para escritura es más rápido ya que no es necesario invalidar ninguna réplica.

DirtyNode: sólo es útil para el caso de ficheros con el campo "dirty" habilitado. Indica la ubicación del nodo que contiene la única copia del fichero.

Replicas: contiene un array de ubicaciones a las réplicas del fichero.

Propiedades: es un campo opcional de tipo objeto que contiene propiedades específicas para cada sistema de fichero. Para el sistema DFS este campo no es interpretable y su contenido sólo es legible para las aplicaciones clientes que requieren almacenar propiedades específicas de cada fichero como por ejemplo fecha, tipo de fichero (directorio, texto, binario, etc.), flags de seguridad, etc.

Ficheros No Propietarios: bajo esta estructura se almacena la información lógica de los ficheros que sin estar gestionados directamente por el nodo pueden ser accedidos ya que el nodo gestiona una de sus réplicas. Podría decirse que las réplicas de un fichero son consideradas como ficheros no propietarios.

Estructura de datos para el catálogo de ficheros **NO PROPIETARIOS**						
Nombre		Inválido	Dirty			Propiedades
String		Boolean	Boolean			Objeto

Nombre: nombre del fichero que se identifica de manera única dentro del sistema.

Inválido: si el flag está activado entonces el fichero no puede ser accedido directamente desde este nodo y debe consultarse si existe una réplica válida en el nodo propietario del fichero.

Dirty: si el campo está activo entonces el nodo contiene la única copia del fichero y la aplicación cliente puede realizar sucesivas operaciones de escritura sin que se requiera invalidar ninguna réplica.

Propiedades: es un campo opcional de tipo objeto que contiene propiedades específicas para cada sistema de fichero. Para el sistema DFS este campo no es interpretable y su contenido sólo es legible para las aplicaciones clientes que requieren almacenar propiedades específicas de cada fichero como por ejemplo fecha, tipo de fichero (directorio, texto, binario, etc.), flags de seguridad, etc.

5.6.2 Reconstrucción del Catálogo.

Existen tres situaciones en las que el catálogo de un nodo debe ser revisado y actualizado para asegurar que no contiene información inconsistente de ficheros, es decir que debido a cambios en la

configuración del anillo de búsqueda puede suceder que un nodo continúe gestionando ficheros de los cuales ya no es propietario y viceversa. A continuación se exponen estas tres posibles situaciones y como el sistema de catálogo soluciona la inconsistencia de datos.

Reinicio del anillo: la arquitectura DFS permite mantener accesible al sistema de almacenamiento mientras exista un nodo activo en el sistema de búsqueda y catalogación; Sin embargo, existen unas pocas situaciones (en operaciones de mantenimiento globales), que requieren poner durante un corto periodo de tiempo en estado de inactividad a todos los nodos DFS. La operación de "*shutdown*" requiere coordinar a todos los nodos del sistema para que almacenen de una forma ordenada y persistente la información perteneciente al catálogo de cada nodo.

Una vez han terminado las operaciones que llevaron al sistema al estado "*shutdown*", es posible iniciar un proceso de arranque de los nodos del anillo que permita habilitar de nuevo el sistema de almacenamiento y mantener el catálogo de ficheros actualizado.

Durante esta operación de inicialización es posible que se añadan nuevos nodos o, al contrario, es posible que un nodo antes activo no lo esté ahora, por eso se requiere de un proceso de reconstrucción del catálogo de cada nodo para mantener coherente la información asociada al catálogo de cada nodo.

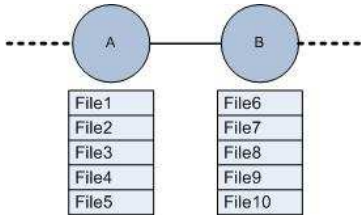

Fig. 29 Estado inicial antes de un reinicio en el sistema.

Conexión de un nuevo nodo al sistema: la inserción de un nodo al sistema requiere la actualización del catálogo del nodo "padre" ya que parte de los ficheros que gestiona pasarán a ser catalogados por el nuevo nodo.

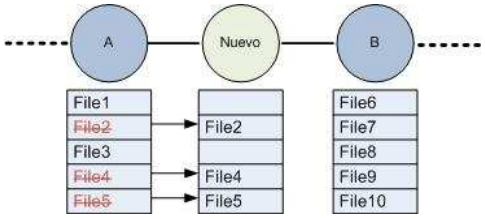

Fig. 30 Reconstrucción tras la inserción de un nodo.

Desconexión de un nodo: la desactivación de un nodo inicia un proceso de recuperación que termina con la actualización del catálogo del nodo vecino inferior del que falla. El sistema de backup introduce en el catálogo los ficheros que administraba el nodo fallido.

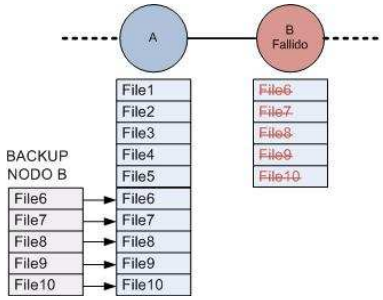

Fig. 31 Reconstrucción del catálogo después del fallo de un nodo.

5.7 Módulo de Operaciones.

El siguiente apartado describe el diseño de cada una de las operaciones que implementa DFS y que pueden ser clasificadas dependiendo si pueden ser ejecutadas por las aplicaciones clientes (operaciones de almacenamiento) o por entre los nodos DFS (operaciones de almacenamiento y mantenimiento del sistema).

Para comprender mejor el proceso de ejecución de las operaciones se explican a continuación algunos términos que se utilizarán más adelante.

- **Aplicación cliente**: se refiere a la aplicación que utiliza el API DFS de un nodo (nodo cliente) para invocar una operación. Para realizar esto, la aplicación envía un mensaje del tipo operación junto con los parámetros de entrada.

- **Nodo cliente**: se trata del nodo con que contacta una aplicación cliente y que será el encargado de iniciar el proceso de ejecución; para llevar a cabo la operación es posible que este nodo requiera contactar con otros nodos DFS.

- **Nodo propietario de un fichero**: es el encargado de almacenar toda la información metadato de un fichero, por eso en la mayoría de situaciones se requerirá contactar con el nodo propietario de un fichero, para buscar el localizador de una réplica o actualizar su información metadato.

- **Nodo réplica**: son aquellos nodos que almacenan una réplica de un fichero aunque no sean propietarios del fichero.

- **Fichero inválido**: un fichero marcado como inválido en el catálogo de un nodo significará que el fichero fue almacenado alguna vez por el nodo pero una posterior operación de borrado o escritura invalidó el fichero para ese nodo. Por esta razón, el nodo cliente debe contactar finalmente con el nodo propietario que mantiene la información de los localizadores de réplicas.

- **Fichero dirty**: se trata un fichero del que únicamente se tiene una réplica en todo el sistema y por tanto las operaciones de escritura sobre el nodo de réplica son inmediatas sin necesitar la intervención del nodo propietario.

