[1]

HDFS (Noviembre 2017)

Quim Aguado Puig, *UAB*, Alejandro Alonso, UAB

***Abstract*—Los sistemas de ficheros distribuidos o SFD, están presentes en nuestra vida diaria y los usamos continuamente. Sin embargo, estos son transparentes a nosotros y no nos dejan ver todo el trabajo que realizan. En este paper explicamos HDFS que pertenece a esta categoría de sistemas, haremos una profunda explicación de su arquitectura y funcionamiento, explicando cómo resuelve ciertos problemas comunes entre los SFD. Pero no sin antes hacer una explicación de que son los SFD, cómo funcionan y cuales son estas problemáticas comunes que deben resolver. Por último compararemos HDFS con otros SFD como por ejemplo GFS y Cassandra.**

I. INTRODUCCIÓN

Los SFD se utilizan para construir una visión jerárquica de los múltiples servidores de archivos que tenemos en la red y de archivos compartidos externos. Esto significa que en vez de buscar una máquina con los archivos que queremos, el usuario solo tiene que obtener un identificador que nos llevará hasta el sistema de archivos compartido y dentro de este sistema podremos encontrar el archivo que buscamos sin especificar la máquina.

Los SFD guardan un tipo de archivo concreto, el cual llamamos archivo de almacenamiento permanente. Esto consiste en que, (1) los archivos se crean de forma explícita, (2) son inmunes a cualquier fallo del sistema, (3) persisten hasta que se especifica su destrucción. En los SFD vemos cómo estos archivos se almacenan de forma compartida entre diferentes nodos.

Una idea muy importante de los SFD actuales es que el acceso a un archivo remoto puede hacerse de forma similar a el acceso a un fichero local, aunque en realidad el proceso sea mucho más complejo. Esto es el concepto que entendemos como transparencia, y es una de las características importantes que determinan la calidad de un SFD.

Los SFD tienen transparencia y redundancia para evitar fallos de servicio durante un fallo del sistema o un momento donde la carga del sistema sea muy elevada. Esto se realiza agrupando archivos que están replicados en varios nodos bajo el nombre de un solo directorio o nodo central del sistema. Muchos SFD tienen un rendimiento bajo comparado con un sistema de archivos local, debido a que tienen que realizar operaciones de I/O en la caché para la coherencia y seguridad de los datos. Otro de los problemas que los SFD suelen tener que tratar es el fallo de uno de los nodos o la caída de un canal de comunicación entre sus nodos..

II. CARACTERÍSTICAS DE UN SFD

Una vez vistas las principales problemáticas de los SFD, la forma más fácil de diferenciarlos es ver cómo trata cada uno estos problemas comunes que tienen.

A. Arquitecturas

Arquitectura cliente-servidor: Proporciona una visión estándar de su sistema de archivos local. Esta arquitectura funciona con un protocolo de comunicación que permite a los clientes acceder a los archivos de un servidor formado por una serie de nodos que tienen procesos de diferentes sistemas operativos trabajando en un sistema de archivos compartido entre ellos. Esta arquitectura permite un fuerte independencia de los sistemas de archivos locales de cada nodo.

Arquitectura cluster-based distributed file system: Consiste en un nodo maestro que dirige grupos de servidores agrupados en 64 Mbytes cada uno. La ventaja es que permite un buen control de todo el servidor a través de la administración del nodo maestro y proporciona gran escalabilidad al poder añadir muchos nodos debido a la fácil administración.

Arquitectura simétrica: Se basa en la tecnología peer-to-peer utilizando DHT para distribuir los datos y les asigna un identificador para poder localizarlos. Es decir, los clientes son los propios hosts que contienen los metadatos sobre la ubicación de los archivos del sistema. De esta forma todos los nodos tienen conocimiento de la estructura de todos los discos.

Arquitectura asimétrica: Al contrario que la simétrica, lo que encontramos son algunos nodos dedicados exclusivamente al manejo de los metadatos del sistema, que mantienen en sistema de archivos y las estructuras de disco asociadas a él.

B. Procesos

Encontramos 2 tipos de SFD en esta temática. Tenemos los sistemas “stateful” en los cuales se guarda un registro de todo lo que se modifica en el sistema. Los otros son sistemas stateless los cuales no guardan ningún registro o tipo de información sobre las modificaciones del sistema. Respecto a la escalabilidad del sistema es mucho mejor usar un sistema stateless ya que al no manejar datos de las operaciones del propio sistema, esto nos ahorra una carga de trabajo que puede provocar overhead. pero sin embargo los sistemas de este tipo son difíciles de implementar.

