Desarrollo de un conector de un sistema de ficheros paralelo para el sistema HDFS

Evaluación de una prueba de concepto utilizando Expand

Adrián Ruiz Arroyo

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Tutor del proyecto: Félix García Carballeira

Leganés, 26 de septiembre de 2017

Índice de contenidos

Resumen

La integración de nuevos sistemas de ficheros en Apache Hadoop es un proceso largo y complejo debido a que está programado en un lenguaje de muy alto nivel y su interfaz no se ajusta a ningún estándar popular entre estos sistemas.

Para paliar o dar solución a este problema, la propuesta es un conector genérico que oculte la interfaz para sistemas de ficheros proporcionada por Hadoop y suministre, en su lugar, una interfaz POSIX en el lenguaje de programación C.

Como prueba de concepto, se proporciona la conexión con el sistema de ficheros paralelo Expand, desarrollado por el Grupo de Arquitectura de Computadores, Comunicaciones y Sistemas (ARCOS) de la Universidad Carlos III de Madrid.

Agradecimientos

Este proyecto no habría sido posible sin el apoyo recibido de un montón de personas. Os agradezco enormemente haber conseguido que llegue hasta aquí y haber puesto todo vuestro esfuerzo y confianza en mí, incluso en los momentos difíciles.

Agradecerle también a la universidad y a mi tutor haberme proporcionado los medios y los conocimientos para poder llevar a cabo este proyecto, que me ha demostrado que, al final, los cinco años de lucha han merecido la pena.

Este proyecto es un antes y un después, tanto en experiencia como en la manera de afrontar las cosas. Espero mirar atrás con el tiempo y seguir sintiendo la misma emoción que durante el transcurso de su desarrollo.

Gracias a todos.

Capítulo 1

Introducción

Este capítulo presenta el contexto del trabajo realizado en este proyecto. El primer apartado expone las áreas a las que afecta y sienta los límites sobre lo que da como resultado. El segundo apartado muestra la realidad bajo la que se ampara en el momento de su realización. El tercer apartado declara los objetivos detrás del proyecto. En el último apartado, se incluye una breve indicación sobre la organización del contenido dentro del presente documento.

1.1. Ámbito y alcance

Este proyecto se enmarca en distintas áreas dentro del ámbito de las ciencias de la computación. El proyecto en sí requiere de especialización en el ámbito de los sistemas distribuidos, concurrentes y paralelos. Sin embargo, sus beneficios aplican a otros campos, como el análisis de datos y la inteligencia artificial. En concreto, este proyecto está dirigido a los desarrolladores de sistemas de ficheros paralelos.

Respecto a su alcance, el proyecto incluye el diseño técnico, la implementación de código, una prueba de concepto y la validación de los resultados de dicha prueba. Este documento forma parte también del material entregado e incluye el presupuesto para el proyecto, diversos análisis que destacan su utilidad y todo tipo de detalles sobre el desarrollo general del trabajo.

1.2. Contexto

El mundo de la computación está sufriendo en la actualidad la sacudida del fenómeno conocido como *hiperconectividad*. Esta realidad se resume en que las personas necesitan estar en conexión constante con las demás y con los sucesos a su alrededor. Aunque esta necesidad

no es nueva, la proliferación de soluciones tecnológicas para hacerlo posible está dando lugar a una exigencia de cada vez más y mejores medios para cubrirla.

La situación está afectando a múltiples campos de la informática, habiendo dado lugar a lo que se puede considerar un nuevo área más de estudio, el *big data*. Este concepto gana importancia como consecuencia de la hiperconectividad, aunque tampoco es nuevo: atiende a las técnicas de análisis y almacenamiento de conjuntos de información en contextos donde la cantidad de datos hace poco eficientes las aproximaciones tradicionales.

Las empresas, conscientes de esta situación, han tenido que adaptarse rápidamente para poder seguir siendo relevantes y aprovechar en su propio beneficio las circunstancias. Están realizando, de este modo, fuertes inversiones en *big data* que les permitan conseguir distintos objetivos, entre ellos:

- Solicitar y brindar información de todo tipo a gran cantidad de usuarios simultáneamente sin esperas.

- Analizar, con objetivos de seguimiento, marketing o estadística, los datos recabados de los usuarios en el menor tiempo posible.

- Emplear soluciones baratas, sencillas y eficientes que sirvan su propósito sin suponer una inversión desmesurada, incluso reutilizando recursos ya disponibles.

- Descentralizar la información, manteniendo las mismas condiciones, para poder prestar servicio a una base de usuarios más grande sin sofocar los sistemas informáticos.

Con este mercado en auge, han proliferado distintas herramientas que intentan satisfacer estas necesidades, estando aun en fase de adopción en la mayoría de casos. El perfil más demandado es el de un marco de trabajo que incorpora múltiples tecnologías y simplifica los procesos de *big data* haciendo invisible la complejidad interna al usuario final. Dentro de esta

categoría se encuentran Apache Hadoop y Apache Spark como máximo exponentes, aunque se están popularizando otras alternativas como Apache Storm, Apache Samza, Apache Flink y Apache Kafka. Como se puede comprobar, la comunidad de Apache está siendo muy activa en este campo.

Estas herramientas exigen un grado de sofisticación tecnológico que está incrementando con el tiempo. Uno de los pilares que sostienen su avance es el sistema de ficheros paralelo que gestiona el almacenamiento de la información. Almacenar y analizar *big data* en un tiempo prudente impone unas condiciones complejas de satisfacer manteniendo la consistencia de los datos. Es en este entorno en el que se sitúa el proyecto realizado.

1.3. Objetivos

El objetivo de este proyecto es proporcionar un conector genérico para Apache Hadoop que permita integrar en él sistemas de ficheros a través de una interfaz POSIX.

Este objetivo busca alcanzar otra serie de metas a través de su consecución:

- *Meta 1*. Facilitar la integración de nuevos sistemas de ficheros en Hadoop utilizando un estándar popular.

- *Meta 2*. Ampliar el espectro de sistemas de ficheros disponibles para Hadoop.

- *Meta 3*. Probar el funcionamiento en Hadoop del sistema de ficheros paralelo Expand.

- *Meta 4*. Poner fin a los largos procesos de migración de datos cuando el sistema de ficheros donde están almacenados no está disponible en Hadoop.

1.4. Contenido del documento

El objetivo de este documento es el análisis del proyecto realizado en todas sus dimensiones. A lo largo de sus capítulos, se realiza un análisis técnico del problema surgido y la solución propuesta, defendiéndola frente a sus alternativas. En los anexos, se analiza el propio desarrollo del proyecto y las cuestiones de índole económica y social.

- El **capítulo 1** sitúa el proyecto en su contexto, lo asocia a un ámbito, limita su alcance, presenta sus objetivos y organiza el contenido de este documento.

- El **capítulo 2** aporta una visión de todos los aspectos tecnológicos a los que se asocia y se enfrenta el proyecto en la actualidad, estudiando con detenimiento el diseño y la implementación de cada una de las tecnologías implicadas.

- El **capítulo 3** tiene como objetivo el análisis de la solución y, en consecuencia, del problema al que se asocia. Justifica y define la solución que se va a adoptar de entre una serie de candidatas propuestas.

- El **capítulo 4** ofrece las opciones de diseño para la solución, justifica la utilizada y analiza su implementación. Además, se define una prueba de concepto.

- El **capítulo 5** supone la validación y el volcado y análisis de resultados de la prueba de concepto. Se contraponen sus resultados a los de la alternativa disponible por defecto y se extraen las conclusiones pertinentes.

- El **capítulo 6** analiza el impacto que el lanzamiento del software tendrá sobre la sociedad y sobre la economía. Además, se presentan futuras líneas de trabajo para el proyecto.

- En los **anexos**, se exploran la planificación del proyecto, su presupuesto, el marco regulador que lo rodea y se adjunta un resumen extenso de este documento en inglés.

- Por último, se proporcionan la bibliografía y las conclusiones sobre el proyecto.

Capítulo 2

Estado de la cuestión

Este capítulo explora el estado de la cuestión tanto en sistemas de ficheros paralelos como en algoritmos de análisis de datos, que son las tendencias actuales en el área de *big data*. En el primer apartado, se describen los principales algoritmos de análisis de datos disponibles en la actualidad. En el segundo apartado se caracterizan los sistemas de ficheros paralelos de mayor repercusión. Se incluye dentro de esta categoría al sistema de ficheros paralelo Expand, diseñado e implementado en la Universidad Carlos III de Madrid. Por último, se describe la arquitectura de Apache Hadoop y la relación entre cada uno de sus componentes, investigando el enfoque que cada sistema de ficheros adopta para la integración.

2.1. Algoritmos de análisis de datos

Los algoritmos de análisis de datos en el contexto del *big data* cambian respecto a su enfoque tradicional con conjuntos de datos reducidos. En la década pasada, surgieron nuevos modelos y paradigmas de programación para dar soporte a las nuevas necesidades surgidas por el nacimiento de dicho área. Los más representativos son los siguientes:

2.1.1. MapReduce

MapReduce es un modelo de programación y una implementación asociada que se utiliza para el procesamiento y generación de grandes conjuntos de datos en sistemas distribuidos y paralelos representados por clústers[1]. Google fue el primero en utilizar este algoritmo para las operaciones internas de cálculo de PageRank, aunque a partir de 2014 descartó su uso[2]. La implementación realizada por esta empresa es propietaria.

En 2004, el modelo de programación pasó al dominio público tras la publicación del artículo "MapReduce: Simplified Data Processing on Large Clusters" por parte de Google[1]. A partir de esta fecha, comenzaron a aparecer las primeras implementaciones de código abierto.

2.1.1.1. Esquema de funcionamiento

El paradigma de MapReduce especifica dos funciones a ser definidas por el usuario: la función *map* y la función *reduce*[1]. A nivel interno, hace uso de tres funciones definidas por la implementación. El orden de operaciones final es el siguiente:

- **Preparación de la entrada para *map***. Se leen los datos de entrada del almacenamiento persistente de datos, se separan en fragmentos y se asigna una clave única k_1 a cada uno que los relaciona con un procesador del sistema.

- **Ejecución de *map***. Se ejecuta en cada procesador del sistema la operación *map* sobre su fragmento de los datos de entrada, lanzándose tantas operaciones en total como claves únicas k_1 definidas en el paso anterior. La salida de esta operación consiste en un conjunto de claves intermedias organizadas por una clave nueva, k_2.

- **Mezcla de la salida de *map***. Las salidas de las operaciones de *map* se organizan por listas de valores para cada k_2, asignándose cada una a un procesador *reduce*.

- **Ejecución de *reduce***. Al igual que en la función de *map*, cada lista de valores con clave k_2 se ejecuta en un procesador independiente. La salida intermedia de esta operación se organiza por una clave nueva, *k3*.

- **Generación de la salida**. Se colecta la salida intermedia del paso anterior y se ordena según su clave k_3, escribiéndose los resultados finales en el almacenamiento persistente de datos.

La razón por la que este sistema se puede paralelizar radica de la ejecución independiente de las operaciones de *map* y *reduce*, que solo operan sobre el fragmento de datos que le es asignado al procesador en que ejecutan. En conjuntos grandes de datos, típicos en *big data*, esta distribución del procesamiento entre nodos es útil para explotar el acceso paralelo a los datos y reducir el tiempo total de procesamiento.

2.1.1.2. Implementaciones de código abierto

Algunas de las implementaciones cuyo código está disponible para su consulta y modificación en la red son las siguientes:

- **Hadoop MapReduce**. El proyecto comunitario Apache Hadoop desarrolla una implementación de código abierto de MapReduce basada en el artículo original publicado por Google. Está escrita en el lenguaje de programación Java.

- **Twister**. MapReduce necesita almacenar los datos intermedios en el almacenamiento persistente para cada operación, lo cual tiene un impacto negativo en el rendimiento cuando se introducen algoritmos recursivos, ya que no se reutiliza información. La implementación de MapReduce realizada por Twister aporta eficiencia ante tales casos, reutilizando los datos estáticos entre iteraciones[3]. Está escrita en el lenguaje de programación Java.

La más popular es la implementación de Hadoop[4], que se encuentra en desarrollo activo. El desarrollo de Twister, por otro lado, no es muy activo, datando la última versión de 2011 y no habiendo alcanzado aun la versión 1.0.

2.1.2. Spark

Spark es un proyecto mantenido por la comunidad de Apache. Su objetivo es proporcionar un modelo de programación y una implementación para el análisis de datos que supere las limitaciones de MapReduce. Para ello, hace uso de estrategias de memoria compartida frente al almacenamiento persistente para las operaciones de análisis. Esta filosofía consigue un rendimiento muy superior al de MapReduce al no sufrir la latencia de los discos y evita los problemas de eficiencia derivados de la utilización de algoritmos recursivos[5].

Su funcionamiento gira entorno a una estructura de datos propia, denominada Resilient Distributed Dataset (RDD). Esta estructura alberga un multiconjunto de datos de sólo lectura y se distribuye a través del clúster mediante memoria compartida[6]. Para conseguir tolerancia a fallos en el acceso a la estructura, se ponen restricciones tanto en la manera en que se mantiene como en la operativa de la memoria compartida.

Este algoritmo de análisis de datos soporta en su modelo de programación diversos lenguajes de programación. La lista actual incluye Scala, Java, Python y R.

2.2. Sistemas de ficheros paralelos

Los sistemas de ficheros paralelos son un tipo de sistemas de ficheros de red capaces de realizar operaciones de entrada y salida en paralelo. Esto rompe con el esquema tradicional de sistema de ficheros, en el cual cada operación sobre el almacenamiento persistente no puede ejecutar a la vez que otra.

La arquitectura habitual de un sistema de ficheros paralelo consta de, como mínimo, dos roles dentro de la red:

- **Nodo de cómputo** o **cliente**: envía las peticiones para interactuar con el sistema de ficheros paralelo.

- **Nodo de E/S** o **servidor**: recibe las solicitudes para realizar operaciones de entrada y salida que corresponden a su espacio de almacenamiento.

Este conjunto de roles se puede ampliar, añadiendo, por ejemplo:

- **Nodo de nombres**: resuelve las peticiones del nodo de cómputo que solicitan los nombres de los nodos de E/S que involucran a una operación dada.

- **Balanceador de carga**: registra la carga de los nodos de E/S y establece un criterio de selección para evitar posibles cuellos de botella. Este rol puede resultar útil en sistemas que introducen el concepto de redundancia: ante varias copias de un bloque de datos, se evita el acceso constante al mismo nodo de E/S.

Todos los roles colaboran para ofrecer a los usuarios una visión global del sistema de ficheros, abstrayendo la distribución de los ratos en la red. Por otra parte, el diseño del sistema debe tener en cuenta la *tolerancia a fallos*, habituales en sistemas de red y la *consistencia de datos* en caso de que varios nodos de cómputo operen simultáneamente sobre un mismo dato.

El auge del *big data* ha dado lugar a la aparición de gran cantidad de alternativas en el ámbito de los sistemas de ficheros paralelos. Algunos ejemplos son los siguientes.

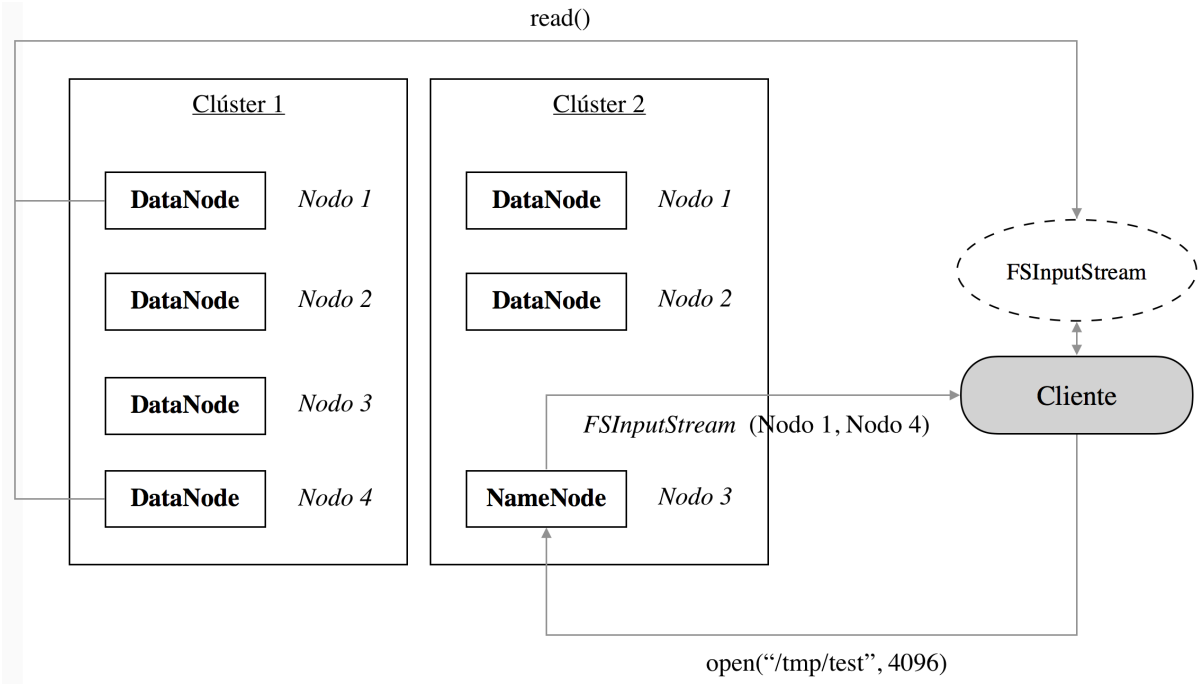
2.2.1. HDFS

El sistema de ficheros paralelo HDFS (Hadoop Distributed File System) está desarrollado por el proyecto Apache Hadoop. Está inspirado por el artículo de Google sobre su sistema de ficheros propietario, GFS, publicado en 2003[7].