A continuación se analiza la lógica de control de cada operación; se comenzará describiendo las operaciones públicas a nivel de interfaz y accesibles a las aplicaciones clientes, posteriormente se detallan las operaciones entre nodos que tienen que ver con el mantenimiento del sistema.

5.7.1 get(String virtualFileName).

La operación get permite que las aplicaciones recuperen ficheros del sistema de almacenamiento por medio de un identificador de fichero único a todo el sistema. La Fig. 32 muestra el diagrama de control para una operación get, del esquema puede deducirse que la ejecución de esta operación puede involucrar a más de un nodo del sistema a parte del nodo inicial de conexión al cliente.

A continuación se enumeran en detalle los pasos que tienen lugar tras la invocación de una operación get.

1. El nodo cliente (nodo A) recibe un mensaje de operación get que contiene el nombre del fichero a recuperar. El nodo consulta el sistema de catálogo para averiguar si el fichero es válido y puede descargarse directamente del módulo de almacenamiento.

2. Si el fichero no es válido el nodo hace uso del sistema de búsqueda para averiguar que nodo es propietario del fichero. Si el nodo cliente es propietario del fichero, el control de ejecución sigue en los pasos 3, en caso contrario lo hace para los pasos 4.

3. En el caso de que el propietario sea el mismo nodo cliente entonces se comprueba la existencia de un nodo dirty o de una réplica del fichero. Si no se da esta última circunstancia, entonces significará que el fichero no existe en el sistema y se devolverá a la aplicación cliente un mensaje de error.

4. Con la referencia a un nodo de réplica o al nodo dirty se transfiere el control al sistema de almacenamiento que será el encargado de recuperar directamente el fichero desde alguno de estos nodos.

5. La copia del fichero recuperada será mantenida por el propio nodo cliente y por tanto se convierte en un nodo réplica del fichero. Finalmente se envía el fichero a la aplicación cliente.

6. En caso de que el nodo no sea propietario del fichero entonces se debe contactar con el nodo propietario para recuperar las referencias a los nodos réplica. Si el nodo propietario no devolviera ninguna referencia, entonces significará que no existe el fichero en el sistema y se informará a la aplicación cliente.

7. El nodo cliente selecciona uno de los nodos de réplica disponibles para transferir el fichero por medio del sistema de almacenamiento.

8. Finalmente el fichero se transfiere a la aplicación cliente.

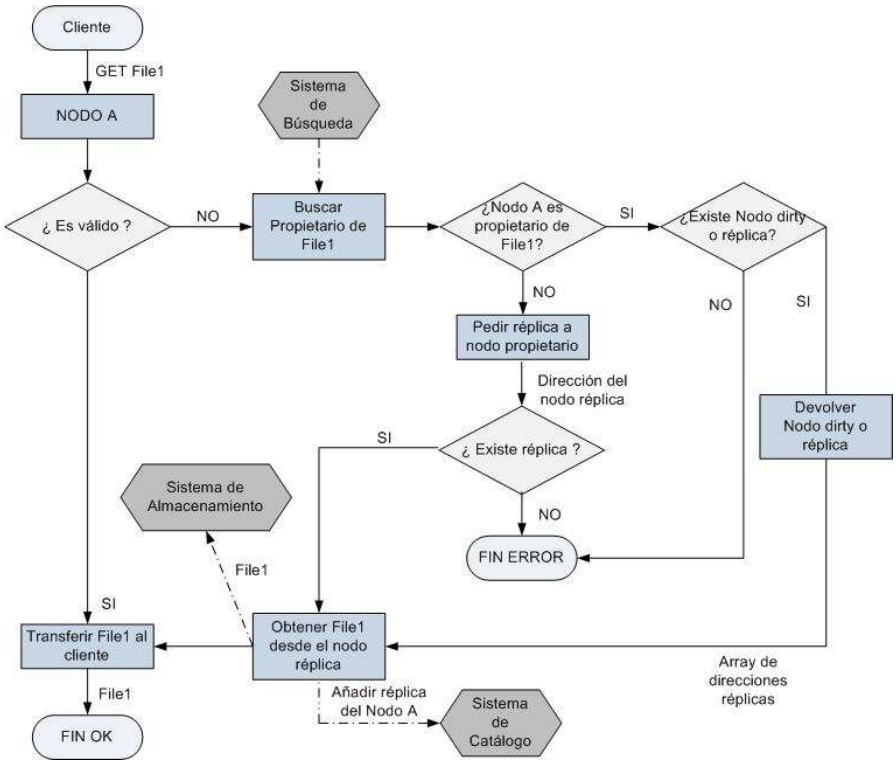

Fig. 32 Diagrama de flujo para una operación GET.

5.7.2 put(String virtualFileName, int n_replicas).

La operación put permite almacenar un fichero dentro del sistema además las aplicaciones clientes pueden especificar el número de copias que se mantendrán en el sistema. La Fig. 33Fig. 33 describe el diagrama de flujo para una operación put que implica la interacción con varios módulos.

Se describen a continuación los pasos que se realizan tras la invocación de la operación put.

1. La aplicación cliente envía un mensaje de solicitud de operación put a un nodo DFS conocido (nodo A) y espera la aceptación por parte del nodo. Si el estado del nodo permite la realización de la operación, entonces el cliente transmite el fichero al nodo A y espera la respuesta de final de operación.

2. El nodo A utiliza el módulo de búsqueda para obtener el nodo propietario del fichero que se quiere almacenar. En el caso de que el propio nodo A fuera propietario del fichero File1 y el cliente no solicitase replicación, entonces es necesario consultar si el fichero está marcado con el flag "dirty". Si el fichero es de tipo "dirty" significa que es la única copia válida que existe en el sistema y por tanto es posible realizar una operación de escritura sin tener que invalidar el resto de réplicas. Si este caso ocurre, el nodo A almacena finalmente el fichero por medio el sistema de almacenamiento.

3. Si el nodo A no es propietario del fichero pero el fichero no está marcado como "dirty" o el cliente pide replicación, entonces es necesario averiguar el nodo propietario por medio del sistema de búsqueda. Una vez obtenido el nodo propietario del fichero entonces el nodo A inicia una operación putDFS con éste nodo. El nodo propietario es el encargado de decidir por medio del módulo de heurística a qué nodos de almacenamiento deberían transferirse las nuevas réplicas.

4. Una vez se reciben los identificadores de réplica, el nodo A utiliza el sistema de almacenamiento para transferir cada réplica. Es importante recalcar que el encargado de decidir dónde debe replicarse el fichero es el propietario del fichero aunque finalmente sea el nodo A quien transfiere las réplicas.

5. En el caso de que el nodo A sea propietario del fichero entonces el fichero pasa a quedar bloqueado para otras operaciones de escritura aunque las operaciones de lectura del fichero continúen permitiéndose.