[1]

C. Comunicación

La mayoría de los SFD utilizan el "Remote Procedure Call" (RPC), este les permite comunicarse de forma independiente al sistema operativo, la red y el protocolo de transporte que poseen.

El RPC tiene 2 protocolos de comunicación a tener en cuenta, el TCP y el UDP. El TCP es el más usado, sin embargo por ejemplo hadoop utiliza el UDP para acelerar según que comportamientos.

Otro sistema de comunicación es el "plan 9", que consiste en que todos los recursos son accedidos de la misma forma, utilizando un nombre similar a la sintaxis de archivos local y unas operaciones que indican lo que ahcer con el archivo. Esto nos permite una gran ventaja que es la independencia de redes. Este sistema se trata en el SFD llamado Lustre.

D. Objetos

Es realmente significativo el hecho de cómo cada SFD pone un pathname y una dirección física a cada uno de sus archivos, ya que el conjunto de todos los pathnames genera el espacio o dominio en que se encuentra nuestro sistema distribuido. Este pathname es lo que utilizamos para acceder a las direcciones donde accedemos a la información del sistema. No solo a los archivos sino a todos los metadatos del sistema. Es importante que para gestionar este tema, los sistemas mantengan la transparencia que se les exige, por eso no es un tema trivial.

La mayoría de sistemas utilizan el método de centralizado de los metadatos en un servidor. Esto significa que habrá un nodo en concreto que se encargue de contener estos metadatos y de gestionar los pathnames de todo el sistema. Este método es ventajoso en el sentido que, desacoplar los datos y los metadatos reduce el estrés que sufre el throughput de los nodos de almacenamiento de datos.

La segunda opcion seria distribuir los metadatos entre todos los nodos que también poseen los datos. Lo que resulta en tener conocimiento de donde están todos los datos de disco en todos los nodos.

E. Sincronización de datos

En un SFD como hemos dicho, los ficheros están replicados en diferentes nodos para evitar la pérdida de datos por la caída de un nodo. Esto genera muchos conflictos cuando queremos hacer lectura y escritura de datos, especialmente la escritura. Ya que tendremos que definir políticas de actualización de datos cuando modificamos un archivo en algún nodo. Y más importante aún, tendremos que imponer políticas de acceso a los datos, para impedir que algun archivo pueda ser leído antes de que sea actualizado de una escritura que se ha hecho en otro nodo u otros conflictos que se pueden generar con el acceso a archivos que están siendo modificados.

Las políticas más comunes dentro de los SFD son las de poner candados a los objetos de forma que se otorga acceso a un cliente a ese objeto y acto seguido se cierra el candado para evitar el acceso de otro cliente a él. Otro método serían los "leases", que son una especie de permisos temporales que se otorgan a los clientes para hacer según que operaciones en el sistema.

F. Consistencia y replicación

Para proporcionar consistencia, la mayoría de SFD utilizan checksum para validar los datos que se envían a través de la red.

Para realizar el caching y la replicación de datos sí que encontramos diferentes políticas, lo más común es el client-side caching y server-side replication. Para realizar la replicación se tienen en cuenta los metadatos y los datos, y dependiendo del sistema se da más importancia a unos u a otros, replicando más unos que otros.

Los datos se replican cuando el sistema considera que hay un fallo en un nodo con estos datos. Cada sistema considera los fallos de una forma diferente. Mientras que algunos consideran que los datos están seguros mientras el nodo físico siga activo, y si este cae se tratará como una excepción. Otros sistemas que necesitan la disponibilidad de los datos immediata son más exhaustivos con este tema y al detectar un fallo de proveer datos, ya asumen que este nodo no contiene estos datos y replican inmediatamente ese dato en otro nodo.

G. Tolerancia de fallos

Ante la caída de un nodo con datos, hay sistemas que simplemente lo ignoran y lo tratan como una excepción en el sistema y simplemente recuperan ese nodo al estado en que estaba antes de caerse, mientras que hay otros que lo trataran como una norma y iniciaran su protocolo de resolución de fallos, donde replican los datos una vez más para no bajar de un mínimo de réplicas del sistema.

H. Seguridad

Los principales problemas en al seguridad de los SFD suelen ser los problemas de autentificación y control de acceso. La mayoría de sistemas aprovechan sistemas de seguridad que ya existen. Sin embargo hay algunos como google o hadoop que prefieren confiar en sus propios nodos no utilizan mecanismos de seguridad no dedicados en su estructura.