Su código está escrito en el lenguaje de programación Java, con el objetivo de poder portarse a cualquier sistema operativo. Su diseño beneficia a los conjuntos grandes de datos, por lo que suele utilizarse en combinación con tamaños de bloque superiores al megabyte. Esto suele ser la norma en todos los sistemas de ficheros orientados a *big data*.

Sus características le proporcionan tolerancia ante fallos y consistencia, a cambio de renunciar a la semántica POSIX y relajar su modelo de datos[8]. Una restricción que impone, por ejemplo, es que solo se puede concatenar datos a un fichero una vez se ha escrito sobre él. Los datos previamente escritos sólo podrán ser leídos.

Los roles principales de los computadores de un sistema HDFS son los siguientes:



- **NameNode**: figura que agrupa funcionalidades del nodo de nombres y del balanceador de carga. Es un único servidor dentro de la red de nodos de HDFS que almacena el espacio de nombres del sistema de ficheros y los metadatos de los ficheros (por ejemplo, su factor de replicación o su tamaño de bloque). Si este nodo cae, es muy complicado recrear el sistema

de ficheros, ya que no se tiene información sobre la localización de los bloques de cada fichero, ni tampoco se conocen sus metadatos.

- **DataNode**: figura que equivale al nodo de E/S. HDFS no restringe la ejecución de varios DataNodes en un mismo nodo, aunque sus desarrolladores ven esta opción poco práctica en la mayoría de situaciones y aconsejan desplegar un único rol de este tipo por cada nodo. El flujo de datos en las solicitudes de escritura y lectura siempre se realiza a través de este rol. Para el resto de solicitudes, el cliente realiza la petición al NameNode y éste interactúa con los DataNode pertinentes para que se lleve a cabo.

La arquitectura de HDFS guarda una fuerte relación con MapReduce y el marco de trabajo de Hadoop, no siendo habitual su uso independiente en otros entornos.

2.2.2. GPFS

El sistema de ficheros paralelo GPFS (General Parallel FileSystem), cuyo nombre comercial en la actualidad es IBM Spectrum Scale, está desarrollado por IBM.

Este sistema se caracteriza por cumplir con la semántica POSIX para operaciones con ficheros y por repartir sus metadatos entre el conjunto de servidores que lo componen. Estas cuestiones son posibles gracias a la implementación de un sistema de bloqueo distribuido[9], que es capaz de aplicarse sobre segmentos de datos concretos para garantizar la consistencia de datos en caso de simultaneidad de peticiones sobre él.

Está pensado para su ejecución en hardware común, por lo que prevé la existencia de fallos. Para implementar tolerancia a fallos, utiliza diversas políticas, como RAID1, para la dispersión de las réplicas de los bloques.

El paradigma de roles que sigue es sencillo, disponiendo únicamente de nodos de cómputo y de nodos de E/S. Toda la información del sistema de ficheros, incluidos el mecanismo de bloqueo, se encuentra distribuida entre todos los nodos de E/S.

Este sistema es muy maduro, remontándose a 1998. Dispone de implementaciones para AIX, GNU/Linux y Microsoft Windows Server. Se permite la formación de clústers heterogéneos con cualquiera de estos sistemas. Por último, su lenguaje de programación es C.

2.2.3. MapR FS

El sistema de ficheros paralelo MapR FS está desarrollado por MapR Technologies y fue liberado al público en 2011. El objetivo detrás de su creación es extender las capacidades de Apache Hadoop ofreciendo una plataforma más estable y de mejor rendimiento que HDFS.

Soporta semántica POSIX para la manipulación de los ficheros y se basa en un modelo de consistencia fuerte. Para la interacción con el sistema de ficheros, se puede utilizar la interfaz de sistemas de ficheros de Hadoop, un cliente NFS o la interfaz FUSE.

Al igual que GPFS, los metadatos no se encuentran concentrados en un nodo de nombres, sino que se encuentran distribuidos entre los nodos de E/S.

Es un proyecto muy reciente que, de momento, sólo soporta sistemas operativos GNU/Linux para el despliegue. Los clientes sí pueden gozar de cierta heterogeneidad dada la compatibilidad con NFS.

2.2.4. Lustre

Lustre es un sistema de ficheros paralelo cuyo nombre proviene de juntar las palabras Linux y Cluster. La compañía detrás de su desarrollo es Cluster File Systems, que la ofrece bajo una

licencia GPLv2. Por este motivo, su uso está muy extendido en el TOP500 de supercomputadores del mundo[10].

En cuanto a su arquitectura, es más parecida a la de HDFS que a la de GPFS y MapR FS. Como característica diferenciadora frente a los demás, es un sistema de ficheros orientado a objetos, lo cual significa que los ficheros albergan estructuras complejas. Lustre dispone de los siguientes roles para los nodos en su red[11]:

- **Metadata Server**: figura que agrupa el rol de nodo de nombres y el de balanceador de carga. Los clientes se conectan a este servidor, con dirección conocida, para consultar los nodos con los que tienen que comunicarse para efectuar una transacción con el sistema. Con el fin de proporcionar tolerancia a fallos con un conjunto tan delicado de información, los metadatos se encuentran replicados. Al contrario que lo sucedido con el NameNode en HDFS, pueden existir varios Metadata Server dentro del sistema, estando la información distribuida entre ellos.

- **Object Storage Target**: figura que se corresponde con el rol de nodo de E/S. Este tipo de nodos se encarga de almacenar los objetos del sistema y responder a las solicitudes de los clientes para operar con ellos.

Al igual que GPFS, se caracteriza por un sistema de bloqueo distribuido que le permite mantener una semántica POSIX. De este modo, garantiza un modelo de consistencia fuerte.

Lustre soporta GNU/Linux como plataforma de despliegue del sistema de ficheros paralelo y está escrito en el lenguaje de programación C.

2.2.5. *Expand*

El sistema de ficheros paralelo Expand, desarrollado en el Grupo de Arquitectura de Computadores, Comunicaciones y Sistemas (ARCOS) de la Universidad Carlos III de Madrid, está disponible para su uso bajo la licencia GPLv2.

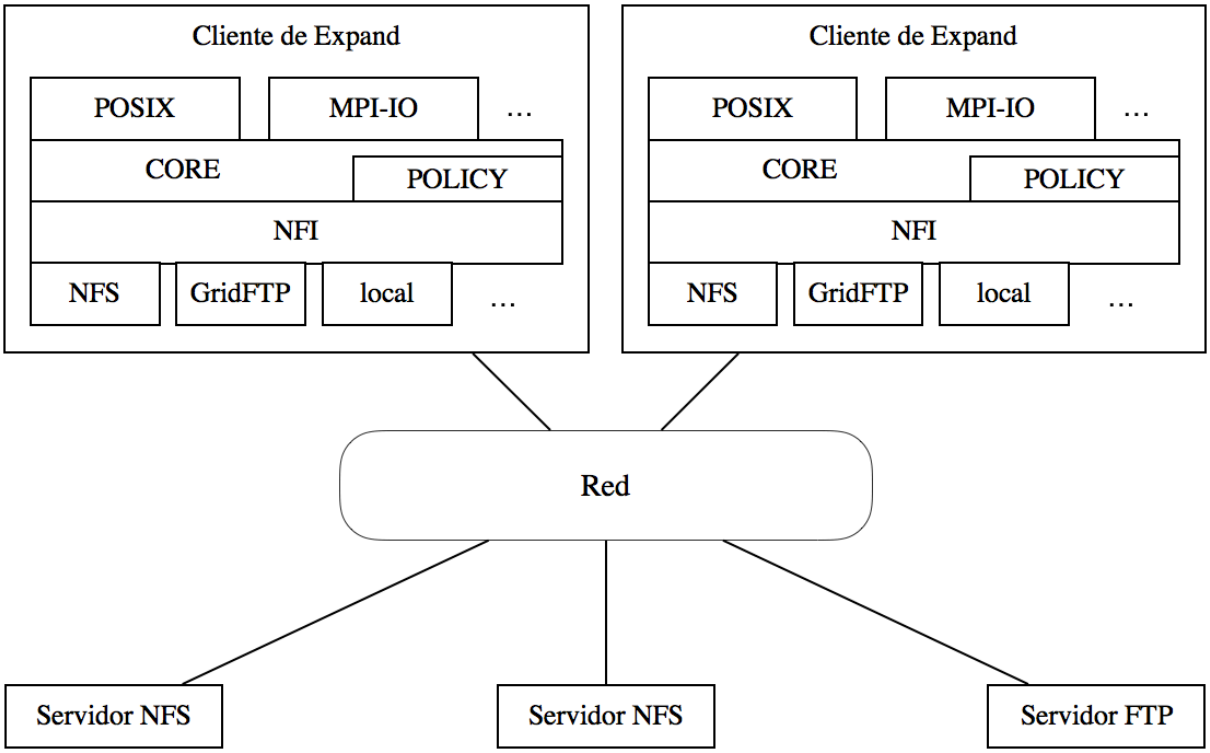
El sistema cuenta con una interfaz POSIX para la manipulación de los ficheros y su modelo de consistencia es débil en la versión que se encuentra en producción. Esto se traduce en que llamadas simultáneas de múltiples clientes no se bloquean, por lo que el estado del sistema de ficheros puede no ser consistente.

La arquitectura de Expand se basa en servidores de red neutros[12], donde se utilizan los dos roles clásicos, cliente y servidor. Los metadatos se reparten entre los servidores y permiten localizar los bloques en su nodo correspondiente.

Los servidores *neutros* pueden basarse en cualquier sistema de ficheros[13], siempre y cuando disponga de implementación dentro de Expand. Es posible, por ejemplo, un sistema de ficheros paralelo conformado por servidores NFS3 y HTTP WebDAV. Estos servidores no tienen constancia de que forman parte de Expand. En el ejemplo anterior, los nodos solo ejecutan un servidor NFS3 o un servidor HTTP WebDAV.

La topología de Expand, por lo tanto, depende del cliente y su configuración. El listado de servidores, protocolos y puntos de montaje se almacena en un fichero de configuración del cliente y es la única formalización de la red del sistema de ficheros paralelo.

El usuario de Expand, a la hora de interactuar con el sistema de ficheros, no tiene constancia de los servidores que forman parte de él. El cliente se encarga de, en base a las políticas que tiene definidas y el fichero de configuración, seleccionar las operaciones a efectuar, sobre qué servidores aplicarlas y usando qué protocolo.



2.3. Arquitectura de Hadoop

Apache Hadoop es un marco de trabajo orientado a procesos de análisis masivo de datos (simulaciones científicas, estudios estadísticos, etcétera) que dispone de todas las herramientas necesarias para el despliegue de la plataforma de datos, la gestión de los recursos para cada nodo del clúster y la programación de trabajos de análisis.

2.3.1. Componentes principales

Hadoop es un proyecto complejo, mantenido por la amplia comunidad de Apache. Su desarrollo se lleva a cabo a través de cuatro proyectos independientes y, en versiones

recientes, goza de modularidad. Ésta se consigue a través del proyecto principal, denominado Common.

La intención de su equipo de mantenimiento es que el proyecto Common sirva de material de unión entre todos los componentes de Hadoop y proporcione una interfaz bien definida de cara al exterior[14]. Los componentes de Hadoop son tres:

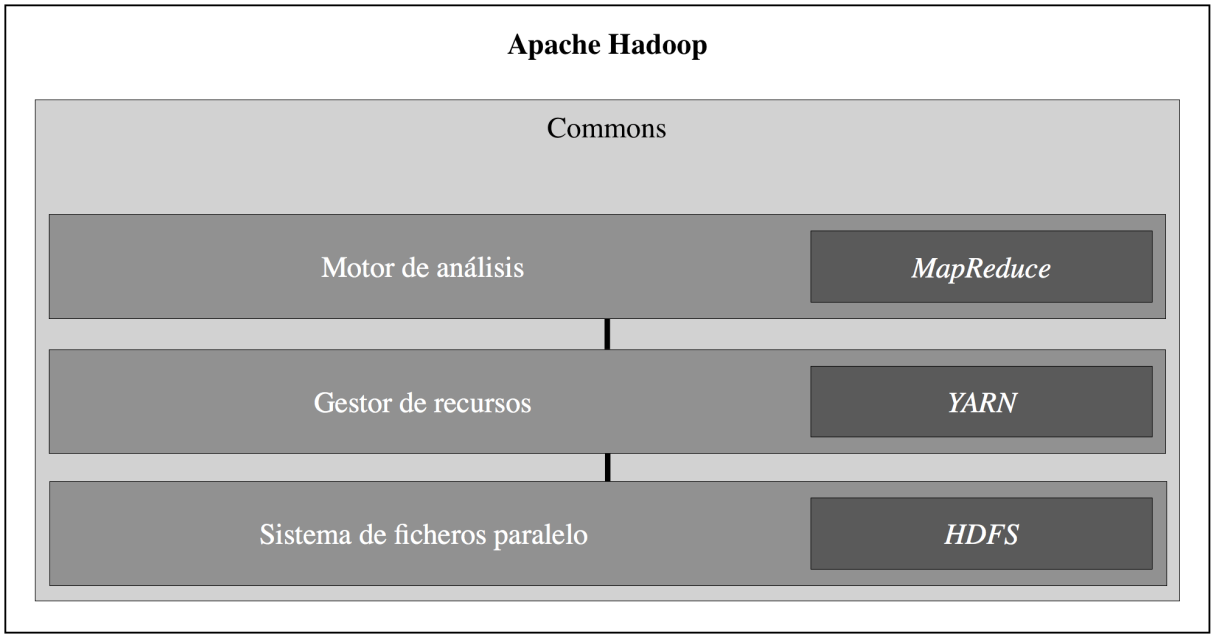
- **Motor de análisis**: formaliza el lenguaje de modelado de los problemas de análisis y los descompone en subproblemas que puedan resolverse en paralelo. Una vez alcanzada la solución de cada subproblema, es el encargado de componer la solución completa.

- **Gestor de recursos**: balancea la carga del clúster seleccionando a qué nodos llevar procesamiento siguiendo unas políticas determinadas.

- **Sistema de ficheros paralelo**: controla la ubicación de cada segmento de datos dentro del clúster y proporciona, mediante abstracción, una visión global de los datos que contiene.

El proyecto Common define las interfaces de estos tres componentes y se encarga de comunicarlos entre sí, permitiendo que cualquier implementación de los componentes que respete la interfaz pueda funcionar dentro de Hadoop.

Los otros tres proyectos que mantiene el equipo de desarrollo de Hadoop son MapReduce, YARN y HDFS. Se corresponden con la implementación de cada uno de los componentes mencionados anteriormente. Así, la versión de Hadoop proporcionada por Apache sigue el siguiente esquema:



Otra posible combinación de elementos sería la de Spark como motor de análisis, YARN como gestor de recursos y MapR FS como sistema de ficheros paralelo. Siempre y cuando los módulos respeten la interfaz, cualquier combinación es factible.