6. El nodo A es propietario del fichero y registra información histórica de los accesos al fichero, por otra parte estos datos son utilizados por el sistema de replicación para decidir que nodos son los más adecuados para recibir una copia del fichero. Una vez se ha seleccionado las direcciones de los nodos réplica entonces se utiliza el sistema de almacenamiento para transferir el fichero.

7. Cuando finaliza la transferencia de las réplicas entonces el nodo propietario necesita invalidar las copias obsoletas del fichero. Finalizado este proceso, el fichero se desbloquea para operaciones de escritura y puede ser accedido nuevamente con normalidad.

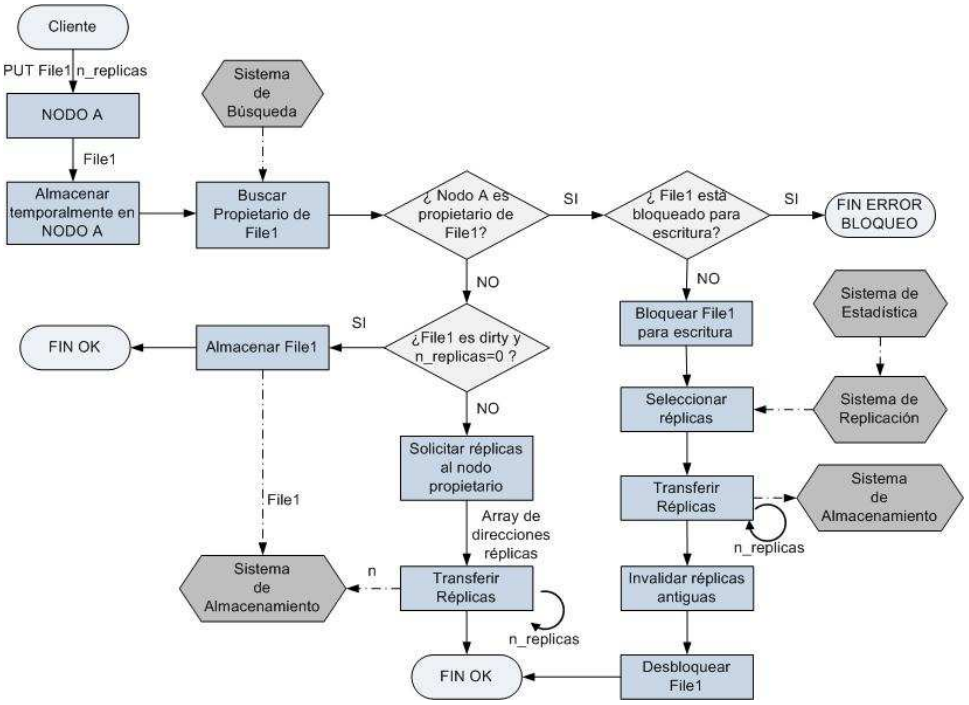

Fig. 33 Diagrama de flujo para una operación PUT.

5.7.3 del(String virtualFileName).

La operación del invalida un fichero y todas sus posibles réplicas distribuidas en el sistema de almacenamiento. El funcionamiento de esta operación es bastante sencillo ya que únicamente se necesita averiguar el propietario del fichero a eliminar. El proceso finaliza cuando el propietario invalida las réplicas desactualizadas.

5.7.4 show(String pattern, int depth).

La operación show permite recuperar un listado con los ficheros que están almacenados en el sistema de almacenamiento y concuerdan con un determinado patrón de búsqueda. Tal y como se comentó anteriormente, el sistema de nombrado de ficheros en DFS es flexible y permite integrar cualquier estructura de sistema de ficheros bien sea jerárquica (existen directorios y subdirectorios) o plana; no obstante, DFS permite diferenciar directorios dentro de un sistema de identificación de ficheros jerarquizado[9] y por eso la operación show incluye un parámetro opcional "depth" que puede ser utilizado para restringir el nivel que se alcanza en la visualización de los ficheros dentro de una estructura de directorios.

Cada fichero en el sistema DFS pertenece al catálogo de su nodo propietario y por eso cuando una aplicación cliente necesita buscar los ficheros que coinciden con un determinado patrón, la búsqueda debe realizarse sobre cada uno de los nodos del sistema.

A pesar que existen soluciones más eficientes que la mostrada en la lectura de directorios, finalmente se decidió mantener el mecanismo de "broadcasting", puesto que la operación de lectura de directorios es utilizada en muy pocas ocasiones en un sistema Fura. Básicamente es invocada cuando los usuarios de Fura acceden al portal para realizar un visualizado de sus directorios de tareas.

Por otra parte el número de nodos DFS máximos previsto para una instalación Fura hacen viable la invocación ocasional de operaciones tipo show.

[9] DFS utiliza el carácter '/' para realizar la separación de directorios, cualquier otro carácter no es tenido en cuenta para realizar la distinción entre subdirectorios.

La Fig. 34 muestra de modo gráfico el funcionamiento de la operación show. Tal y como se ha comentado, se inicia una búsqueda recursiva determinista sobre el anillo de búsqueda devolviendo el listado de ficheros concordantes junto al objeto propiedades asociado a cada fichero.

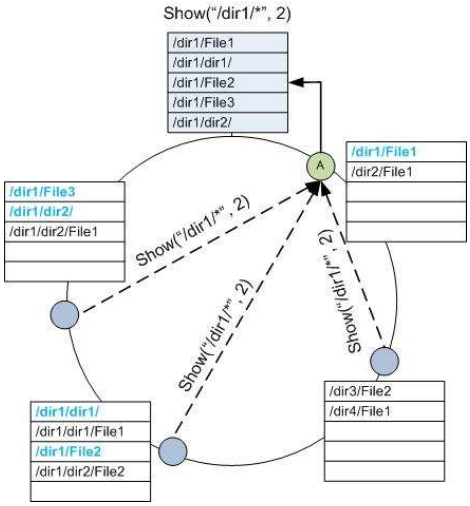

Fig. 34 Operación de Show para el patrón "/dir1/ y nivel de búsqueda 2.

5.7.5 Operaciones de Transmisión de ficheros: getDFS(String virtualFileName), putDFS(String virtualFileName, int n_replicas) y delDFS(String virtualFileName).

Estas tres operaciones tienen una lógica de funcionamiento similar a las operaciones get, put y del excepto que son utilizadas internamente sólo por nodos DFS para transferir ficheros. De este modo, existen algunas diferenciaciones en cuanto a la información que los nodos almacenan como puede ser en operaciones de tipo getDFS dónde requiere enviar el identificador del nodo que inicia la operación.

5.7.6 Operaciones del Sistema de Búsqueda.

5.7.6.1 getOwner (String key) y getAbsolute(String key).