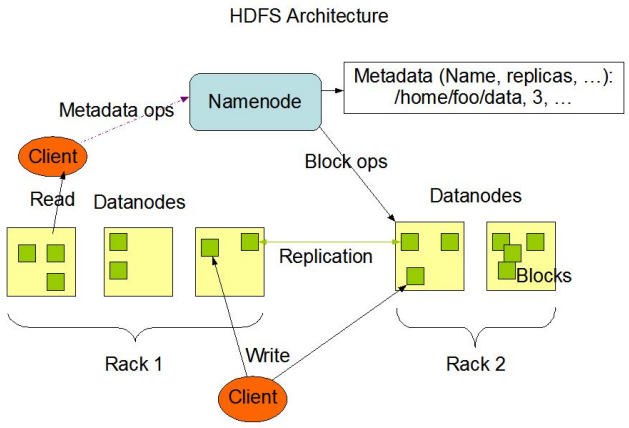
III. ARQUITECTURA DE HDFS

HDFS está diseñado para poder ser ejecutado en casi cualquier hardware, y para ser altamente tolerante a los fallos, para ello se asume que el fallo de nodos en grandes

infraestructuras es la norma más que la excepción.

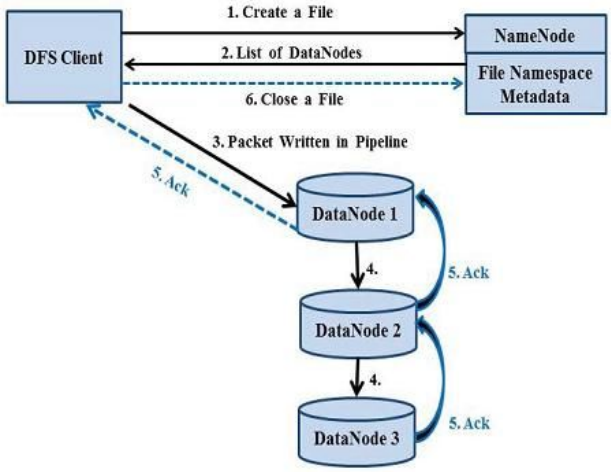
Es un sistema de ficheros software, es decir que trabaja sobre otro sistema de ficheros subyacente gestionado por el sistema operativo (cómo podría ser ext4 y Linux), esto hace que sea más fácil implementarlo, y más portable entre plataformas pero se pierde algo de velocidad, y se depende de la correcta implementación y limitaciones del sistema de ficheros subyacente, por ejemplo en sistemas FAT dónde el tamaño de los ficheros es limitado.

Su arquitectura es master/slave, es master se denomina NameNode y maneja los metadatos del SFD, mientras los slaves se denominan DataNodes y son los que guardan los datos. Los ficheros son divididos en bloques, estos se reparten entre los DataNodes haciendo más de una copia de cada uno de ellos, cosa que mejora la tolerancia de fallos.



A. Replicación de datos

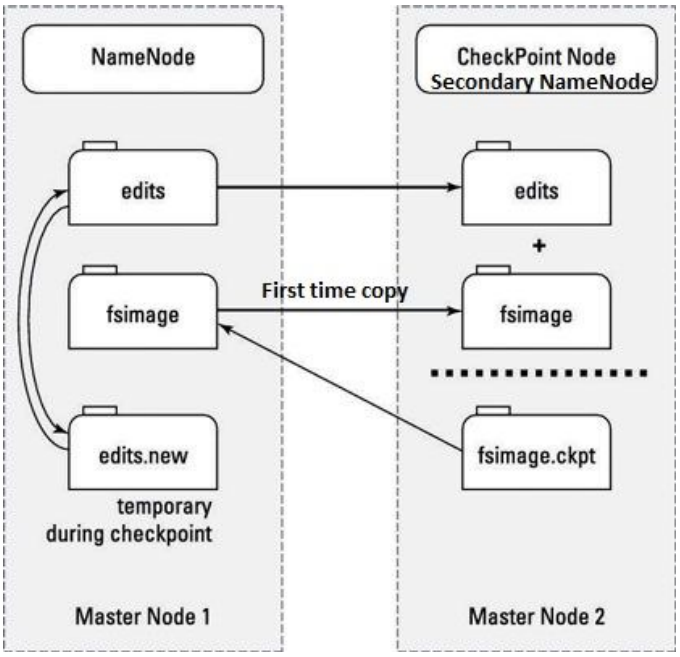
Cómo se ha dicho, HDFS asume que los fallos de nodos son norma. Para afrontar este problema y a la vez poder guardar grandes archivos de forma segura en sistemas distribuidos, los bloques de cada archivo se replican un número determinado de veces (por defecto 3) entre los DataNodes. El responsable de organizar la replicación es el NameNode y si detecta que un nodo se ha caído, automáticamente da la orden de replicar todos los bloques que estaban en ese nodo.
Para la replicación se usa la técnica del *pipelining*, es decir, que en vez de que el cliente envíe los datos 3 veces, a los 3 DataNodes que contendrán el bloque, el cliente le envía los datos el primer DataNode, éste al segundo, y el segundo al tercero. Con ésta técnica se consigue evitar la congestión de la red.