2.3.2. Modelos de integración para sistemas de ficheros

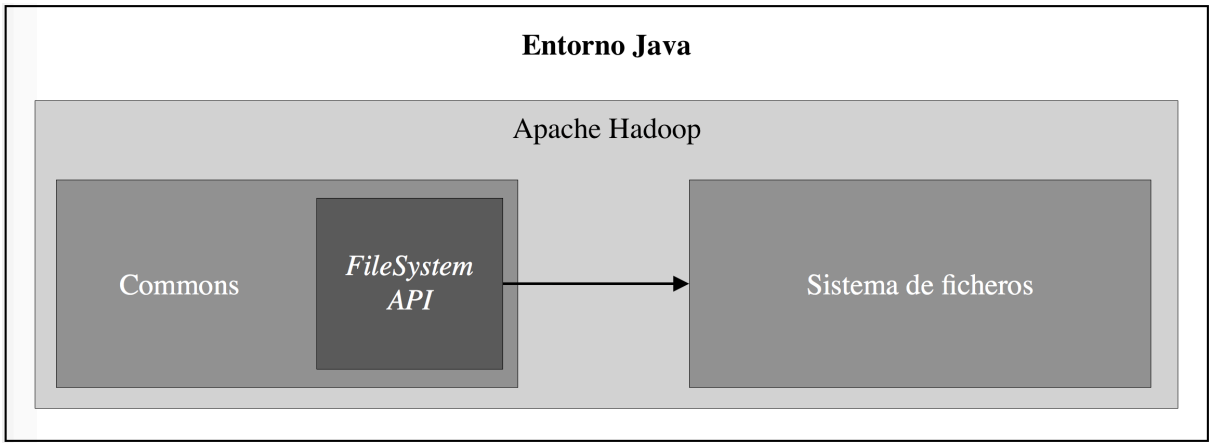
Los desarrolladores de sistemas de ficheros utilizan distintos enfoques para integrarlos dentro de Hadoop. Las implementaciones varían en función del tipo de sistema, el lenguaje en el que está escrito y la interfaz de la que dispone para la interacción.

En función del enfoque, varían el rendimiento y la forma de mantener la integración. Se ha estudiado la implementación para los siguientes sistemas de ficheros paralelos: HDFS, MapR FS, Google Cloud Storage y GPFS.

2.3.2.1. Enfoque específico

Esta primera filosofía se basa en proporcionar una biblioteca Java que implementa un sistema de ficheros siguiendo la interfaz de Hadoop. Esto da lugar a un diseño específico del sistema para satisfacer sus requerimientos.

Esta manera de afrontar el problema es muy exclusivista, generando un código vinculado estrechamente al entorno. Los sistemas de ficheros que siguen esta estrategia suelen incorporar un paquete independiente de utilidades para poder interactuar con el sistema por otros medios. Si no lo proporcionan, existe una dependencia total de Hadoop para el manejo de los ficheros del sistema. Esta forma de integración se encuentra en HDFS.



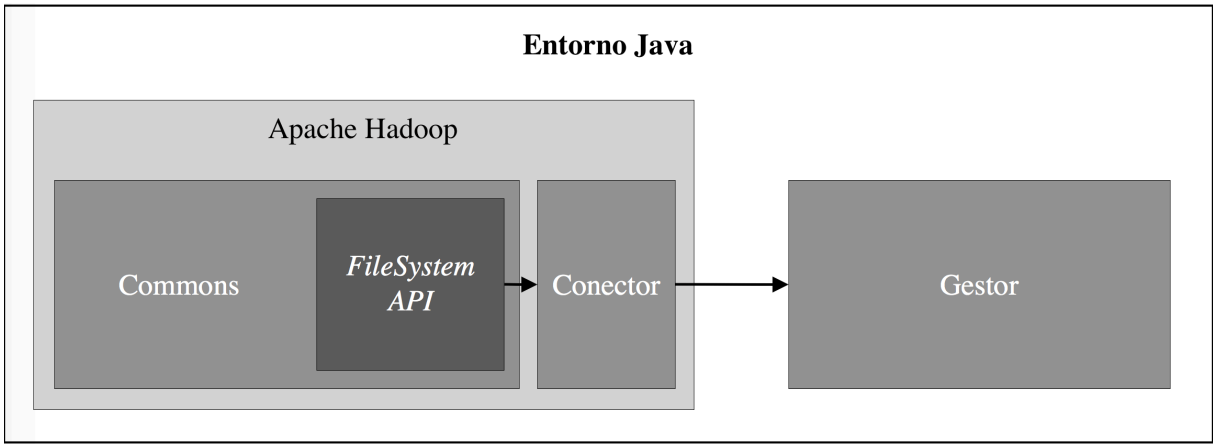
Puede ser una manera efectiva de garantizar la compatibilidad del sistema de ficheros con Hadoop: si está diseñado específicamente para él, es menos probable que aparezcan fallos debidos a discordancias entre la forma en que el sistema está diseñado y la manera en que Hadoop espera que funcione.

Sin embargo, esto puede ser un problema: si los desarrolladores de Hadoop toman una decisión importante respecto al funcionamiento de su sistema de ficheros, es posible que se requiera un rediseño completo de los sistemas basados en esta filosofía.

2.3.2.2. Puente con biblioteca Java

La segunda filosofía se basa en proporcionar una biblioteca Java que implementa la interfaz para sistemas de ficheros de Hadoop pero que no interactúa directamente con el sistema. Dicha labor queda relegada a una segunda biblioteca, también en el lenguaje Java. Por comodidad, se considerará a la primera biblioteca como *conector* y a la segunda como *gestor*.

El funcionamiento es el siguiente: Hadoop manda la solicitud sobre el sistema de ficheros al conector. Después, el conector realiza la traducción de la operación solicitada a una o más solicitudes que sean comprendidas por el gestor, que puede tener otra interfaz completamente distinta para operar con los ficheros. Finalmente, el gestor devuelve al conector una respuesta y éste la traduce para que Hadoop la comprenda.



Esta solución busca independizar de Hadoop el desarrollo del sistema de ficheros. En el caso de aquellos sistemas que ya dispongan de biblioteca Java para interactuar con él, supone el desarrollo únicamente del conector.

Por otra parte, un cambio en la interfaz de sistemas de ficheros de Hadoop solo supone cambios en el conector, no requiere un rediseño del sistema de ficheros.

Esta es la estrategia seguida por MapR FS, el sistema de ficheros local y el protocolo de transferencia de ficheros. Google Cloud Storage tiene una filosofía similar, pero con una excepción: las operaciones sobre el sistema de ficheros se efectúan enviando los datos por red, ya que el sistema de ficheros reside en la nube.

2.3.2.3. Puente con biblioteca C

Esta tercera y última filosofía es equivalente en concepto a la anterior, solo que se requieren cambios en la manera de afrontar el conector y el gestor.

Existen sistemas de ficheros que no disponen de interfaz en lenguaje Java para la interacción. El trabajo de crear una nueva interfaz en Java requiere muchos recursos y puede introducir una sobrecarga que los desarrolladores no están dispuestos a asumir. En este caso, la integración con Hadoop se vuelve más complicada.

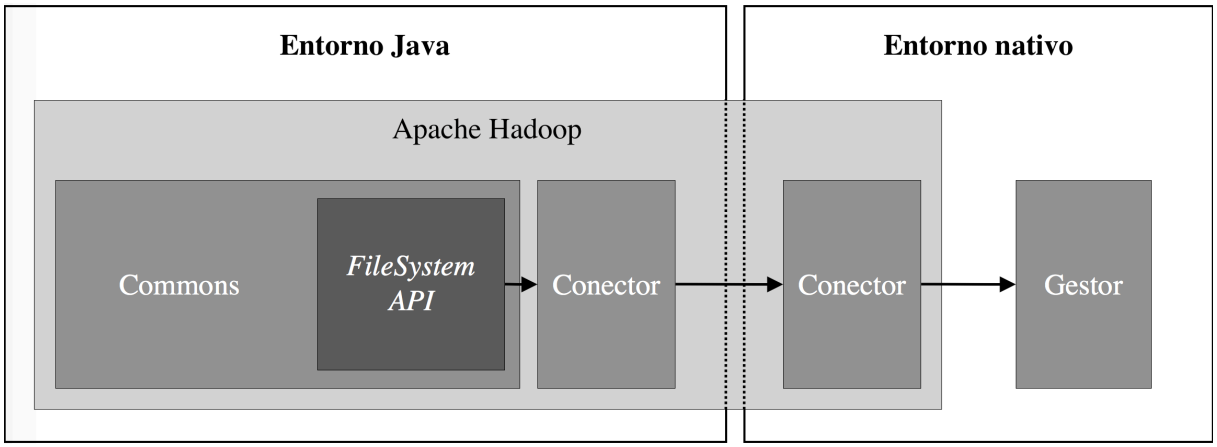
Se parte de la solución anterior. En ella, el conector podía interactuar directamente con el gestor ya que ambos conviven en el entorno Java. Si el gestor cambia de lenguaje, entonces el conector necesitará dos procesos de traducción:

- Traducir las estructuras de datos y las operaciones de Java para que puedan ser comprendidas por el gestor en C y viceversa.

- Traducir las estructuras de datos y las operaciones de Hadoop para que sigan la interfaz definida por el gestor y viceversa.

En este contexto, el conector tiene dos segmentos separados, ambos residentes dentro de Hadoop. El primero es una biblioteca Java y el segundo es una biblioteca C. Para su comunicación, se recurre a la interfaz *JNI* proporcionada por Java, que se encarga del primer proceso de traducción entre las dos partes del conector. El segundo proceso de traducción

tiene lugar en el segmento C del conector, que es capaz de comunicarse con el gestor ya que están en el mismo lenguaje.



Esta filosofía tiene ventajas y desventajas. Por un lado, agiliza las operaciones al realizarse sin la sobrecarga de la máquina virtual de Java y permite integrar sistemas cuya interfaz está escrita en lenguaje C. Sin embargo, la mayor complejidad del código del conector puede dar lugar a errores y puede dificultar las labores de depuración. Por último, el mantenimiento puede ser más engorroso en caso de que aparezcan cambios en la interfaz para sistemas de ficheros de Hadoop.

Esta alternativa se utiliza para integrar GPFS en Hadoop.

Capítulo 3

Análisis de la solución

Este capítulo analiza el problema que da origen a este proyecto y las distintas maneras de solucionarlo. En el primer apartado, se explica en qué consiste el problema. En el segundo apartado, se exponen las implicaciones que tiene sobre su entorno. En el tercer apartado, se elabora una lista de posibles soluciones que se podrían adoptar para paliarlo o solucionarlo. Por último, en el cuarto apartado, se elige una solución argumentando los motivos de la decisión.

3.1. Fundamento del problema

Al llevar a cabo cierta cantidad de proyectos relacionados con *big data* en el marco de trabajo Apache Hadoop, se descubre un proceso tedioso y largo de migración del conjunto de datos cada vez que se necesita analizar éste y no se encuentra ya en Hadoop. Esto se debe a que cuenta con un sistema de ficheros paralelo propio, llamado HDFS, que es estricto en cuanto a la importación y exportación de datos. En conjuntos de datos pequeños, este proceso puede ser tolerable, pero deriva en largos tiempos de espera cuando el conjunto es grande. Además, se puede considerar un proceso inútil si estos datos ya se encuentran distribuidos en un sistema de ficheros distinto.

Es por esto que los desarrolladores de algunos sistemas de ficheros paralelos muy extendidos hoy día han aportado distintas soluciones para que puedan ser utilizados directamente desde Hadoop, sustituyendo a HDFS o conviviendo con él. Algunos ejemplos son S3 de Amazon, GPFS de IBM, MapR FS de Apache o Cloud Storage, de Google. Pero esa no es la única razón: HDFS es bastante rígido en cuanto a la manera de tratar las operaciones concurrentes[8]

y la modificación de los datos ya existentes, por lo que algunas de las alternativas buscan solventar estas limitaciones.

La inclusión de un nuevo sistema de ficheros en Hadoop no es una tarea sencilla. La mayoría de sistemas de ficheros confían en una biblioteca en el lenguaje C de programación para la interacción con él. Sin embargo, la interfaz para implementar sistemas de ficheros en Hadoop está escrita en el lenguaje Java y está específicamente diseñada para él, no sigue ningún estándar conocido como POSIX. Esto dificulta las tareas de implementación y obliga a entender el funcionamiento de Hadoop para poder introducir en él un nuevo sistema de ficheros.

Éste es, en definitiva, el problema encontrado y que fomenta la realización de este proyecto: la dificultad de introducir sistemas de ficheros nuevos en Hadoop cuando éstos no cuentan con interfaz en el lenguaje Java ni tienen una interfaz semejante.

3.2. Implicaciones del problema

Dado que cualquier sistema de ficheros compatible con Hadoop tiene que seguir su interfaz y debe interactuar con él mediante el lenguaje de programación Java, se debe adaptar cualquier sistema existente para que cumpla estas exigencias, ya que no es habitual encontrar sistemas de ficheros paralelos de propósito general que se basen en dicha interfaz.

Además, la gestión de ficheros suele ser una cuestión de muy bajo nivel de abstracción, por lo que tampoco es habitual que se realicen implementaciones en el lenguaje Java, de muy alto nivel y cuyo código se restringe al dominio de una máquina hipotética representada a través de la *JVM*.

Este hecho puedo colocar una barrera para la inclusión de nuevos sistemas de ficheros paralelos en Hadoop, ya sea por falta de recursos o de interés dada la problemática que supone la implementación.

Por otro lado, como se mencionó anteriormente, los usuarios de sistemas de ficheros para los que no existe una adaptación para Hadoop se acaban viendo obligados a realizar un proceso de migración para poder explotarlos mediante este software. Esta situación se agrava si en su flujo de trabajo necesitan cambiar el conjunto de datos habitualmente y si necesitan que la solución resida en otro sistema de ficheros.

3.3. Posibles soluciones al problema

Existen varias aproximaciones a la solución para este problema. Habiéndose estudiado en el punto anterior las diferentes estrategias de implementación que utilizan los desarrolladores para incorporar sistemas de ficheros en Hadoop, se pueden estimar distintas formas de facilitar la inclusión de nuevos sistemas.

3.3.1. Opción 1: Integrar sistemas de ficheros populares

La primera solución consiste en elaborar una lista de sistemas de ficheros populares en el ámbito del *big data* e integrarlos de la manera más cómoda posible en Hadoop.

Ventajas	A corto plazo, soluciona los problemas de migración para la mayoría de usuarios de Hadoop. Al realizar una integración individual para cada sistema, se reducen las probabilidades de que surjan incompatibilidades.
Desventajas	A corto plazo, un grupo minoritario de usuarios sigue sin ver solucionado el problema. A largo plazo, la popularidad de los sistemas de ficheros puede variar y ser necesario buscar nuevas soluciones. Requiere conocimientos de cada sistema de ficheros individual para poder realizarse. No ofrece nada distinto frente a las prácticas realizadas hasta el momento.

3.3.2. Opción 2: Sistema de ficheros paralelo integrador

Esta solución consiste en integrar en Hadoop un sistema de ficheros paralelo que recurra internamente a otros sistemas de ficheros paralelos para obtener los datos.

Ventajas	El esfuerzo de integrar nuevos sistemas pasa a recaer sobre este sistema de ficheros, que puede facilitar el proceso. Permite compaginar varias fuentes de datos con transparencia sobre el sistema que utilicen.
Desventajas	La complejidad de un sistema de ficheros de estas características puede ser muy alta y requerir muchos recursos para su mantenimiento y desarrollo. El problema sigue existiendo aunque haya cambiado de lugar y no hay ninguna garantía de que esta solución lo atenúe o elimine.

3.3.3. Opción 3: Reemplazar la interfaz de Hadoop

Dado que la dificultad para integrar nuevos sistemas de ficheros en Hadoop viene dada por una interfaz específica y un lenguaje de muy alto nivel, la siguiente opción posible es introducir un mecanismo de suplantación de dicha interfaz. Esto es, un componente que presente una interfaz estándar (POSIX) en un lenguaje de bajo nivel (C) y que se encargue de hacer las transformaciones necesarias para que las operaciones y la información sean comprensibles para Hadoop.

Ventajas	Se produce una separación de disciplinas: expertos en Hadoop adaptan la interfaz a POSIX y C mientras que los desarrolladores de un sistema de ficheros se pueden dedicar a integrarlo con POSIX. Reduce la dificultad de integrar nuevos sistemas ya que la interfaz presentada es popular, está bien documentada y es simple. Es robusta de cara al futuro ya que no restringe el diseño ni las capacidades del sistema de ficheros que utilice el mecanismo.
Desventajas	Sigue existiendo dependencia respecto a los desarrolladores: si no quieren invertir recursos, seguirán sin integrar sistemas, aunque sea menos costoso.

3.4. Selección de solución

Analizando el entorno en el que se enmarca el problema, se puede apreciar que Hadoop se mueve en un área que se ha popularizado en la última década, por lo que es previsible que la poca madurez del sector de lugar a cambios importantes en periodos cortos de tiempo. Por lo tanto, se debe seleccionar la solución que mejor pueda soportar los cambios de paradigmas y la aparición de soluciones nuevas.

Teniendo esto en cuenta, la opción 1 queda descartada: por más que se implementen sistemas de ficheros populares para que estén disponibles para los usuarios de Hadoop, la popularidad no es un factor estable. Respecto a las opciones 2 y 3, siguen siendo válidas, ya que ambas tienen capacidad de adaptación frente a los cambios con un empleo leve de recursos.