Dada una clave en el rango de claves del anillo de búsqueda, la operación getOwner proporciona el identificador del nodo que más se aproxima por debajo a una clave dada. Se trata de una operación no determinista ya que no asegura que el nodo retornado se haga cargo de la clave buscada y debe emplearse conjuntamente con la operación getAbsolute que sí proporciona una búsqueda determinista del propietario de una clave[10].

getAbsolute es una operación que invocan internamente los nodos DFS cuando requieren averiguar el propietario de un fichero. El funcionamiento de esta operación queda descrito en los siguientes pasos.

1. Un nodo inicia el proceso de búsqueda de la identidad del nodo propietario de una clave.

2. Si el nodo actual es propietario de la clave K entonces se devuelve la dirección IP del propio nodo. Para comprobar la propiedad de la clave, se chequea si el identificador de clave buscado se encuentra entre el identificador de nodo actual y el de su vecino inmediatamente superior.

[10] La clave de una operación de tipo búsqueda puede ser indistintamente el identificador de un nodo o de un fichero.

3. Si el nodo actual no es el nodo propietario de la clave K entonces debe realizarse una búsqueda en el módulo de cache del sistema de búsqueda. El proceso continua seleccionando el nodo más próximo por abajo a la clave buscada.

4. Una vez obtenida la identidad del nodo conocido más próximo a la clave, se inicia un proceso iterativo que consiste en preguntar al nodo candidato si es propietario de la clave. En caso de que no serlo el nodo devolverá la identidad del nodo superior más próximo a la clave. La operación getOwner es la encargada de resolver esta consulta.

5. El proceso finaliza cuando el nodo preguntado devuelve con su mismo identificador lo que supone que este nodo es el propietario de la clave buscada.

5.7.6.2 initializeRing(String KnownNode) y joinRing(String NodeIP).

Cuando un nuevo nodo quiere conectarse al sistema de almacenamiento, se ejecuta la rutina initializeRing.

Primeramente el nuevo nodo contacta con alguno de los nodos conocidos y activos en el sistema con el objetivo de averiguar su nodo "padre". El nodo "padre" es el encargado de realizar la conexión del nuevo nodo y asignar los identificadores de los nodos que serán vecinos al nodo que se debe insertar en el sistema. Precisamente el identificador de este nodo deberá ser el inmediatamente inferior posible al nuevo nodo.

La Fig. 35 muestra la interacción entre nodos para una operación de conexión del nodo B. A continuación se explican cada uno de los pasos que se llevan a cabo tras la conexión de un nuevo nodo.

1. El nodo B utiliza el sistema de búsqueda para encontrar el identificador del nodo encargado de gestionar una solicitud de conexión al anillo, que en el caso de la Fig. 35 corresponde al nodo A.

2. El nodo B invoca la operación joinRing sobre el nodo A que se encarga posteriormente de recuperar la porción del catálogo y los k vecinos superiores asociados al nuevo nodo B.

3. El nodo B actualiza su catálogo junto con la lista de sus vecinos superiores además envía un mensaje a su vecino superior (nodo C) indicándole el cambio de relación entre sus nodos vecinos.

4. Si el proceso anterior terminó correctamente entonces el nodo B envía una confirmación al nodo A que finalmente actualizará su catálogo y creará un enlace a su nuevo vecino superior.

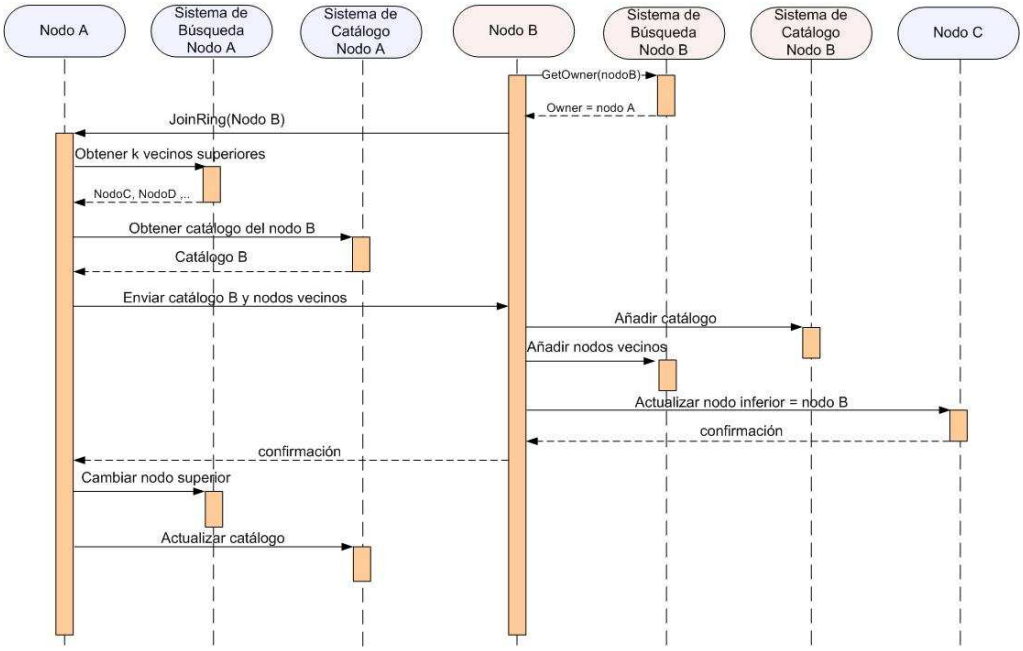

Fig. 35 Diagrama de secuencia para una operación de inicialización de un nodo en el anillo de búsqueda.

5.7.6.3 neighborOperation(int typeOp,String nodeIP, boolean recursiveCount).

Esta función permite realizar operaciones de mantenimiento sobre la cache de nodos del anillo de búsqueda. Se trata de una operación recursiva ya que permite retransmitir la operación un cierto número de veces hacia los nodos vecinos. A continuación se listan las tres operaciones que implementa esta función.

changeMinorNeighbor: cambia el identificador del vecino inferior del nodo actual por el identificador que se indica como parámetro de la función.

changeMajorNeighbor: cambia el identificador del vecino superior del nodo actual por el identificador que se indica como parámetro de la función.

addNeighbor: introduce un nuevo nodo en la cache de nodos del sistema de búsqueda.

5.7.6.4 statusRing(String FirstNode).

Esta función permite monitorizar la conectividad del anillo de búsqueda ya que devuelve una lista de enlaces activos entre nodos.

La operación es iniciada sobre un primer nodo que recursivamente la transmite hacia sus nodos vecinos inferiores hasta completar el anillo de búsqueda. En el caso de que la lista de nodos no esté completa significara que existe un fallo en alguno de los enlaces entre nodos y debe iniciarse un proceso de recuperación.

5.7.6.5 manageRing(int typeOp, boolean recursive).