Por otra parte, las empresas están reclamando soluciones baratas y rápidas para sus sistemas. Aunque cualquiera de estas soluciones tiene un coste, puede verse compensado por los ahorros en migración y la reducción del tiempo total empleado en los trabajos. En este contexto, es posible que el precio de la solución desarrollada no sea un factor determinante para la compra siempre y cuando produzca resultados, pero ante dos alternativas funcionales es muy posible que las empresas se decanten por la opción más económica.

En este contexto, la opción 2 es demasiado compleja y requerirá de gran cantidad de recursos para llevar a cabo su desarrollo.

Aunque es posible que los sistemas de ficheros paralelos acaben programándose en lenguajes de alto nivel y la interfaz POSIX acabe relegada a un segundo plano en favor de un nuevo estándar, las cifras anticipan que esta situación no va a darse a corto plazo. En el caso de que se diese a largo plazo, no hay certezas sobre cómo de similar será la interfaz a la utilizada por Hadoop, ni si ésta habrá cambiado también, por lo que no parece ser un factor determinante por el momento.

Es por todo esto que la opción 3 es la seleccionada para ser desarrollada frente a las alternativas:

- Es robusta frente al futuro.

- Es más económica que la segunda alternativa de solución.

- Los estándares que utiliza podrían cambiar a largo plazo, pero este cambio afectaría a cualquier solución admisible.

A partir de este momento, al elemento que "suplanta la interfaz" de Hadoop se le pasará a llamar "conector", ya que su encaje dentro de Hadoop es el mismo que dicho componente cuando se le mencionó al realizar el estudio de mecanismos de integración existentes en el capítulo 2.

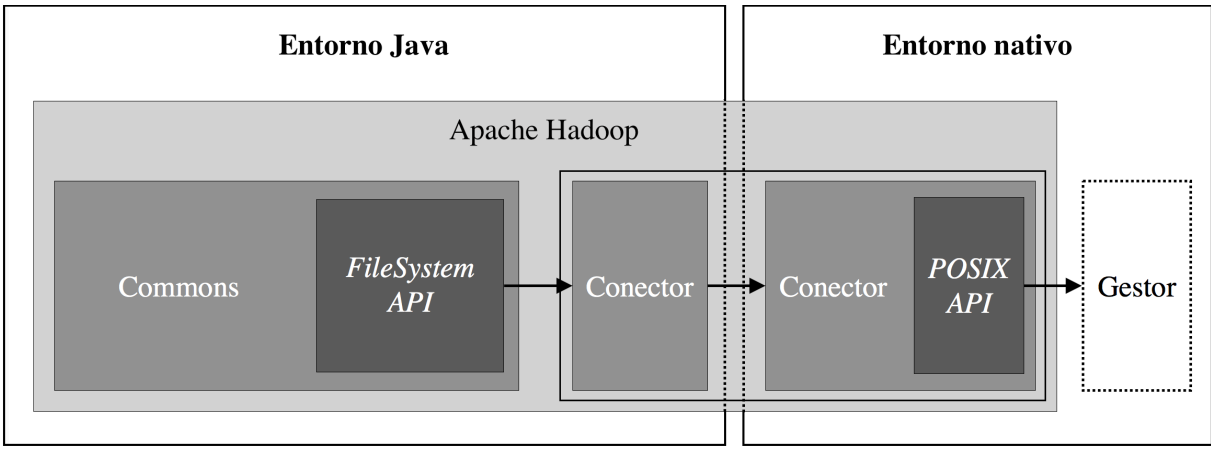
Capítulo 4

Diseño e implementación

En este apartado se detalla el diseño de la solución y la manera en que se ha implementado, justificando todas las decisiones y describiendo las consecuencias de ciertos aspectos del diseño. Por último, se define la prueba de concepto para esta implementación.

4.1. Diseño de la solución

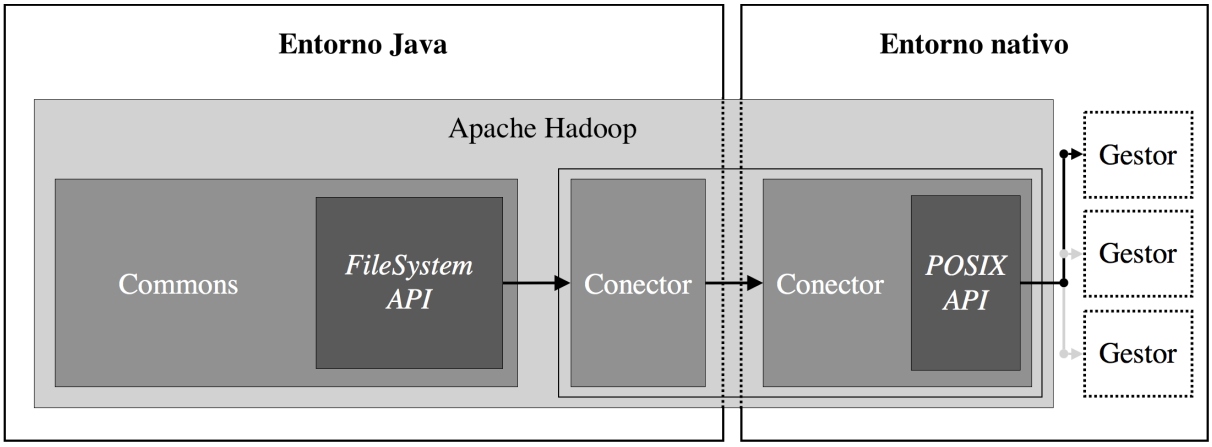
El conector actúa como capa de abstracción entre la interfaz de sistemas de ficheros proporcionada por Hadoop y cualquier sistema que quiera integrarse en él.



Una de las ventajas de Hadoop es la interoperabilidad y convivencia que se permite entre sus sistemas de ficheros. Con esto en mente, se tiene que tomar una decisión importante de diseño sobre la manera en que se integran los gestores, a elegir entre tres posibilidades.

4.1.1. Conector único ajustable

Esta opción consiste en incluir un único conector en Hadoop y que, a través de un parámetro de configuración, el usuario pueda elegir qué gestor es el que utiliza internamente.

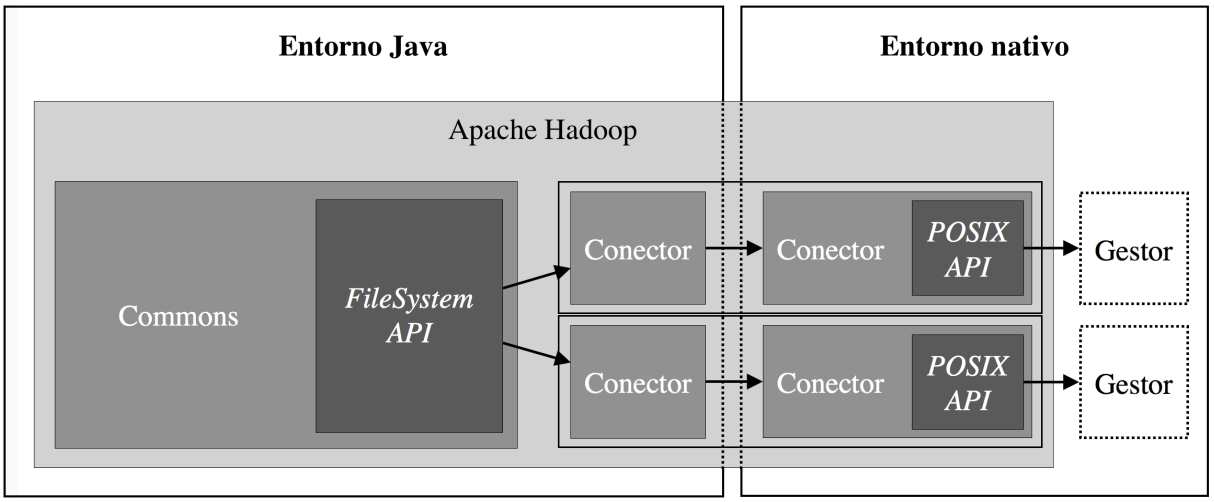


Introducir este diseño supone un ahorro en almacenamiento y memoria ya que sólo se requiere una instancia del conector por cada sistema de ficheros a integrar. Además, el conector puede tener un espacio de nombres propio para el que no se requiere ninguna alteración; no tiene que convivir con otro conector dentro de la misma instalación de Hadoop cuyo espacio de nombres pueda colisionar.

Sin embargo, tiene la desventaja de que rompe con la interoperabilidad: aunque el conector tenga acceso a varios gestores, para Hadoop sólo es visible un sistema de ficheros, representado por el conector en sí. Esto impide que se pueda interactuar entre los distintos sistemas de ficheros.

4.1.2. *Múltiples conectores no ajustables*

Esta opción consiste en que cada sistema de ficheros a integrar en Hadoop utilice un conector independiente, siendo una entidad separada de las demás existentes.



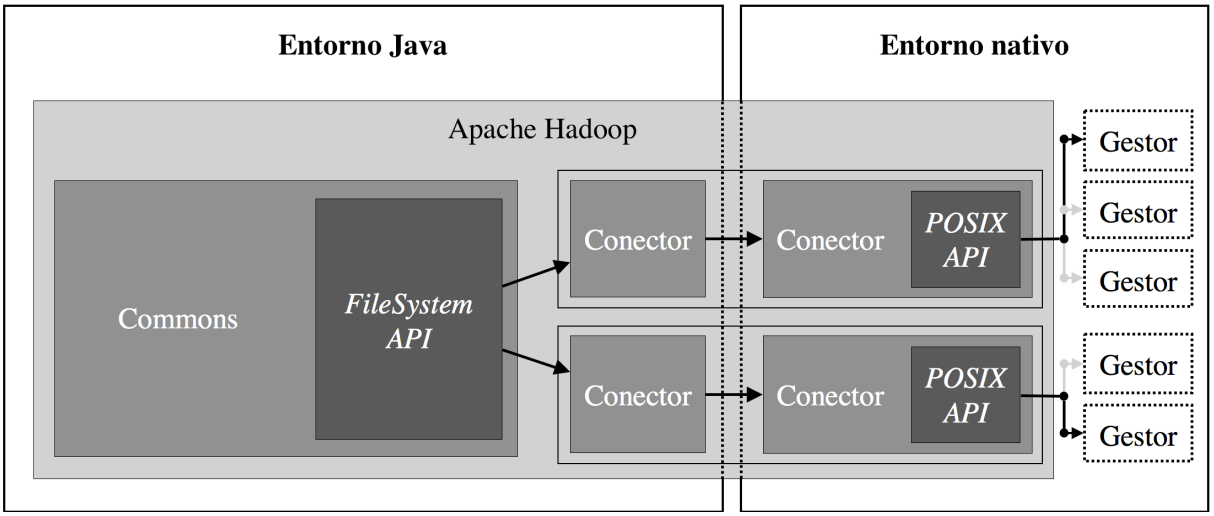
La ventaja de este diseño es que garantiza la interoperabilidad entre los distintos sistemas de ficheros que se integren. Además, pueden coexistir y utilizarse indistintamente cambiando la *URI* de las rutas, por lo que a nivel práctico el usuario no percibe diferencias en funcionalidad respecto a la opción anterior.

Sin embargo, tiene varios problemas. El primero es el conflicto del espacio de nombres. No se puede utilizar exactamente el mismo conector cada vez que se quiera integrar un nuevo sistema de ficheros ya que los nombres colisionarían con el resto de conectores ya instalados. Por tanto, el programador tiene una labor extra: personalizar el espacio de nombres para que sea único para su sistema.

El segundo problema es el de memoria y disco. El código del conector estaría replicado por cada sistema de ficheros. No obstante, el código del conector no supone una sobrecarga tan grande como para que esta cuestión se convierta en un problema grave. Tampoco es habitual que los usuarios utilicen simultáneamente un abanico amplio de sistemas.

4.1.3. *Múltiples conectores ajustables*

Por último, se puede realizar un diseño que aproveche las características de los dos anteriores, es decir, que coexistan múltiples conectores y que cada uno de ellos pueda utilizar internamente un gestor determinado mediante configuración.



Esta alternativa, sin embargo, se puede interpretar como un caso específico de la segunda. Los conectores ajustables y no ajustables pueden convivir, pasando a ser una decisión del programador la alternativa que desea implementar.

No parece ser una opción interesante, ya que agrega complejidad de cara al usuario y no ofrece ventajas frente a la alternativa de los múltiples conectores no ajustables. En su lugar, recuperaría los problemas de la primera opción y los uniría a los problemas de la segunda opción.

De estas tres alternativas, se escoge la segunda, ya que garantiza interoperabilidad a cambio de forzar cambios en el espacio de nombres del conector y replicar código.

4.2. Implementación de la solución

Para la implementación del conector, se deben tener en cuenta varias cuestiones: la dependencia del conector de las clases de Hadoop que residen en Common, la necesidad de introducir JNI y la definición de la interfaz POSIX para la conexión con el sistema de ficheros.

4.2.1. Estructura del proyecto

Para automatizar el proceso de compilación, el proyecto utiliza Apache Maven. Este sistema proporciona varias ventajas. En primer lugar, es capaz de descargar e inyectar las dependencias necesarias de Common para la obtención de la biblioteca en lenguaje Java del conector. En segundo lugar, cuenta con un plugin denominado Native Maven Plugin que integra dentro del proceso la compilación de su biblioteca en lenguaje C[17]. Por último, proporciona flexibilidad en cuanto a la plataforma para la que se compila el código en C, por lo que no se restringen nuevas implementaciones para otros sistemas operativos en el futuro.

Para que el proyecto esté organizado, la compilación se realice por etapas y los errores se aprecien en su contexto, es importante que se separe en módulos. En este caso, se han estimado necesarios tres módulos principales, denominados:

- **hadoop-connector-fs**: es el módulo de arranque, donde se define la codificación de los ficheros de código, los plugins utilizados y los módulos a lanzar.

- **hadoop-connector-fs-java**: es el módulo que se encarga de la compilación de la biblioteca Java asociada al conector. Descarga el proyecto Common de Hadoop como dependencia. Como resultado, genera el fichero *hadoop-connector-fs-java.jar*.

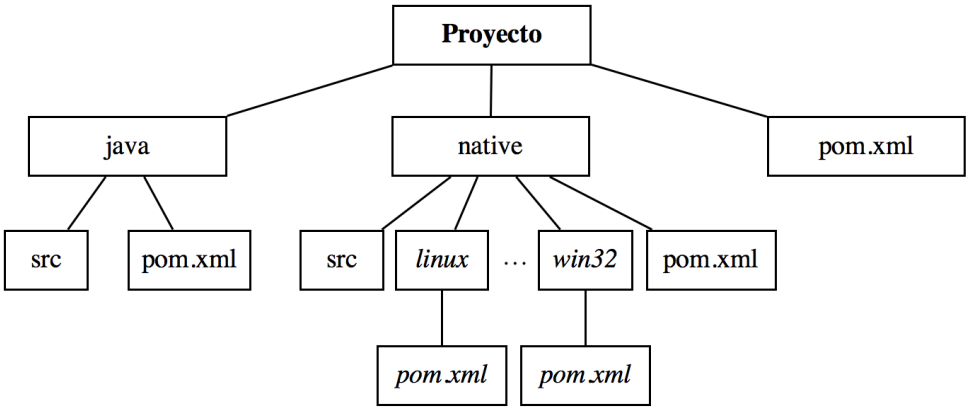
- **hadoop-connector-fs-native**: es el módulo de arranque de la sección C del conector. Descarga el proyecto Common de Hadoop como dependencia, ya que hace uso de sus clases traducidas. Además, define el módulo a lanzar para la compilación en función del sistema operativo definido como parámetro.

Como se anticipa en el último módulo, es necesario agregar, además de los principales, tantos módulos como sistemas operativos se quieran compatibilizar con el conector. Por ejemplo, para la compilación en GNU/Linux y Microsoft Windows:

- **libconnectorX**: este módulo compila la biblioteca C asociada al conector del sistema de ficheros X para el sistema operativo GNU/Linux. Como resultado, genera el fichero *libconnectorX.so*.

- **connectorX**: este módulo compila la biblioteca C asociada al conector del sistema de ficheros X para el sistema operativo Microsoft Windows. Como resultado, genera el fichero *connectorX.dll*.

Todas estas configuraciones vienen definidas en ficheros *pom.xml*. La estructura final de directorios es la siguiente, atendiendo a la organización de los módulos mencionada en la lista anterior:



Respecto a la ubicación de las bibliotecas generadas, éstas deben colocarse en los directorios convenientes para que Hadoop las detecte y puedan utilizarse. Para la biblioteca Java, el directorio es, partiendo desde la raíz de la instalación de Hadoop:

share/hadoop/common/lib

Respecto a la biblioteca en C, debe colocarse en otro directorio diferente junto a las demás bibliotecas nativas utilizadas por Hadoop. Este directorio es, partiendo desde la raíz de la instalación de Hadoop:

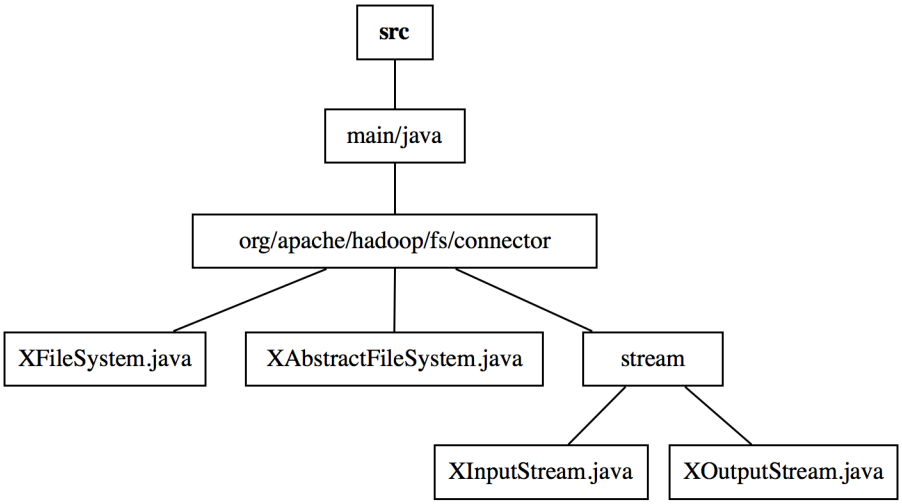
lib/native

La máquina virtual de Java buscará en este directorio las bibliotecas JNI. Estas bibliotecas se cargan en tiempo de ejecución y deben tener resueltas todas sus dependencias para poder funcionar. Es por esto que el módulo de Maven asociado a la compilación de la biblioteca C

del conector debe indicar tanto al compilador como al enlazador la ubicación de todas las bibliotecas e interfaces asociadas al gestor.

4.2.2. Biblioteca en lenguaje Java

El código fuente asociado a la biblioteca del conector en lenguaje Java sigue la siguiente estructura de directorios, partiendo del directorio *Proyecto/java/src* indicado anteriormente:



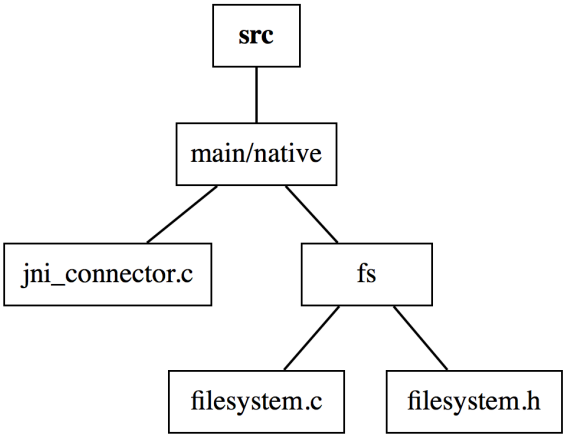
Todas estas clases son las que implementan el sistemas de ficheros de Hadoop. En concreto, XAbstractFileSystem y XFileSystem implementan las operaciones de manipulación de ficheros, mientras que XInputStream y XOutputStream implementan las operaciones de manipulación del flujo de datos de entrada y salida de los ficheros. Para todas estas clases, es necesario sustituir X por el identificador único del sistema de ficheros a conectar. Estos cambios son los que garantizan que no existan conflictos con el espacio de nombres.

La manera en que interactúa la biblioteca Java con la biblioteca C del conector es a través de funciones que utilizan la palabra clave *native*. En las clases de Java se ubica la definición de la función, que en un principio no está implementada.

En la función de inicialización del sistema de ficheros en XFileSystem, se realiza la llamada *System.loadLibrary()* para cargar en memoria la biblioteca C asociada al conector. Después de esta llamada, toda las funciones native que tengan una implementación correspondiente en C podrán usarse y JNI realizará los procesos de traducción procedentes.

4.2.3. Biblioteca en lenguaje C

El código fuente asociado a la biblioteca del conector en lenguaje C sigue la siguiente estructura de directorios, partiendo del directorio *Proyecto/native/src* indicado anteriormente:



La conexión con Java se realiza mediante el código en *jni_connector.c*. En él se implementan las funciones con palabra clave native dentro de las clases Java del conector. La correspondencia entre nombres de funciones, objetos y tipos de dato básicos se puede realizar de distintos modos. El ejecutable *javah* proporcionado en el JDK puede ser útil, ya que busca funciones con dicha palabra clave en un proyecto Java y devuelve una interfaz C lista para su implementación[16]. Sin embargo, no es de uso obligado y las equivalencias se pueden realizar a mano.

Por ejemplo, una función en Java con esta definición, perteneciente a la clase XInputStream:

```
private native synchronized int readBytes(byte b[], int off, int len) throws IOException;
```

Se transforma al lenguaje C, de manera que pueda ser aceptado por la interfaz JNI de Java, quedando del siguiente modo:

```
JNIEXPORT jint JNICALL
Java_org_apache_hadoop_fs_connector_stream_XInputStream_readBytes(JNIEnv *env,
jobject obj, jbyteArray jbuffer, jint off, jint len)
```

Los detalles más importantes de esta cuestión, sin entrar a explicar la mecánica completa de JNI, es que el nombre de la función debe contener el nombre completo del paquete Java y recibir necesariamente los argumentos **env* y *obj*, además de las equivalencias JNI de los argumentos reales de la función. Ésta es la manera en que se resuelve el conflicto para comunicar un lenguaje procedural y estructurado como C con un lenguaje también procedural, pero orientado a objetos, como es Java.

El puntero *env* sirve para interactuar con la máquina virtual de Java, pudiéndose crear objetos, manipularlos o realizar cualquier tipo de operación donde se vea necesaria su mediación. Por otro lado, *obj* representa al objeto que realiza la invocación de la función. La importancia de este último radica en que los objetos Java pueden mantener un estado individual, mientras que las llamadas a C mantienen un estado global compartido, independiente del objeto que las invoque. Es por esto que cualquier valor que sea propio de un objeto debe establecerse a través de este argumento y no mantenerse como variable global en C.

La implementación de cada función utiliza para acceder al sistema de ficheros la interfaz definida en *filesystem.h*, cuya implementación se realiza en *filesystem.c*. Esta interfaz es equivalente a nivel semántico a POSIX salvo algunas excepciones, que contemplan

operaciones sólo presentes en los sistemas de ficheros paralelos y no disponibles en el estándar.

En la implementación (filesystem.c) es donde el desarrollador que desea integrar un nuevo sistema de ficheros para Hadoop debe colocar el código que interactúe con el gestor. El resto del código no requiere modificaciones salvo aquellas necesarias para resolver el conflicto del espacio de nombres.

Las funciones con equivalencia semántica a POSIX son las siguientes:

Retorno	***Nombre***	***Argumentos***		
int	*fs_init*			
int	*fs_destroy*			
DIR*	*fs_opendir*	const char* path		
struct dirent*	*fs_readdir*	DIR* dirp		
int	*fs_closedir*	DIR* dirp		
int	*fs_mkdir*	const char* path	mode_t mode	
int	*fs_rmdir*	const char* path		
int	*fs_open*	const char* path	int oflag	…
int	*fs_close*	int fildes		
int	*fs_unlink*	const char* path		
ssize_t	*fs_read*	int fildes	void* buf	size_t nbyte
ssize_t	*fs_write*	int fildes	const void* buf	size_t nbyte
int	*fs_stat*	const char* path	struct stat* buf	
off_t	*fs_lseek*	int fildes	off_t offset	int whence
int	*fs_rename*	const char* src	const char* dst	
int	*fs_chmod*	const char* path	mode_t permission	
int	*fs_chown*	const char* path	uid_t uid	gid_t gid

Por otro lado, las funciones que no siguen esta semántica, pero que son necesarias para la integración con Hadoop, son las siguientes:

Retorno	***Nombre***	***Argumentos***		
int	*fs_translate*	const char* authority	const char* path	char* fspath
int	*fs_replication*	const char* path		
int	*fs_locate*	const char* path	char*** urls	

Dado que las funciones no siguen el estándar, se detallan a continuación el funcionamiento esperado de cada una de ellas.

Definición de fs_translate		
Descripción	Transforma la autoridad y la ruta solicitados por Hadoop en la ruta aceptada por el sistema de ficheros en sus operaciones.	
Argumentos	authority	Cadena de caracteres que representa la autoridad solicitada.
	path	Cadena de caracteres que representa la ruta solicitada.
	fspath	Cadena de caracteres donde escribir la ruta aceptada.
Retorno	Devuelve 0 en caso de éxito y -1 en caso de error.	

Definición de fs_replication		
Descripción	Obtiene el factor de replicación dada una ruta del sistema de ficheros.	
Argumentos	path	Cadena de caracteres que representa la ruta.
Retorno	Devuelve el factor de replicación en caso de éxito y -1 en caso de error.	

Definición de fs_locate		
Descripción	Obtiene los nombres de host que contienen cada una de las réplicas de cada bloque de un fichero.	
Argumentos	path	Cadena de caracteres que representa la ruta.
	urls	Puntero triple donde escribir el resultado, representando el primer nivel cada uno de los bloques, el segundo cada una de sus réplicas y el tercero el nombre del host que la almacena.
Retorno	Devuelve 0 en caso de éxito y -1 en caso de error.	

Aunque no se requieren más funciones para que el conector funcione correctamente, los desarrolladores pueden agregar tantas funciones auxiliares o código de apoyo como les resulte necesario, siempre y cuando preparen el sistema de compilación para que pueda aceptarlo si realizan alguna modificación muy particular.

4.3. Prueba de concepto

Para proporcionar una implementación sobre la que efectuar pruebas y que demuestre el funcionamiento del conector, se utiliza el sistema de ficheros paralelo Expand, desarrollado en el Grupo de Arquitectura de Computadores, Comunicaciones y Sistemas (ARCOS) de la Universidad Carlos III de Madrid.

4.3.1. Despliegue de instalación independiente

La biblioteca de Expand se obtiene utilizando las herramientas *automake*, *autoconf* y *libtool* sobre el código fuente del proyecto. Tiene, además, una dependencia con *mxml* para la interpretación del fichero de configuración que define su topología.

Estas herramientas no están disponibles en todos los servidores ni tampoco debería forzarse su instalación para todos los usuarios del sistema. Es por este motivo que, para la prueba de concepto, se ha preparado un proceso de despliegue de Hadoop, el conector y la biblioteca de Expand para GNU/Linux que no requiere permisos de administrador y cuyas dependencias vienen incluidas en la mayoría de distribuciones.

Para que este proceso funcione, es necesario empaquetar en el instalador el código fuente y/o los ejecutables de las dependencias. En el momento de instalación, todo el código fuente se compila y los ejecutables se copian a un prefijo situado en una ruta no privilegiada.

Dado que no todos los sistemas son iguales, la ruta de este prefijo puede modificarse para adaptarse a las necesidades del usuario. Si el prefijo se sitúa en una carpeta compartida entre todos los nodos de un clúster, la ejecución distribuida de Hadoop puede realizarse de manera muy sencilla y sin copiar información.

Otro factor tenido en cuenta es que, por defecto, el compilador y el enlazador sólo buscan en las carpetas del sistema los ficheros de cabecera y las bibliotecas que pueden requerirse como dependencia. Para solucionar este problema, se modifican las variables de entorno locales durante la instalación para que el prefijo también se tenga en cuenta durante la búsqueda.

También es necesario modificarlas una vez finalizada la instalación para que la biblioteca de Expand pueda encontrar sus dependencias. Como Hadoop es un sistema distribuido, no es suficiente con que estén cambiadas en un solo nodo. Es por esto que dichas variables se modifican en el script de configuración del entorno de Hadoop, en cada uno de los nodos.

Capítulo 5

Validación de los resultados

En este apartado se modela una evaluación de la prueba de concepto con Expand que analice el funcionamiento del conector en diversas situaciones. Después, se ejecutan estas mismas evaluaciones con HDFS, el sistema de ficheros por defecto de Hadoop, de manera que se pueda realizar una comparativa. Finalmente, se presentan las conclusiones que se pueden extraer de los resultados obtenidos.

5.1. Parámetros de prueba

Las circunstancias bajo las que se prueba el conector con Expand deberían acercarse lo máximo posible a las condiciones técnicas a las que suele someterse en un entorno real. En este caso, alcanzar estos escenarios es complicado con los recursos disponibles, por lo que los datos deberán inducirse a situaciones con mayores requerimientos.

Los parámetros que se pueden variar para un sistema de ficheros paralelo en Hadoop y que permiten tanto toma de tiempos como pruebas de funcionalidad son los siguientes:

- **Tamaño de bloque**. En un contexto general, los sistemas operativos manejan tamaños de bloque de 4KB para sus sistemas de ficheros. En *big data*, donde se manejan ficheros grandes y se debe beneficiar el acceso a cantidades voluminosas de información, este tamaño suele ser superior a 64KB, encontrándose la media en el orden del megabyte.

- **Tamaño del conjunto de análisis**. Los conjuntos de análisis son un poco más complicados de estimar, ya que las empresas grandes pueden manejar conjuntos del orden de los petabytes, y las empresas pequeñas o medianas del orden de los terabytes. En cualquier

caso, los resultados pueden basarse en un único fichero de gran tamaño para reducir el tiempo de prueba y las características del entorno de pruebas. Para una comparativa, esto puede resultar suficiente para poder estimar los resultados en conjuntos de datos de mayor tamaño.

- **Número de nodos de almacenamiento de datos**. El sistema de ficheros paralelo puede soportar desde un único nodo para almacenamiento hasta un número que no supere los límites teóricos que presente su lenguaje de programación o su diseño. Al igual que el tamaño del conjunto de datos, es difícil estimar la media de nodos ya que existen fuertes diferencias entre las empresas grandes y las medianas o pequeñas.

- **Número de nodos de procesamiento**. Los límites en este caso los establece Hadoop, partiendo de un nodo en ejecución local hasta el número máximo que soporte el gestor de recursos utilizado. En el caso de YARN, que utiliza máquinas virtuales de Java como contenedores del procesamiento, puede alcanzarse un límite cuando se superan las capacidades de dicho entorno de ejecución.

- **Tipo de operaciones**. Hay gran variedad de operaciones que se pueden aplicar sobre un sistema de ficheros: lectura, escritura, creación de un fichero, eliminación de un directorio, etcétera. Sin embargo, un escenario real no realiza constantemente un mismo tipo de operación, por lo que probar cada una por separado puede inducir a resultados que no tengan ningún significado en la práctica. Por este motivo, se recurre a operaciones de alto nivel que se componen internamente de llamadas a múltiples operaciones simples. Para esto, resulta útil las utilidades de consola de Hadoop ara la interacción con el sistema de ficheros, o la ejecución de programas MapReduce de análisis de ejemplo.

Se ignora el factor de replicación para la tolerancia a fallos ya que no es el objetivo de estas pruebas evaluar el sistema mientras se aplica este mecanismo.

5.2. Entorno de ejecución

El entorno del que se dispone para la ejecución de las pruebas es un clúster formado por un total de ocho nodos. Cada nodo tiene las siguientes características:

Nodo	*Procesador*	*Frecuencia*	*Núcleos*	*Memoria*	*Almacenamiento*	*Sist. Operativo*
1	*Xeon E5405*	2.0 GHz	8	8 GB	2,00 TB	Ubuntu 16.04.2
2					1,05 TB	
3					2,00 TB	
4					0,08 TB	
5					2,00 TB	
6					2,00 TB	
7					2,00 TB	
8					2,00 TB	

En estos sistemas, la carpeta de usuario se encuentra compartida en un servidor NFS y el acceso a los nodos se realiza a través del protocolo SSH. Aunque ofrece comodidad colocar la instalación de Hadoop en una carpeta compartida, de cara a pruebas podría afectar negativamente al tener que obtener los datos por red para la ejecución.

Cada uno de los nodos cuenta con un servidor NFS que exporta el directorio local de cada uno de los nodos:

```
/exports/expand/nfs
```

Se dispone de un directorio con permisos de lectura, escritura y ejecución para el despliegue de la instalación de Hadoop en el directorio de cada uno de los nodos:

/exports/expand/inst

Por último, se comparte un mismo usuario en cada uno de los nodos, lo que evita complicaciones derivadas de utilizar múltiples usuarios. En un entorno de producción, esta práctica está desaconsejada en favor de utilizar diferentes usuarios para cada componente de Hadoop. De este modo se gana en seguridad y se proporciona un mayor aislamiento.