La operación manageRing pone al nodo en un estado de actividad determinado por el parámetro de entrada typeOp, además la operación puede ser propagada hacia nodos vecinos de forma recursiva. Las operaciones que pueden efectuarse sobre un nodo tienen que ver con el cambio del estado de actividad de nodo tal y como se indicó en la Fig. 16. A continuación se muestra el listado de operaciones que son posibles realizar sobre el nodo.

setPersistence: esta operación permite almacenar de modo persistente la información de control del catálogo y la del sistema de búsqueda en tres ficheros binarios.

setBuilding: esta operación realiza una reconstrucción del catálogo tras un reinicio de un nodo.

setOnline: pone el nodo en el estado Online permitiendo una funcionalidad completa.

setShutdown: pone a un nodo en el estado de inactividad realizando primeramente una operación de persistencia de datos.

5.7.6.6 notifyFailedNode(String nodeIP).

Los nodos utilizan esta operación para informar al resto del fallo de uno de los nodos del anillo de búsqueda. Cada vez que un nodo recibe esta operación, elimina del sistema cualquier referencia al nodo caído, esto implica interaccionar con los sistemas de búsqueda, catalogación y estadística.

El mensaje de notificación es enviado por el nodo "padre" ya que es el encargado de monitorizar el estado del nodo vecino superior, además para mantener la coherencia del sistema de búsqueda el mensaje ha de ser retransmitido recursivamente por todos los nodos del anillo de búsqueda.

5.7.6.7 keepAlive().

Todos los nodos del sistema envían cada cierto tiempo un mensaje a su nodo vecino inferior que permite determinar si el nodo continúa activo o por el contrario debe iniciarse una operación de recuperación del anillo de búsqueda.

5.8 Módulo de Backup.

El sistema de catálogo contiene toda la información lógica de la existencia de un fichero en el sistema y por eso es importante que esta información se resguarde ante posibles fallos. El módulo de backup es el encargado de almacenar la información del sistema de catálogo y del sistema de búsqueda que podrá ser recuperada cuando no esté accesible.

DFS implementa un sistema de respaldo que consiste en disponer de una copia del catálogo del nodo sucesor que se actualiza cada vez que varía un dato en el catálogo del nodo sucesor.

El funcionamiento del sistema de backup DFS es el siguiente:

Existe un buffer que utilizan las operaciones del sistema para almacenar temporalmente las líneas del catálogo que se han modificado desde la última vez de backup. Cada cierto tiempo configurable por el administrador, el contenido del buffer es enviado en bloque al nodo vecino inferior para que lo almacene en su repositorio de backup.

En determinadas ocasiones es necesario realizar un backup del catálogo del nodo vecino sin esperar ningún intervalo de tiempo. Estas situaciones pueden estar obligadas por ejemplo al sobrepasar el máximo número de registros disponibles en el buffer o cuando se requiera inmediatez para el respaldo de datos en circunstancias excepcionales como la inserción de un nuevo nodo, fallo de un nodo o reconstrucción del catálogo en la inicialización del anillo de búsqueda.

El proceso de restauración del catálogo de un nodo es simple ya que sólo requiere incorporar los registros de ficheros del nodo fallido al catálogo del nodo que inicia la recuperación. Tras este proceso el nodo vuelve a hacerse cargo del backup del nuevo nodo vecino.

5.9 Módulo de Transferencia.

El módulo de transferencia permite abstraer al resto de componentes del sistema de las tareas de transmisión de los ficheros ya que las comunicaciones se realizan directamente al mismo nivel de módulo de los nodos implicados.

El componente operación hace uso de dos únicas funciones que ofrece el módulo transferencia para enviar o recibir ficheros desde o hacia un nodo remoto y son las siguientes

putRemoteFile(String localFileName, String virtualFileName, String remoteNode): permite enviar un fichero identificado en el sistema por el parámetro *virtualFileName* y almacenado localmente en la ruta *localFileName* al nodo remoto.

getRemoteFile(String localFileName, String virtualFileName, String remoteNode): permite recuperar un fichero identificado en el sistema por el parámetro *virtualFileName* y almacenarlo localmente en la ruta *localFileName* desde el nodo remoto.

Es importante reseñar que el parámetro *remoteNode* de las funciones anteriores no se refiere a la máquina encargada de almacenar físicamente el fichero sino al nodo que gestiona las ubicaciones finales del fichero. En este sentido la Fig. 36 muestra este aspecto que permite la abstracción tanto de la ubicación física del almacenamiento del fichero como del protocolo usado para su transferencia.

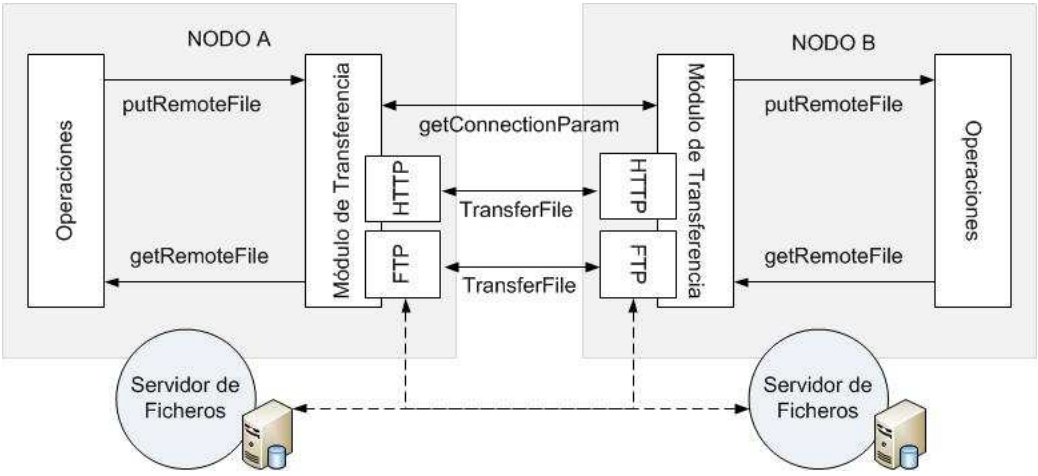

Fig. 36 Abstracción en la transferencia de ficheros entre nodos.

A continuación se explica con más en detalle los pasos que tienen lugar en la transferencia de ficheros Fig. 36.

1. El módulo de operaciones hace uso de dos funciones para enviar o recibir un fichero; tras la llamada a la función el proceso de transferencia continua de modo transparente a la operación que lo invocó.

2. Dentro del módulo de transferencia el siguiente paso es contactar con el sistema de transferencia del nodo remoto para recuperar los parámetros que permitirán establecer una conexión directa con el dispositivo de almacenamiento remoto. Los parámetros que se reciben son estos dos:

   protocolType: indica el protocolo de transferencia que el nodo actual debe utilizar.

   protocolParams: contiene los parámetros de conexión específicos para el tipo de protocolo de transferencia especificado en el parámetro anterior. El contenido de este campo queda oculto al propio módulo transferencia ya que serán los submódulos específicos de cada protocolo los encargados de interpretar los parámetros de conexión.

3. El nodo emisor hace uso del submódulo que implementa el protocolo de transferencia indicado por el parámetro *protocolType*. Este argumento contiene los parámetros de conexión específicos para recuperar o almacenar un fichero con éxito; Así por ejemplo para un protocolo FTP el parámetro contendría los siguientes argumentos: servidor ftp, usuario de acceso, clave cifrada, ruta al fichero en el servidor ftp. Para un protocolo basado en sockets los argumentos se limitarían a indicar la ruta completa del fichero en la máquina remota.

4. Una vez almacenado el fichero en el dispositivo remoto o recuperado en el sistema local de la máquina, se devuelve el control de ejecución a la operación que inició la transferencia.

Capítulo 6 – Integración del sistema DFS con Fura

En el capítulo 3 se realizó un resumen de la funcionalidad más importante que incorpora el software Fura. También se realizó una breve descripción de su sistema de almacenamiento y se analizaron los inconvenientes que introducía el diseño centralizado de almacenamiento Fura.

Con el objetivo de comprender mejor el proceso de integración de la arquitectura propuesta, se realizará una breve explicación sobre cómo interaccionan los componentes Fura con el sistema de almacenamiento.

Cualquier despliegue Fura que preste servicio a una OV dispone de dos tipos de entidades como son varios agentes y un único servidor Fura. Los agentes Fura se instalan sobre cada uno de los recursos que forman el Grid de la OV y son los encargados de monitorizar el estado de carga del recurso y ejecutar tareas en él cuando el nivel de carga de trabajo del recurso se encuentra por debajo de un umbral definido por el administrador. Los agentes solicitan tareas al servidor cuando se encuentran en un estado bajo de carga y son los mismos agentes los encargados de iniciar la transferencia de los ficheros asociados a la tarea a ejecutar. Una vez finaliza la ejecución de una tarea sobre el Grid, los ficheros de resultados son enviados al servidor que los almacena en su sistema local. La Fig. 37 representa un esquema general de las comunicaciones entre agentes Fura y el servidor para la gestión de ficheros.

Para realizar la transmisión de los ficheros bien por parte de los agentes o directamente por los usuarios de Fura, el servidor implementa dos Apis que exponen una serie de operaciones de transferencia y gestión de ficheros similar a la utilizada en un sistema local de almacenamiento.

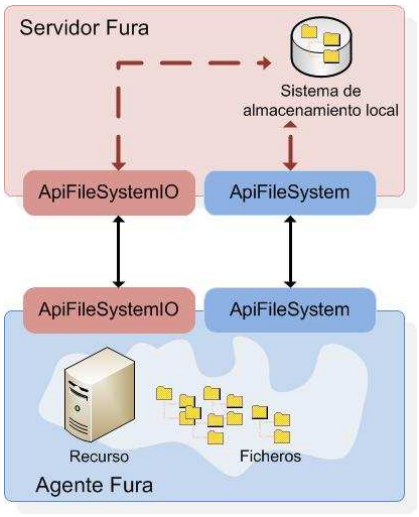

Fig. 37 Esquema de comunicaciones antes de la integración con el sistema DFS.

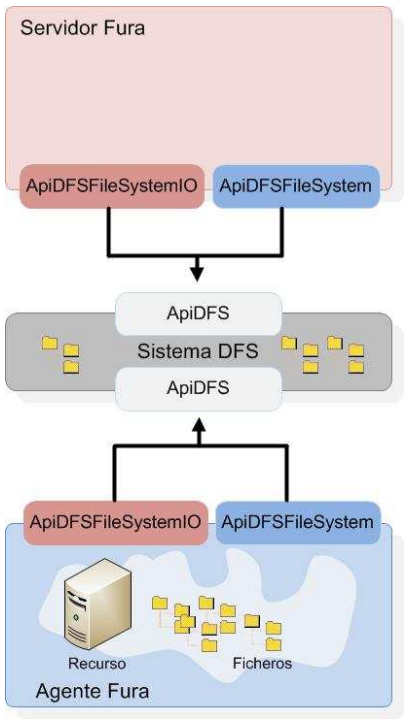

Fig. 38 Esquema conceptual del proceso de integración obtenido al sustituir el sistema de almacenamiento Fura por DFS.

La integración del sistema DFS con el software Fura consistió en implementar dos plugins que replicasen completamente la funcionalidad expuesta por las APIs ApiFileSystemIO y ApiFileSystem de Fura sobre un despliegue DFS. La Fig. 38 muestra un esquema del proceso de integración y en él se puede observar como los nuevos dos plugins implementados exponen su funcionalidad por medio de los api's ApiDFSFileSystemIO y ApiDFSFileSystem.

Básicamente la implementación de los plugins realiza una traducción de cada una de las operaciones que ofrecía el sistema de almacenamiento Fura en su anterior plugin, a invocaciones sobre las operaciones del API que expone el sistema DFS. En el momento de la integración existieron un conjunto de operaciones de Fura que ofrecían una traducción directa a operaciones DFS, pero otro grupo de operaciones requirieron una integración más complicada ya que implicaba la invocación de varias operaciones individuales DFS. Para comprender mejor esto, la Fig. 39 muestra un ejemplo de implementación de la operación download cuya implementación es inmediata ya que únicamente cambia respecto a la operación original Fura la invocación de la operación get sobre el api DFS.

```
 /**
  * Reads a file and return its content as an attachment.
  *
  * {@inheritDoc}
  */
public DataHandler downloadFile(String fileName) throws KernelException {
   DataHandler dataHandler = null;
   KernelContext.getContext().setEffectiveUser(Configuration.getPrivilegedUser());
   File f = RixosFileManager.getInstance().createBaseDirectory();

   String rixosLocalPath =  ApiFileSystemImpl.normalize(Configuration.rootDFSPath + "/" + fileName);
   String localPath = new File(rixosLocalPath).getAbsolutePath();

   new RepositoryFile(f.getParent()).mkdirs();

   DFSClientLibrary.get(localPath, virtualFileName)

   dataHandler = new DataHandler(new FileDataSource(new File(localPath)));

   return dataHandler;
}
```

Fig. 39 Extracto de código para la integración de una operación simple de trasferencia de fichero.