5.3. Definición de la batería de pruebas

Conociendo el entorno de ejecución y los parámetros que se pueden modificar, se define una batería de pruebas lo más completa posible y que se pueda llevar a cabo en un tiempo prudencial.

Tamaño de bloque	256 KB	1 MB	32 MB	64 MB
Tamaño del conjunto de datos	500 MB	1 GB		
Nº de nodos de datos	4	8		
Nº de nodos de procesamiento	1	4	8	
Tipo de operaciones	cp	cat	wordcount	grep

NOTA: Se descarta la ejecución en múltiples nodos de procesamiento dado que escapa al alcance de la prueba de concepto. Expand no soporta concurrencia y tiene un modelo de consistencia débil. Estas características impiden la ejecución de este grupo de pruebas. Sin embargo, se dejan planteadas para líneas de trabajo futuras.

El número total de escenarios que se prueban con la variación de estos parámetros es de:

Escenarios de prueba = (4 × 2 × 2 × 1 × 4) tipos × 2 sistemas de ficheros = 128 escenarios

5.4. Resultados de las pruebas

Se ha confeccionado un script que automatiza el proceso de obtención de resultados y ejecución de pruebas. Este script es sencillo: modifica los ficheros de configuración cíclicamente y lanza, finaliza y limpia las instancias de Hadoop junto con su respectivo sistema de ficheros.

5.4.1. Tabla de resultados usando HDFS

A continuación se listan las medias y desviaciones típicas de tiempos obtenidas para cada una de las configuraciones utilizando el sistema HDFS. El tamaño de bloque de 256KB no está permitido porque se encuentra por debajo del mínimo permitido para este sistema. Se han efectuado un total de 10 ejecuciones por escenario para mejorar la precisión:

Nº Nodos D.	T. Op.	T. Conjunto	T. Bloque	Media
4	cp	500 MB	256 KB	
4	cp	500 MB	1 MB	0min 34s 955ms
4	cp	500 MB	32 MB	0min 27s 356ms
4	cp	500 MB	64 MB	0min 21s 15ms
4	cp	1 GB	256 KB	
4	cp	1 GB	1 MB	1min 4s 280ms
4	cp	1 GB	32 MB	0min 54s 920ms
4	cp	1 GB	64 MB	0min 39s 403ms
4	cat	500 MB	256 KB	
4	cat	500 MB	1 MB	0min 27s 455ms
4	cat	500 MB	32 MB	0min 21s 123ms
4	cat	500 MB	64 MB	0min 17s 97ms

Nº Nodos D.	T. Op.	T. Conjunto	T. Bloque	Media
4	cat	1 GB	256 KB	
4	cat	1 GB	1 MB	0min 52s 550ms
4	cat	1 GB	32 MB	0min 42s 89ms
4	cat	1 GB	64 MB	0min 38s 942ms
4	wordcount	500 MB	256 KB	
4	wordcount	500 MB	1 MB	4min 13s 188ms
4	wordcount	500 MB	32 MB	3min 44s 743ms
4	wordcount	500 MB	64 MB	2min 58s 21ms
4	wordcount	1 GB	256 KB	
4	wordcount	1 GB	1 MB	6min 2s 259ms
4	wordcount	1 GB	32 MB	5min 34s 504ms
4	wordcount	1 GB	64 MB	3min 1s 879ms
4	grep	500 MB	256 KB	
4	grep	500 MB	1 MB	1min 26s 152ms
4	grep	500 MB	32 MB	0min 56s 841ms
4	grep	500 MB	64 MB	0min 41s 304ms
4	grep	1 GB	256 KB	
4	grep	1 GB	1 MB	2min 43s 433ms
4	grep	1 GB	32 MB	1min 24s 221ms
4	grep	1 GB	64 MB	0min 59s 311ms
8	cp	500 MB	256 KB	
8	cp	500 MB	1 MB	0min 39s 204ms
8	cp	500 MB	32 MB	0min 30s 588ms
8	cp	500 MB	64 MB	0min 18s 504ms
8	cp	1 GB	256 KB	
8	cp	1 GB	1 MB	1min 10s 86ms

Nº Nodos D.	T. Op.	T. Conjunto	T. Bloque	Media
8	cp	1 GB	32 MB	0min 54s 754ms
8	cp	1 GB	64 MB	0min 43s 603ms
8	cat	500 MB	256 KB	
8	cat	500 MB	1 MB	0min 31s 219ms
8	cat	500 MB	32 MB	0min 29s 741ms
8	cat	500 MB	64 MB	0min 18s 640ms
8	cat	1 GB	256 KB	
8	cat	1 GB	1 MB	0min 57s 490ms
8	cat	1 GB	32 MB	0min 53s 886ms
8	cat	1 GB	64 MB	0min 42s 799ms
8	wordcount	500 MB	256 KB	
8	wordcount	500 MB	1 MB	3min 9s 886ms
8	wordcount	500 MB	32 MB	2min 50s 876ms
8	wordcount	500 MB	64 MB	2min 56s 552ms
8	wordcount	1 GB	256 KB	
8	wordcount	1 GB	1 MB	6min 23s 277ms
8	wordcount	1 GB	32 MB	5min 35s 748ms
8	wordcount	1 GB	64 MB	5min 37s 747ms
8	grep	500 MB	256 KB	
8	grep	500 MB	1 MB	1min 29s 205ms
8	grep	500 MB	32 MB	0min 51s 899ms
8	grep	500 MB	64 MB	0min 43s 951ms
8	grep	1 GB	256 KB	
8	grep	1 GB	1 MB	2min 47s 435ms
8	grep	1 GB	32 MB	1min 34s 810ms
8	grep	1 GB	64 MB	1min 26s 142ms

5.4.2. Tabla de resultados usando Expand

A continuación se listan las medias y desviaciones típicas de tiempos obtenidas para cada una de las configuraciones utilizando el sistema HDFS. Se han efectuado un total de 10 ejecuciones por escenario para mejorar la precisión:

Nº Nodos D.	**T. Op.**	**T. Conjunto**	**T. Bloque**	**Media**
4	cp	500 MB	256 KB	1min 3s 976ms
4	cp	500 MB	1 MB	1min 3s 910ms
4	cp	500 MB	32 MB	1min 2s 589ms
4	cp	500 MB	64 MB	1min 1s 858ms
4	cp	1 GB	256 KB	2min 6s 873ms
4	cp	1 GB	1 MB	2min 7s 39ms
4	cp	1 GB	32 MB	2min 4s 723ms
4	cp	1 GB	64 MB	2min 5s 165ms
4	cat	500 MB	256 KB	0min 49s 3ms
4	cat	500 MB	1 MB	0min 48s 611ms
4	cat	500 MB	32 MB	0min 47s 996ms
4	cat	500 MB	64 MB	0min 47s 418ms
4	cat	1 GB	256 KB	1min 35s 462ms
4	cat	1 GB	1 MB	1min 33s 177ms
4	cat	1 GB	32 MB	1min 33s 453ms
4	cat	1 GB	64 MB	1min 33s 27ms
4	wordcount	500 MB	256 KB	8min 46s 158ms
4	wordcount	500 MB	1 MB	4min 1s 489ms
4	wordcount	500 MB	32 MB	3min 31s 586ms
4	wordcount	500 MB	64 MB	3min 30s 733ms

Nº Nodos D.	T. Op.	T. Conjunto	T. Bloque	Media
4	wordcount	1 GB	256 KB	31min 28s 41ms
4	wordcount	1 GB	1 MB	7min 27s 236ms
4	wordcount	1 GB	32 MB	6min 53s 792ms
4	wordcount	1 GB	64 MB	6min 58s 797ms
4	grep	500 MB	256 KB	3min 44s 457ms
4	grep	500 MB	1 MB	2min 22s 196ms
4	grep	500 MB	32 MB	1min 24s 776ms
4	grep	500 MB	64 MB	1min 24s 669ms
4	grep	1 GB	256 KB	27min 39s 485ms
4	grep	1 GB	1 MB	3min 52s 773ms
4	grep	1 GB	32 MB	2min 44s 814ms
4	grep	1 GB	64 MB	2min 42s 794ms
8	cp	500 MB	256 KB	1min 0s 840ms
8	cp	500 MB	1 MB	1min 1s 124ms
8	cp	500 MB	32 MB	1min 0s 659ms
8	cp	500 MB	64 MB	0min 58s 117ms
8	cp	1 GB	256 KB	2min 1s 163ms
8	cp	1 GB	1 MB	2min 1s 327ms
8	cp	1 GB	32 MB	1min 58s 317ms
8	cp	1 GB	64 MB	1min 57s 430ms
8	cat	500 MB	256 KB	0min 46s 237ms
8	cat	500 MB	1 MB	0min 45s 944ms
8	cat	500 MB	32 MB	0min 46s 212ms
8	cat	500 MB	64 MB	0min 45s 442ms
8	cat	1 GB	256 KB	1min 31s 148ms
8	cat	1 GB	1 MB	1min 30s 687ms

Nº Nodos D.	T. Op.	T. Conjunto	T. Bloque	Media
8	cat	1 GB	32 MB	1min 30s 752ms
8	cat	1 GB	64 MB	1min 29s 853ms
8	wordcount	500 MB	256 KB	5min 29s 761ms
8	wordcount	500 MB	1 MB	4min 18s 533ms
8	wordcount	500 MB	32 MB	3min 32s 683ms
8	wordcount	500 MB	64 MB	3min 34s 535ms
8	wordcount	1 GB	256 KB	31min 41s 669ms
8	wordcount	1 GB	1 MB	8min 20s 546ms
8	wordcount	1 GB	32 MB	6min 57s 870ms
8	wordcount	1 GB	64 MB	6min 58s 690ms
8	grep	500 MB	256 KB	3min 42s 59ms
8	grep	500 MB	1 MB	2min 26s 330ms
8	grep	500 MB	32 MB	1min 22s 988ms
8	grep	500 MB	64 MB	1min 22s 887ms
8	grep	1 GB	256 KB	27min 57s 784ms
8	grep	1 GB	1 MB	3min 52s 233ms
8	grep	1 GB	32 MB	2min 40s 318ms
8	grep	1 GB	64 MB	2min 37s 771ms

5.5. Conclusiones finales

Los test llevados a cabo muestran varias curiosidades respecto a ambos sistemas:

- Para ficheros de 500MB y 1GB, el comportamiento de HDFS es mejor con 4 nodos de datos que con 8. Esto puede deberse a que, para este conjunto, el aumento del servidores de datos perjudica el rendimiento en lugar de mejorarlo. Esto puede diferir si los conjuntos de datos son más grandes. Por otro lado, cuanto mayor es el tamaño de bloque menor es el tiempo en alcanzar los resultados. Esto puede deberse a que el procesamiento vence al tiempo de acceso a disco. Es decir, merece la pena en este caso hacer mayor número de operaciones por nodo que distribuir los datos entre más nodos y que hagan menor número de operaciones.

- En Expand, el rendimiento se mejora ante la presencia de mayor número de nodos de datos. Para las características de este sistema y los conjuntos de datos de 500MB y 1GB, emplear más nodos beneficia el rendimiento. Por otro lado, se percibe un problema en Java cuando se utiliza un tamaño de bloque de 256KB: se está produciendo un colapso de la memoria de la pila de Java que reduce drásticamente el rendimiento. Este valor parece ser demasiado bajo para el tipo de operaciones que efectúa Hadoop, siempre y cuando sean operaciones MapReduce como wordcount y grep y no operaciones directas con el sistema de ficheros. Es posible que esta sea la razón por la que HDFS lo limita como tamaño de bloque permitido. Por último, el tiempo de copia desde el sistema de ficheros local parece estar encontrando un cuello de botella muy evidente: para cualquier tamaño de bloque el tiempo es muy similar. Esto puede deberse a que el sistema de ficheros local divide los bloques en 4KB y las transferencias a Expand no se efectúan con eficiencia.

La desventaja comparando los resultados obtenidos para Expand frente a los obtenidos con HDFS puede venir dada por distintas razones:

- La interfaz de Hadoop está diseñada para HDFS. En algunos casos, esto puede derivar en un mayor número de cálculos para una misma operación, dado que la interfaz tiene que ser alterada por razones de compatibilidad.

- HDFS cachea los datos mientras que Expand no y muestra una curva mucho más pronunciada en cuanto al rendimiento con tamaños de bloque más grandes.

- No hay traducción de lenguaje cuando se comunican dos procesos Java. El impacto de JNI puede ser mayor sobre el rendimiento que la existencia de una máquina virtual para casos de este tipo.

- El NameNode puede optimizar operaciones retrasándolas para ciertos casos. Además, libera de cierta carga a los clientes. En Expand, son los clientes quienes soportan todo el peso de la implementación, mientras que los servidores se pueden considerar meros operadores.

Capítulo 6

Impacto socio-económico

En este apartado se analiza el impacto que el proyecto tiene en la sociedad y en el ámbito económico, con el fin de que un posible comprador conozca las ventajas y desventajas que puede reportar el uso de este software.

6.1. Repercusión social

Este proyecto supone una ayuda para los desarrolladores de sistemas de ficheros y para la comunidad de Hadoop en general. A nivel social, promueve la aparición más ágil de alternativas a HDFS, lo cual proporciona un mayor abanico de posibilidades a los usuarios de Hadoop y fomenta la competitividad.

Si se mejoran las herramientas de *big data* se beneficia a toda la sociedad, ya que es utilizada en una gran variedad de ámbitos e instituciones y con todo tipo de fines, ya sea transacciones bancarias, bancos de datos médicos, investigaciones sociológicas, etcétera.

No obstante, esto tiene su lado negativo. Como casi todas las herramientas software, existen usos maliciosos que perjudican al conjunto de la sociedad. De este modo, las herramientas de *big data* pueden utilizarse como apoyo para campañas de *phishing* dirigido, coordinación de *botnets*, despliegue de redes de espionaje masivo, etcétera.

A la hora de desarrollar este software, se ha entendido que las ventajas que puede ofrecer indirectamente a la sociedad compensan en gran medida sus aspectos negativos. Se espera, además, que los usos de *big data* encuentren nuevos campos y que las empresas decidan aprovechar sus ventajas para ofrecer un mejor servicio personalizado a sus clientes.

6.2. Repercusión económica

Este proyecto tiene ventajas económicas para las dos partes involucradas en la cuestión. Estas dos partes son las empresas que incluyen Hadoop en su estrategia corporativa respecto al *big data* y los distribuidores de sistema de ficheros.

Para el primer sector, el abaratamiento en la integración de sistemas de ficheros nuevos en Hadoop les abrirá a más alternativas, que les pueden suponer un ahorro en los recursos empleados para el análisis, tanto en gasto energético –reduciéndose la carga de los nodos con un sistema de ficheros más eficiente– como en recursos de otro tipo –los análisis pueden finalizar en menos tiempo y con mayor fiabilidad–.

Respecto a la segunda parte involucrada, la aparición de este software le supone un menor esfuerzo a la hora de integrar su sistema en Hadoop, por lo que pasará a ser una opción más manejable disponiendo de recursos limitados. Con una base de usuarios como la de Hadoop, esto se traduce en el incremento de la popularidad de su sistema. En caso de establecer cuotas por su uso, esto tiene como consecuencia directa un aumento del beneficio.

6.3. Líneas de trabajo futuras

La prueba de concepto realizada con Expand ha mostrado el potencial de este sistema de ficheros y la eficacia de su diseño de servidores de red neutros. Sin embargo, su implementación está lejos de ser perfecta todavía: las cuestiones de concurrencia y, en extensión, de consistencia de datos, no están resueltas aun.

Es por esto que se presenta una línea interesante de trabajo ampliando dicho sistema de ficheros y continuando su desarrollo. Esto posibilitaría ampliar las estadísticas tomadas con la prueba de concepto e incluir varias combinaciones de nodos de procesamiento en su esquema.

Por otra parte, el diseño de este conector está limitado a POSIX, pero es posible que surjan nuevas y mejores alternativas a este estándar que se pongan de moda entre los sistemas de ficheros paralelos. En este contexto, se pueden modelar nuevos conectores que sigan la misma filosofía de diseño y que introduzcan las mismas facilidades pero proporcionando una interfaz más adecuada a sus tiempos.

Conclusiones

En este trabajo se ha tomado contacto con tecnologías sofisticadas que están a la orden del día y estudiado su funcionamiento interno. Esto, combinado con el aprendizaje de JNI, de los diferentes entornos de compilación (libtool, Apache Maven) y con la programación de ciertas secciones de un sistema de ficheros paralelo (Expand), ha supuesto una gran diferencia de conocimientos respecto al inicio del proyecto.