La Fig. 40 muestra un ejemplo de la operación *mkdirs* que requiere una integración más compleja que la anterior ya que el API DFS no dispone de una operación de creación de ficheros en bloque y por tanto es necesario realizar mediante un proceso iterativo la creación de los directorios sobre el sistema DFS.

```
/**
  * {@inheritDoc}
 * @throws RemoteException
 */
 public boolean mkdirs(String directory) throws RemoteException {
    ApiUtils.checkNullParam(directory, "directory");

    KernelContext.getContext().setEffectiveUser(Configuration.getPrivilegedUser());
    File f = RixosFileManager.getInstance().createBaseDirectory();
    String base = RixosFileManager.getInstance().getBaseDirectory(f);
    String virtualDirectoryName = RixosFileManager.getInstance().normalize(directory, true);
    String rixosDirectory = Configuration.rootDFSPath + "/" + base + virtualDirectoryName;

    ApiUtils.checkNullParam(directory, "directory");

   String baseUploadDirectory = RixosFileManager.getInstance().downloadDFSDirDownTree(base,   virtualDirectoryName);
   KernelContext.getContext().clearEffectiveUser();
   new RepositoryFile(rixosDirectory).mkdirs();
```

```
RixosFileManager.getInstance().normalize(baseUploadDirectory, true);

File[] files = f.listFiles();
   if (files != null)
      for (int i=0; i<files.length; i++){
         if (files[i].isDirectory()) {
            uploadDirectoryName = normalize(virtualFileName + "/" + files[i].getName(), true);
            uploadDirectoryUpTree(base, uploadDirectoryName, true, uploadAll);
         } else if (uploadAll || files[i].getName().startsWith(IxosSecurityManager.SECURITY_PREFIX)) {
            uploadDirectoryName = normalize(virtualFileName + "/" + files[i].getName(), false);
            uploadDirectoryUpTree(base, uploadDirectoryName, false, uploadAll);
         }
      }

 String localDirectoryName = new File(Configuration.repositoryDirDFS, base +   virtualFileName).getAbsolutePath();
 uploadDirectoryName = normalize(virtualFileName, true);

 DFSClientLibrary.put(localDirectoryName, uploadDirectoryName)

KernelContext.getContext().setEffectiveUser(Configuration.getPrivilegedUser());
RixosFileManager.getInstance().deleteTmpDirectory(f);
KernelContext.getContext().clearEffectiveUser();

return true;
}
```

Fig. 40 Extracto de código para la integración de la operación mkdirs de Fura en el sistema DFS.

Para realizar el testeo de la correcta integración del prototipo DFS en el sistema Fura, se utilizó una batería de tareas de prueba que se ejecutaron sobre un despliegue Fura compuesto por 25 agentes Fura. Cada tarea ejecutó un procedimiento previo de lectura de un fichero de texto y generó como salida otro fichero con las líneas invertidas. Al mismo tiempo, se realizó la instalación y configuración del sistema DFS desplegando la cantidad de 50 nodos sobre un cluster de PCs, además cada agente se configuró para interaccionar con uno de los nodos DFS.

Debido a que para desplegar el sistema DFS se utilizó un cluster de altas prestaciones, la selección del nodo DFS al que debe conectarse cada agente Fura no es relevante ya que, todos los nodos DFS están igual de próximos a los agentes si tenemos en cuenta la distancia en términos de saltos de red. Para un sistema Fura en producción la configuración idónea sería que cada agente Fura se configurara para conectarse al sistema DFS por medio del nodo DFS más próximo a él.

La integración del sistema de almacenamiento DFS con Fura se realizó en las siguientes dos fases:

En una primera fase de la integración, se mantuvieron los plugins originales de Fura junto a los implementados para DFS de manera que los agentes continuaron utilizando el plugin Ixos de Fura para descargar o subir ficheros al servidor de Fura. Para cada una de las operaciones del plugin Ixos se duplicó la funcionalidad para los dos sistemas Fura y DFS, de este modo cada llamada a una operación de almacenamiento de Fura supuso la ejecución de dicha operación sobre el propio sistema de almacenamiento local del servidor Fura y además la ejecución de la operación de transferencia sobre el sistema DFS. Terminada la batería de pruebas se comprobó que el contenido del sistema de ficheros almacenado en el servidor Fura era similar al contenido de ficheros que fueron almacenados en el sistema DFS.

La segunda fase de integración consistió en la sustitución de los plugins Fura por los nuevos plugins DFS tanto en los agentes como en el servidor Fura. Después de finalizar esta segunda fase de integración, se realizó nuevamente el testeo del sistema obteniendo los resultados satisfactorios.

De esta manera el prototipo del sistema DFS se ha quedado correctamente integrado sobre la plataforma Fura y prestando por completo la funcionalidad que la arquitectura DFS ofrece tal y como se ha descrito en esta Tesina de Máster.

Capítulo 7 – Conclusiones

El trabajo de Tesis de Máster presentado en este documento describe la arquitectura y la implementación de un sistema de almacenamiento de ficheros que utiliza las tecnologías distribuidas P2P para ofrecer a las aplicaciones Grid, una plataforma de almacenamiento de alta disponibilidad y tolerante a fallos.

A pesar que la arquitectura propuesta fue inicialmente pensada para resolver los problemas de centralización del almacenamiento de un software Grid particular como es Fura, la arquitectura es lo suficientemente robusta para ser utilizada como plataforma de almacenamiento para un sistema con ámbito de aplicación genérico.

La realización de este trabajo de Tesis de Máster ha permitido demostrar cómo la utilización de sistemas centralizados para el almacenamiento de ficheros está acotada a un volumen pequeño de transacciones en el sistema. Estos sistemas además de ser poco escalables, suelen ser sistemas que no permiten obtener alta disponibilidad en el acceso a los datos. Este trabajo permite mostrar que es posible convertir un sistema centralizado, poco escalable y con una tolerancia baja a fallos (Fura) en una arquitectura distribuida de almacenamiento robusta, adaptable y eficiente en el acceso a los ficheros.

Por una parte, la característica de escalabilidad que introducen los sistemas distribuidos ha permitido implementar un sistema que adapte de modo flexible su infraestructura de almacenamiento en función del volumen de actividad existente en tiempo real. Por otra parte, la utilización de un sistema descentralizado P2P permite ofrecer tolerancia a fallos de los componentes del sistema, haciendo posible un sistema de almacenamiento de alta disponibilidad ya que los posibles fallos en los dispositivos de almacenamiento no implican la pérdida de datos o la inaccesibilidad temporal al sistema.

Por medio de la arquitectura presentada en esta tesina, se ha podido solventar algunas de las desventajas que adolecen los sistemas de almacenamiento distribuidos. Así por ejemplo, se ha diseñado un sistema que utiliza los mismos mecanismos de coherencia de cache de los sistemas multiprocesadores, para extrapolar estas mismas técnicas a los sistemas distribuidos. Por medio de la adaptación de estos algoritmos, la arquitectura DFS permite asegurar, en tiempo real, la coherencia de los ficheros que se almacenan.

Otra de las principales desventajas de los sistemas distribuidos es la ineficiencia en la localización de la información contenida en él. En el caso de un sistema de almacenamiento este punto puede ser crítico cuando el volumen de ficheros o de nodos que se encargan de almacenarlos es elevado. La arquitectura DFS propone un mecanismo inteligente de almacenamiento de los ficheros, consistente en aproximar los ficheros a los nodos que se prevé recibirá, en el futuro, más accesos para un fichero en particular.

Tras la definición de la arquitectura y la implementación del prototipo funcional, el sistema fue integrado de forma satisfactoria en el sistema Fura realizando pruebas de funcionalidad con volúmenes de trabajos medios.

Como conclusión del trabajo realizado en esta tesina de Máster, se puede decir que la utilización de los sistemas distribuidos junto a las tecnologías Grid permite implementar herramientas lo suficientemente potentes para ofrecer soporte de almacenamiento a sistemas que actualmente requieren características específicas de disponibilidad y tolerancia a fallos.

Capítulo 8 – Trabajo Futuro

La arquitectura presentada en esta tesina supone una primera aproximación a la resolución de los problemas de almacenamiento de una herramienta Grid como es Fura. Sin embargo, a pesar de disponer de un prototipo totalmente funcional e integrado en la plataforma Grid, existen algunas tareas que podrían realizarse en un futuro con el objetivo de mejorar su funcionalidad y eficiencia.

A continuación se exponen algunas posibles mejoras que pueden hacerse a la arquitectura y prototipo presentado.

- **Mecanismos de seguridad**. La seguridad en el acceso a los ficheros es un punto importante para cualquier sistema que requiera salvaguardar la identidad y contenido de los ficheros que almacena. La arquitectura propuesta no ofrece ningún mecanismo para autentificar y autorizar a los elementos que se agregan al sistema de almacenamiento. Por otra parte, el acceso y la integridad del contenido de los ficheros almacenados, queda restringido únicamente por las políticas de seguridad internas de los nodos que se ocupan de almacenarlos. Existen en la literatura varias alternativas que resuelven todos estos aspectos y que podrían ser aplicadas de forma flexible al sistema propuesto [[21].

- **Escalabilidad en redes WAN**. El prototipo implementado tiene un ámbito de aplicación específico para las características de la plataforma Grid en donde se ha integrado (Fura). Esto implica ciertas limitaciones que fueron asumidas a priori como pueden ser el número de nodos máximo necesarios para gestionar el sistema (aprox. 100), el tipo de red de comunicación utilizada es local y propietaria a la organización dónde se despliega el sistema, las operaciones realizadas sobre los ficheros son mayoritariamente de lectura/escritura y en menor proporción las de listado de directorios, etc.

  Desplegar el sistema DFS al margen de estas restricciones implicaría dotar a la arquitectura de otros mecanismos de localización de ficheros más específicos. Como se comentó, el sistema utilizado para la búsqueda distribuida de los ficheros es una simplificación de la arquitectura Pastry. Debido al diseño modular y estratificado de los componentes de la arquitectura DFS, es posible agregar fácilmente nuevos módulos sin interferir en la funcionalidad del resto de componentes del sistema.

- **Eficiencia de las transferencias**. El propósito de este trabajo se limitó a la implementación funcional de un prototipo que diera servicio a la arquitectura DFS y por tanto queda fuera de este trabajo, la implementación de mecanismos para mejorar la eficiencia en las comunicaciones y transferencias de los ficheros. No obstante y aún sin haber realizado un estudio exhaustivo de la eficiencia del sistema para cargas de trabajo elevadas, se han diseñado diferentes mecanismos para mitigar posibles cuellos de botella que puedan provocar un estado ineficiente en el sistema. Alguna de esas posibles técnicas son el propio algoritmo de aproximación de los ficheros a los posibles clientes, que reduce el número y tiempo de las transferencias de los ficheros a los clientes.

  Alguna de las posibles mejoras podría ir encaminadas a optimizar las búsquedas de directorios. La arquitectura DFS permite una solución sencilla ya que la localización recursiva de los ficheros en una estructura de directorios, puede ser solventada haciendo que los directorios sean tratados en el sistema del mismo modo que los ficheros. De esta forma, cada directorio estaría asignado a un nodo propietario que contendría información acerca de que nodos contienen ficheros pertenecientes al directorio. De esta forma en sistemas en que la operación de lectura de directorios es habitual, la operación podría ser tratada de forma más eficiente.

Referencias Bibliográficas

[1] I. Foster and C. Kesselman. The Grid: Blueprint for a new computing infrastructure. Morgan Kaufmann Publishers, Morgan Kaufmann Publishers, San Francisco, USA, 1 edition. 2003.

[2] M. Armbrust, A. Fox, R. Griffith, and A. Joseph, Above the clouds: A berkeley view of cloud computing*, 2009.*

[3] Armbrust, M., Fox, A., Gri_th, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.2009.

[4] EGGE's Replica Manager System. http://egee.itep.ru/User_Guide.html.

[5] Jean-Philippe Baud, James Casey, Sophie Lemaitre, Caitriana Nicholson, Graeme Stewart. LCG Data Management: From EDG to EGEE. CERN, European Organisation for Nuclear Research, 1211 Geneva, Switzerland.

[6] LCG File Catalog administrators' guide. https://twiki.cern.ch/twiki/bin/view/LCG/.

[7] Globus Replica Location Service. http://www.globus.org/rls/

[8] The DataGrid Architecture. 2001, EU DataGrid Project.

[9] The Apache Hadoop project. http://hadoop.apache.org/core/.

[10] Wuala by Lacie. http://www.wuala.com/.

[11] Dropbox storage online. https://www.dropbox.com/.

[12] Peter Drunschel and Antony Rowstron, PAST a large scale, persistent peer to peer storage utility. http://research.microsoft.com/en-us/um/people/antr/PAST/hotos.pdf.

[13] Microsoft Distributed File System. http://technet.microsoft.com/en-us/library/cc738688.aspx.

[14] A. Rowstron and P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329-350. 2001.

[15] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris and M. Frans Kaashoek. Comparing the Performance of Distributed Hash Tables Under Churn. Lecture Notes in Computer Science. pp 87-99, Volume 3279/2005. 2005.

[16] FreePastry. http://www.freepastry.org/

[17] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking. pp. 17-32. 2003.

[18] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph,and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE Journal On Selected Areas in Communications, vol. 22, nº. 1. 2004.

[19] GridSystems Fura. http://www.gridsystems.com/document_files/Fura_Product_Tour.pdf.

[20] Directory-based Cache Coherence. http://www.icsa.inf.ed.ac.uk/research/groups/hase/models/dir-cache/index.html.

[21] Dominik Grolimund, Luzius Meisser, Stefan Schmid, Roger Wattenhofer. Cryptree: A Folder Tree Structure for Cryptographic File Systems. 25th IEEE Symposium on Reliable Distributed Systems1. pp 189-198.

[22] Grid Systems S.A . http://www.gridsystems.com/

[23] Fura Project. http://www.gridsystems.com/?p=products/products.php&s=8

[24] I. Foster, The Grid Organizations and Problem-Solving Environments. Euro-Par 2001, pp 1-4, Volume 2150/2001. 2001.

[25] SOAP Protocol. http://www.w3.org/TR/soap/