Se ha apreciado también la inestabilidad del mundo del *big data*, donde se siguen buscando formas de adaptar las tecnologías actuales a nuevos modelos que permitan darles nuevos usos. Esta inestabilidad no es negativa, sino positiva. La gran variedad de alternativas y los distintos paradigmas que se presentan en el tiempo permiten elegir la mejor opción para cada escenario y evaluar nuevos algoritmos que antes podrían no haberse valorado.

Por último, la realización del conector genérico ha sido un largo proceso, lleno de obstáculos por resolver, pero merece la pena cuando se valora la ayuda que puede proporcionar a los desarrolladores, que disfrutarán de un camino más allanado para colaborar en el proyecto comunitario que representa Hadoop.

Anexo 1

Planificación del proyecto

En este anexo se expone la organización que ha seguido el proyecto en cuanto a los plazos y las etapas de su desarrollo. Con el fin de estructurarlo, se realiza una estimación inicial que se intentará seguir durante la ejecución del mismo. Al darse por finalizado, se vuelven a revisar estas etapas y plazos. Cualquier diferencia entre ambos puede significar un retraso o un adelanto frente a la estimación y se estudiará al final del apartado.

1.1. Etapas del proyecto

Habiéndose estudiado a priori las características del proyecto, se extraen una serie de tareas genéricas cuya realización tiene que llevarse a cabo para que su desarrollo sea satisfactorio.

- **Etapa 1**: Acotación del problema y estudio inicial de soluciones. En esta primera fase, se moldea y se define la propuesta del proyecto, explorando algunas soluciones que se pueden emplear para solventar el problema y manteniendo entrevistas con el tutor.

- **Etapa 2**: Análisis y toma de contacto con las tecnologías involucradas. Dada la inexperiencia con el entorno al que se asocia el problema, se explora su funcionamiento y se trata de poner en marcha las herramientas en un entorno de pruebas controlado.

- **Etapa 3**: Valoración del entorno y estudio final de soluciones. Vistas las características del entorno y comprendido su funcionamiento, se revisan las soluciones planteadas inicialmente, se retiran aquellas que se hayan desestimado y se agregan las nuevas que hayan podido aparecer.

- **Etapa 4**: Selección de solución y diseño. Del conjunto de soluciones final se extrae aquella que presente un conjunto de ventajas superior a las demás y se plantean sus posibles diseños. De entre todos los posibles, se elige el más ventajoso para su implementación.

- **Etapa 5**: Implementación de la solución. En base al diseño realizado, se realiza una implementación de la solución que resuelva el problema de la manera prevista.

- **Etapa 6**: Prueba de concepto. Se expone la solución a una situación específica que permita comprobar su eficacia en un escenario real.

- **Etapa 7**: Validación y obtención de estadísticas. Se realiza una validación de la prueba de concepto obteniendo estadísticas sobre su ejecución.

- **Etapa 8**: Comparativa frente a algunas alternativas existentes. Se realiza una obtención de estadísticas de otras alternativas a la solución y se comparan con las obtenidas con ella. De este modo, podrá equipararse su idoneidad frente el entorno.

- **Etapa 9**: Generación de la documentación asociada al proyecto. En esta última etapa, se recoge todo el conocimiento generado en las etapas anteriores y se genera la memoria del proyecto.

1.2. Estimación inicial

El inicio del proyecto tiene lugar con la presentación del mismo por parte del tutor. En dicho momento, se elabora una estimación que es la que se plantea a continuación.

1.2.1. Diagrama de Gantt

En el diagrama de Gantt para la planificación, se asocian etapas a distintos intervalos de tiempo dentro del total estimado para el proyecto. Se considera que su duración será de un año y la segmentación del diagrama se realizará por meses.

		Mes												
		2016							2017					
		Jun	Jul	Ago	Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun
Etapa	1	**x**	**x**											
	2			**x**										
	3				**x**									
	4					**x**	**x**							
	5							**x**	**x**					
	6									**x**	**x**			
	7											**x**		
	8												**x**	
	9													**x**

La propuesta del proyecto se plantea el 21 de junio de 2016. Se estima que en junio de 2017, el proyecto estará terminado y se procederá a la entrega de la documentación.

1.3. Planificación final

Antes de la entrega de la documentación del proyecto y durante la última etapa del mismo, se elabora una nueva planificación reflejando la situación real.

1.3.1. Diagrama de Gantt

Respecto a la estimación, el Gantt para la planificación final comparte la fecha de origen pero discrepa en la fecha de fin.

		Mes															
		2016							2017								
		Jun	Jul	Ago	Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun	Jul	Ago	Sep
Etapa	1	**x**	**x**														
	2			**x**													
	3				**x**												
	4					**x**	**x**										
	5							**x**	**x**	**x**							
	6										**x**	**x**	**x**	**x**			
	7														**x**		
	8															**x**	
	9																**x**

La propuesta del proyecto se plantea el 21 de junio de 2016. La entrega final de la documentación se produce en septiembre de 2017, 3 meses después frente a la estimación.

1.4. Retrasos

1.4.1. Etapas afectadas

Comparando los dos diagramas de Gantt, se puede comprobar que la etapa 5 (implementación de la solución) se ha extendido un mes más de lo esperado y la etapa 6 (prueba de concepto) se ha extendido dos meses más de lo esperado.

Como consecuencia, las etapas 7 (validación de estadísticas), 8 (comparativa frente a algunas alternativas existentes) y 9 (generación de la documentación asociada al proyecto) han empezado tres meses después de lo estimado. Ha resultado imposible acortar estas fases para lograr cumplir con la estimación y terminar el proyecto en el plazo esperado, ya que el retraso era demasiado grande.

1.4.2. Causas

Los retrasos experimentados se han sufrido por dos problemas que no se pudieron anticipar a la hora de realizar la estimación:

- La solución seleccionada depende de JNI, la interfaz de Java para la comunicación con bibliotecas nativas en lenguaje de programación C. Esta tecnología es compleja y no se tenía conocimiento previo de ella, por lo que requirió de un proceso de aprendizaje que extendió el tiempo de implementación para la fase 7.

- El sistema de ficheros paralelo utilizado para la prueba de concepto, denominado Expand, tenía ciertos problemas en su implementación que se tenían que corregir para que funcionase como se esperaba. La complejidad interna de este sistema y el desconocimiento de la implementación de NFS3 con el protocolo RPC extendió el tiempo de desarrollo de la prueba de concepto para la fase 8.

- Por otra parte, el sistema de compilación de Expand sólo estaba preparado para producir una biblioteca estática. Los requerimientos del conector y JNI fuerzan a usar una biblioteca compartida, por lo que se ha introducido *libtool* para obtener ambos tipos de biblioteca durante la compilación del código fuente de Expand. Esta modificación ha acrecentado el retraso ya existente sobre el desarrollo de la prueba de concepto de la fase 8.

Anexo 2

Presupuesto

En este anexo se ofrece un presupuesto para la realización de este software en el mercado actual y atendiendo a los gastos en que incurre. En primer lugar, se estudian los gastos directo e indirecto asociados al proyecto. En segundo lugar, se analiza el nivel técnico del proyecto, su público objetivo y la cantidad de alternativas para establecer un margen de beneficio adecuado. Finalmente, se presenta el presupuesto calculado sin IVA.

2.1. Gastos directos asociados al proyecto

Se comienza desglosando los gastos directos en los que se ha incurrido para desarrollar y documentar el proyecto. Para una mejor organización, se agrupan por categorías.

2.1.1. Salarios del personal

El desarrollo de este proyecto ha sido llevado a cabo por una única persona con rol de analista programador cuya jornada laboral ha sido de 4 horas diarias, con 5 días laborables por semana. El sueldo por hora de este perfil para proyectos de perfil especializado se encuentra en 40€ por hora en el mercado actual.

El desarrollo completo cubre un total de 305 días laborables, por lo que el número de horas total empleado es de 1.220 horas. El salario bruto total desembolsado para el desarrollador es por tanto de:

Desembolso en salarios = (40 € / hora) × 1.220 horas = 48.800 €

2.1.2. Dietas y gastos de transporte

Los desplazamientos realizados cuya finalidad es el desarrollo del proyecto también se cuentan como gastos asociados al proyecto.

El desarrollador utiliza transporte público y obtiene un abono mensual joven de un precio mensual de 20€. Sus viajes se utilizan para las reuniones con el tutor y para efectuar las pruebas en el clúster del departamento y el desarrollo del software.

Teniendo estas cuestiones en cuenta, el gasto total en dietas es de:

Desembolso en dietas = 20 €

2.1.3. Licencias de software

El software utilizado para la realización del proyecto se lista a continuación, junto al precio de cada herramienta. Se comienza por el software disponible en el entorno de desarrollo y que se ha utilizado tanto para la programación del código como para le elaboración de la documentación.

Nombre	Precio
(1) Sistema operativo macOS Sierra 10.12.6	€ 0,00
(1) Suite ofimática iWork (Pages, Numbers, Keynote)	€ 0,00
(1) Editor de código Atom	€ 0,00

Por otra parte, el entorno de pruebas hace uso del siguiente software, necesario para la ejecución de la prueba de concepto y el script de pruebas:

Nombre	Precio
(8) Sistema operativo Ubuntu 16.04.2	€ 0,00

En total, el gasto atribuido a las licencias de software es de:

Desembolso en licencias software = 0 €

2.1.4. Amortización de equipos

Se dispone de dos entornos: el entorno de desarrollo, en el que se programa el código y se realiza la documentación, y el entorno de pruebas, que es el clúster sobre el que se ejecutan las pruebas del código y se toman los tiempos.

Se establece un periodo de amortización de 4 años (48 meses) para el equipo informático utilizado y se debe recordar que la duración del proyecto es de 16 meses. Los precios y costes unitarios no incluyen el IVA. El valor en el momento de compra era del 21%.

Componentes del entorno de desarrollo	Precio unit.	Per. Amortización	Coste unit.
(1) Portátil MacBook Pro	€ 948,00	48 meses	€ 316,00
(1) Monitor externo Dell U2414H	€ 199,08	48 meses	€ 66,36
(1) Teclado y ratón inalámbricos Belkin	€ 18,96	48 meses	€ 6,32

El coste total asociado a la amortización de los componentes del entorno de desarrollo es de:

Amortización para el entorno de desarrollo = 316 € + 66,36 € + 6,32 € = 388,68 €

Componentes del entorno de pruebas	Precio unit.	Per. Amortización	Coste unit.
(8) Nodo del clúster	€ 1185,00	48 meses	€ 395,00

El coste total asociado a la amortización de los componentes del entorno de pruebas es de:

Amortización para el entorno de desarrollo = 395,00 € × 8 = 3.160,00 €

No se utilizaron más equipos durante el transcurso del proyecto, por lo que el total de costes de amortización asciende a:

Desembolso en amortizaciones = 388,68 € + 3.160,00 € = 3.548,68 €

2.1.5. Total de gastos directos

Sumando todos los valores desglosados, se concluye que el proyecto ha incurrido en unos gastos directos por un valor de:

Categoría	Monto total
Salarios del personal	€ 48.800,00
Dietas y gastos en transporte	€ 20,00
Licencias de software	€ 0,00
Amortización de equipos	€ 3.548,68
	€ 52.368,68

2.2. Gastos indirectos asociados al proyecto

Los gastos indirectos son aquellos que surgen entorno al proyecto como fruto de la actividad humana y derivados del uso de equipos informáticos. En lugar de presentar un desglose completo, se estima que los gastos indirectos equivalen a un 15% del total de gastos del proyecto.

2.3. Cálculo del presupuesto total

Las últimas cuestiones a aclarar para poder aportar un presupuesto son el porcentaje empleado como colchón –dado el riesgo que supone la realización del proyecto– y el porcentaje de beneficio que se obtiene.

El porcentaje dedicado al riesgo se establece en un 10% respecto al total del gasto del proyecto. Este valor se elige al haber evaluado que la cantidad resulta suficiente para cubrir cualquier eventualidad.

Por otra parte, el beneficio se ajusta teniendo en cuenta la falta de alternativas a este conector y las ventajas que puede reportar su uso. Una cifra coherente pero que tampoco busca la explotación de la situación ni poner en riesgo las ventas por un presupuesto excesivo para el proyecto es el 20% respecto al total del gasto del proyecto.

Con estas dos cuestiones confirmadas, el presupuesto total desglosado, sin IVA, es de:

Concepto		Monto total
Gastos directos	€	52.368,68
Gastos indirectos	€	7.855,30
Fondo de riesgo	€	6.022,40
Beneficio	€	12.044,80
	€	78.291,18

Anexo 3

Marco regulador

En este anexo se describen los estándares técnicos, paquetes de software y dependencias a los que se hace alusión a lo largo de este documento y que se encuentran relacionados con el proyecto. Se incluyen también otras referencias que pueden necesitar de aclaración.

3.1. Interfaces de aplicación

3.1.1. Especificación POSIX

El acrónimo POSIX (Portable Operating System Interface) hace referencia a una familia de estándares mantenida por la IEEE Computer Society y cuyo objetivo es garantizar la interoperabilidad entre los sistemas operativos[15].

En este documento, las alusiones a POSIX se refieren, en concreto, a las interfaces que define para la interacción con el sistema de ficheros, como pueden ser open(), close(), mkdir(), read(), etcétera.

3.2. Paquetes software y dependencias

3.2.1. Expand

La biblioteca que implementa el cliente para este sistema de ficheros es de dominio público y cuenta con una licencia GPLv2. Se ha mantenido en el interior de su código fuente la información acerca de la licencia que debe incluirse en la distribución del software por ley.

3.2.2. Dependencias de Expand

La biblioteca de Expand depende de una serie de paquetes de software y requiere el despliegue de servidores NFS3 para funcionar. Todos estos programas son de dominio público y de licencia variada, pero todas permiten su libre distribución y modificación siempre que se distribuya la información sobre la licencia junto al código fuente. Se ha comprobado que esta condición se cumple en todos los casos.

3.2.3. Oracle Java JDK 8

La prueba de concepto incluye los binarios de Java JDK 8 para disponer de una instalación de Java aislada e independiente. El uso de este software está limitado por la licencia BCL de Oracle. Esta licencia permite el uso del JDK gratuitamente para fines no comerciales pero puede establecer restricciones legales en algunos ámbitos. Se invita a los usuarios de la prueba de concepto a leer las condiciones y decidir si su uso entra dentro de las limitaciones establecidas por la licencia.

Para una implementación alternativa con licencia libre GPLv2, recurrir al proyecto OpenJDK, que dispone de un entorno Java 8. No se garantiza, sin embargo, el funcionamiento de Apache Hadoop o Apache Maven en este entorno.

3.3. Licencia de este software

Aunque este producto se comercializa, esto no significa que su código fuente se encuentre bajo una licencia privativa. Dado que la mayoría de herramientas utilizadas en el proyecto son software libre y el proyecto Apache Hadoop está construido mayoritariamente por voluntarios y bajo licencia Apache, parece injusto crear soluciones que rompan con este esquema y que no protejan los derechos del usuario del software.

Por esta razón, todo el código fruto de este proyecto se vuelca al dominio público con una licencia GPLv2. La presente documentación se rige también por sus condiciones y puede distribuirse y modificarse libremente, siempre y cuando se incluya esta licencia.

Anexo 4

Resumen (English)

4.1. Abstract

Integration of new file systems in Apache Hadoop is a long and complex task due to its implementation in a high level language (Java) and its interface, which doesn't conform to any popular standard used in those systems.

To diminish or solve this problem, a fairly generic connector is proposed. Its main goal is to hide the Hadoop FileSystem interface and provide a POSIX interface for a C application to use.

As a proof-of-concept, a connection with Expand parallel filesystem is included in the project. This system has been designed by the research group of computer architecture, communications and systems at Universidad Carlos III de Madrid.

4.2. Introduction

4.2.1. Scope

This project is related to different areas of the computer science field. The project itself requires qualification in the distributed, concurrent and parallel system areas, but its benefits apply to many other study subjects, like data mining and analysis, machine learning or artificial intelligence. The target of this project are parallel filesystem developers.

The scope of this project includes the technical design, its code implementation and a usable proof-of-concept. Furthermore, validation results are provided for the reader to judge. This whole document is also part of the final product handed to the user, including a budget calculation, some pieces of analysis that heighten its importance and every kind of detail into the project development.

4.2.2. Context

The world of computing is suffering recently from the shockwaves of *hyperconnectivity*. This phenomenon can be summarized as people's urgent need of connection with everybody else in the world and the events in their surroundings. This need is nothing new, but spread of new technical solutions into mainstream that make this possible is increasing it and making this need a lot more demanding than it used to be, asking for more and better ways to satisfy it.

This situation is affecting multiple fields in computer science, making the perfect breeding ground for new study subjects and areas. One of this incorporations is *big data*, a concept which gained focus as a direct consequence of hyperconnectivity. As a matter of fact, this is not a new concept either. Big data is the field that studies technics for huge data set analysis and storage in contexts where data size make any other traditional approach obsolete.

Companies and corporations, which are now beginning to accept this new landscape, have been evolving and changing their strategies towards it. They need to do this in order to keep relevant and take advantage of the circumstances. As part of this setting, they are investing heavily in big data technologies that enable them to achieve a new set of objectives. Some of them are:

- Share any kind of information with a huge user base concurrently and without any significant delay.

- Analyze data retrieved from users with many purposes, like marketing, trends, tracking or statistics; in the shortest time possible.

- Deploy cheap, simple and efficient solutions that get their job done without using too many resources and recycling those which are currently available.

- Distribute information, keeping the same conditions it met before, so that a better service can be provided to a bigger user base without suffocating computer systems.

With this market skyrocketing, tools to satisfy new customer needs have arised. Most of this new tools are still in an early adoption stage, being the most demanded profile the one of a framework which incorporates multiple technological advancements and facilitates big data processes by making them invisible to the end user. In this category, the top applications are Apache Hadoop and Apache Spark, but some alternatives can be found that make the same or even a better job. Some of this tools are Apache Storm, Apache Samza, Apache Flink and Apache Kafka. The Apache community has proven itself very productive in the field, as this set of solutions demonstrates.

This frameworks require a degree of technological sophistication and complexity that is increasing year by year. Some of the most important advancements that help them keep their pace are parallel file systems. They are the engines that keep track and manage every piece of information that enters and exits the working load. They also provide the abstract view of data that enables the user to see a very complex system as a traditional artifact.

This task is not easy at all. Storing and analyzing big data in a discreet amount of time impose an aggregation of complex conditions that are hard to meet keeping consistency and fault tolerance. This is the kind of environment that this project joins in.

4.3. Goals

The main goal of this project is to develop a generic connector for Apache Hadoop that provide a mean to integrate file systems in it through a POSIX interface.

This goal hopes to cover a set of partial objectives through its execution:

- **Objective 1**. Reduce the difficulty of integrating new file systems into Hadoop that use a popular standard for their interface.

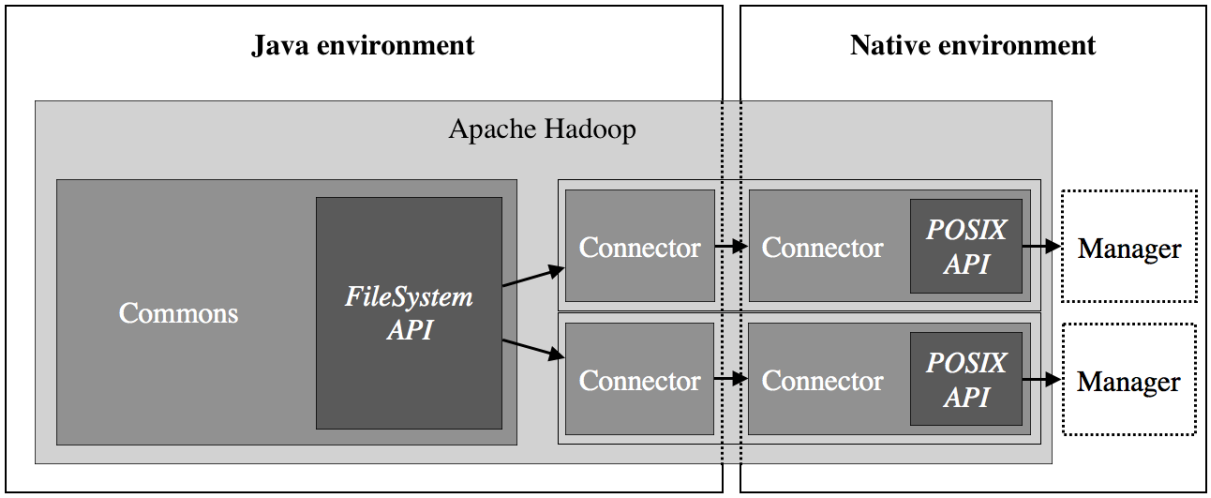
- **Objective 2**. Increase the file system spectrum available through Hadoop, which is currently limited.

- **Objective 3**. Test the Expand parallel file system in a Hadoop environment.

- **Objective 4**. Put to an end the long data migration processes when the file system that store it is not available through Hadoop.

4.4. Solution design and implementation

After a thorough study of currently available file systems for Hadoop in order to use the best approach, the final design adopted for the generic connection is described in this image:



This design allows for multiple connectors to co-exist and interoperate in the same Hadoop installation, but also implies an additional task for the developer in order to resolve the namespace conflict problem. This is, either the package name of the connector classes or the class name itself needs to be replaced, as two equal classes of the same package colliding in the same Hadoop installation leads to conflict.

The connector consists of two libraries, that need to be included in two different directories in the Hadoop home. The first one is written in Java, so it is presented through a ".jar" file and needs to be stored in the "lib/native" relative folder. The second one is written in C, so it is present through a ".so" file and needs to be stored in the "share/hadoop/common/lib" folder.

The Java library implements the FileSystem API exposed by Hadoop. As its functions travel to a more low level space, they communicate with the C library through a Java interface

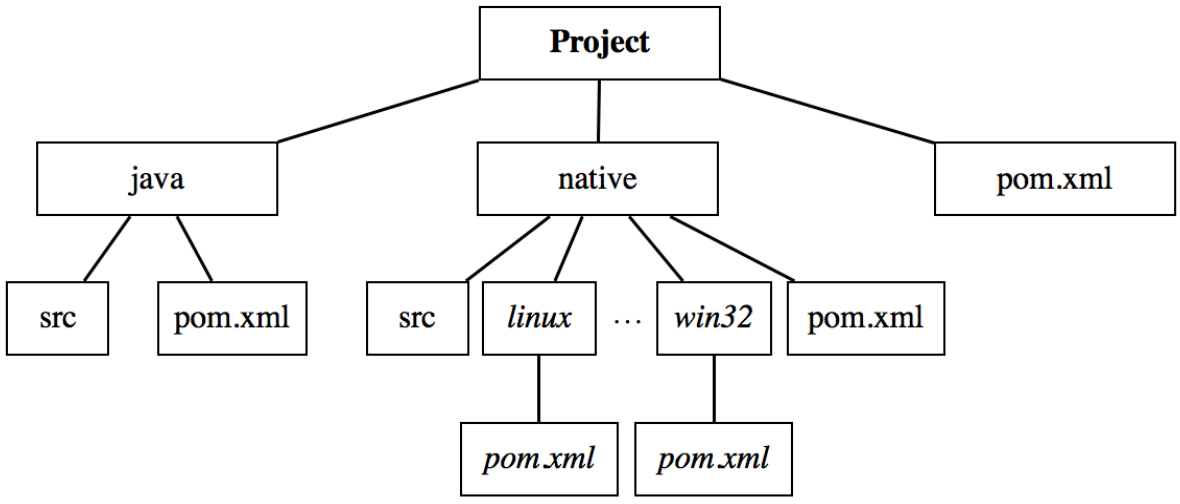
known as JNI, Java Native Interface, that allows the usage of native functions in the JVM, Java Virtual Machine.

In this process, data structures are translated so that they are available in C. In order to make interaction with the JVM possible, the C JNI headers provides functions to communicate with its environment. This way, a structural, not object-oriented language like C can create Java objects and use its particular classes and methods.

The C library provides an interface very similar to POSIX (but not fully compliant with it as it involves two complementary functions) that is to be implemented by the parallel file system to integrate in Hadoop. Each of this interface functions is called by the library as needed and translates and reports its results back to the Java library through JNI again.

This approach groups all the common tasks that any POSIX file system has to do to translate its semantics to the Hadoop FileSystem interface and allows to separate the file system development from the connector, which, in fact, can have its independent development.

This two libraries are obtained through a packaging tool, Apache Maven. The project needs to be correctly organized in order to separate C and Java compilation.



This is the main structure of the project. There are four compilation units:

- **hadoop-connector-fs**: represents the core compilation unit, defining file codification, plugin dependencies and compilation order for the other modules.

- **hadoop-connector-fs-java**: this module compiles the Java library for the connector. It downloads the Hadoop Common project as a dependency, as it contains classes and interfaces involved in the connector development. As a result, it generates *hadoop-connector-fs-java.jar*.

- **hadoop-connector-fs-native**: this module acts as a parent and selector for the C Library compilation, as it is system-dependent. It also downloads the Hadoop Common project, as it needs the JNI translations for its classes. Furthermore, it forks the compilation of the library according to the -P parameter of the Maven executable call, that indicates operating system to compile for and other system-dependent parameters.

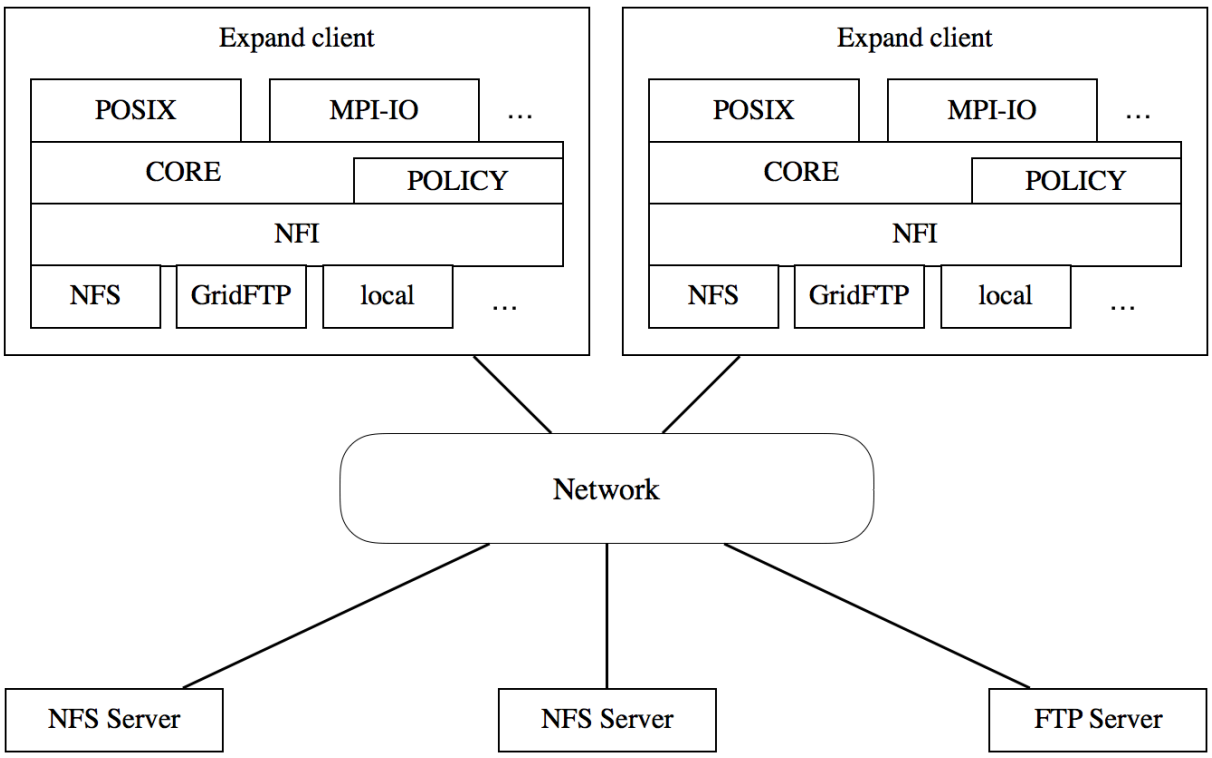
The fourth module is variable, but only one can execute for a compilation process.

- **libconnectorX**: this module is designed for a GNU/Linux native library and generates the file *libconnectorX.so*.

- **connectorX**: this module is designed for a Win32 native library and generates the file *connectorX.dll*.

The X in the last module needs to be replaced by the developer with a unique identifier of the file system that is to be integrated. This is also part of the namespace conflict resolution. In this case, it is more of a file problem: two files cannot be named the same in most file systems, and Java may not find the library through JNI in that case.

As this design cannot be tested without a file system implementation and its corresponding connection, a proof-of-concept using Expand parallel file system has been developed. Its designed is based upon "neutral" network file servers, which implementation is defined by a user-defined configuration file. Its architecture is as follows:



This design allows the exploration of the data in each node using the default protocol for the selected neutral network file server, while maintaining its abstract view when accessing them through the Expand client.

This approach allows to define an heterogeneous set of servers, working without their knowledge for the file system. At the same time, this removes processing overhead from the servers, as clients implement all of the file system functionality and they only act as mere data operators.

4.5. Results

With the development of a proof-of-concept connecting Expand parallel file system with Hadoop, the feasibility of the connector approach has been proved functional. The integration of Expand has involved few additional lines of code, as it is POSIX-compliant.

The test conducted over this implementation has exposed a better performance of HDFS compared to Expand. This seems to be a mixture of various factors:

- Hadoop FileSystem interface is design to fit HDFS. In some cases, this can lead to less computations per operation in comparison to other file systems that implement a different design and need conversion. This is the case of the connector itself.

- HDFS caches data and seems to be much more efficient as it shows a steep curve of performance when the block size is increased.

- There is no language translation between two Java processes, even though they run in a JVM. The complexity of JNI and type conversion could be introducing some latency on function calling.

- The NameNode can optimize operations by deferring some of them, as it interacts directly with the DataNodes for some operations. This could be hiding some processing time to the client, as it can be responded immediately after its interaction with the NameNode.

4.6. Conclusion

In this project, a first contact with sophisticated and state-of-the-art technologies has been made, as well as a study of their inner functionality. This, combined with the new acquired knowledge in JNI, compilation environments like libtool and Apache Maven, and parallel file system implementation in Expand, has made a whole difference.

The instability of big data has also proved existent, as the search for new ways to adapt current technology to newer models that gives it new uses is progressing. This instability is not, by any means, negative. In fact, it is quite positive. The great variety of alternatives and the multiple paradigms that has been appearing through history let everyone chose the option that best fits the context and evaluate new algorithms that couldn't have been taken into account before.

The development of the connector has been a long proccess, full of obstacles to deal with, but the effort is worth it compared to the help that it can be to the community, that will now enjoy an easier path towards collaboration with Hadoop.

Bibliografía

[1] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[2] Barr, V., & Stonebraker, M. (2015). A valuable lesson, and whither Hadoop?. Communications of the ACM, 58(1), 19.

[3] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S. H., Qiu, J., & Fox, G. (2010, June). Twister: a runtime for iterative mapreduce. In Proceedings of the 19th ACM international symposium on high performance distributed computing (pp. 810-818). ACM.

[4] Dittrich, J., & Quiané-Ruiz, J. A. (2012). Efficient big data processing in Hadoop MapReduce. Proceedings of the VLDB Endowment, 5(12), 2014-2015.

[5] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D. (2016). Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 17(1), 1235-1241.

[6] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012, April). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 2-2). USENIX Association.

[7] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM.

[8] Borthakur, D. (2008). HDFS architecture guide. Hadoop Apache Project, 53.

[9] Schmuck, F. B., & Haskin, R. L. (2002, January). GPFS: A Shared-Disk File System for Large Computing Clusters. In FAST (Vol. 2, No. 19).

[10] Wang, F., Oral, H. S., Shipman, G. M., Drokin, O., Wang, D., & Huang, H. (2009). Understanding Lustre Internals (No. ORNL/TM-2009/117). Oak Ridge National Laboratory (ORNL); Center for Computational Sciences.

[11] Schwan, P. (2003, July). Lustre: Building a file system for 1000-node clusters. In Proceedings of the 2003 Linux symposium (Vol. 2003, pp. 380-386).

[12] Garcia-Carballeira, F., Calderon, A., Carretero, J., Fernandez, J., & Perez, J. M. (2003). The design of the Expand parallel file system. The International Journal of High Performance Computing Applications, 17(1), 21-37.

[13] Calderón, A., García, F., Carretero, J., Pérez, J., & Fernández, J. (2002). An implementation of MPI-IO on Expand: A parallel file system based on NFS servers. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 337-345.

[14] Alam, A., & Ahmed, J. (2014, March). Hadoop architecture and its issues. In Computational Science and Computational Intelligence (CSCI), 2014 International Conference on (Vol. 2, pp. 288-291). IEEE.

[15] The Open Group Base Specifications Issue 7. IEEE Std 1003.1, 2008-2016 edition. IEEE.

[16] Liang, S. (1999). The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley Professional.

[17] Nguyen Xuan, J., & Dönszelmann, M. (2011, October). AC/C++ Build System Based On Maven for the LHC Controls System. In Conf. Proc. (Vol. 111010, No. CERN-ATS-2011-202, p. WEPKS026).