B. NameNode

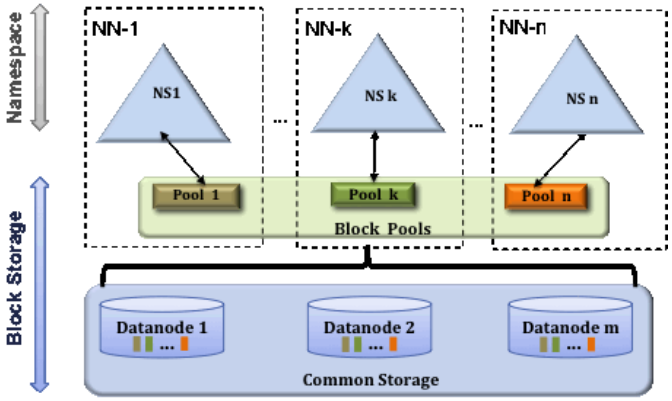
El NameNode es el nodo principal que maneja los metadatos del sistema de ficheros. Los ficheros están organizados en inodos que residen en memoria, es decir, no son persistentes.
Esto genera el problema de que, al reiniciar la máquina, todos los inodos se pierden. HDFS aborda este problema con dos ficheros en el NameNode, el *fsimage* y el *edits* (o *journal*). En el fsimage está guardado el estado del sistema de ficheros la última vez que el NameNode se inició, y en el *edits* están guardados todos los cambios que se han hecho desde entonces. Se fusionan estos dos ficheros para obtener el estado actual del SFD.

Otro problema sale de aquí, en sistemas muy grandes el tiempo de reinicio del NameNode puede ser enorme al tener que rehacer todos los inodos. En este caso se puede tener un NameNode secundario, que se encarga de ir haciendo periódicamente la unión del *fsimage* y el *edits* y ir actualizando de esta forma el *fsimage* del NameNode principal.



En clusters muy grandes, la memoria principal del NameNode es un factor limitante ya que los inodes tienen que guardarse allí. También se depende mucho de un solo NameNode, lo que hace el sistema menos tolerante a fallos. La solución que HDFS nos ofrece es el *HDFS Federation*.

La federación de NameNodes en HDFS es una técnica que nos permite escalar nuestro sistema, dividen estáticamente el sistema de ficheros y cada NameNode contiene solo una parte (por ejemplo, un directorio distinto). Este método sería necesario si queremos tener muchos ficheros de tamaño medio o pequeño en vez de pocos de tamaño muy grande, ya que en este caso la MP del NameNode sería muy limitante.



C. *DataNode*

El DataNode es un nodo mucho más simple que el NameNode. Con cada bloque que se guarda en el nodo, se crean dos ficheros en su sistema de ficheros local, uno para almacenar los propios datos del bloque y otro para los metadatos (checksums) que ayudan a verificar la integridad del bloque, y si este está corrupto volver a pedir los datos a otros DataNodes.

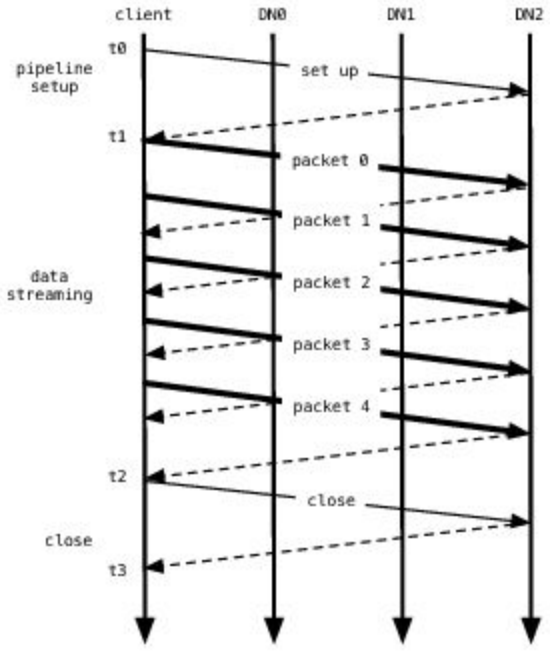
Todos los DataNodes tienen asociado un namespace ID, que es un identificador asignado al sistema de ficheros cuando éste es formateado que tienen todos los nodos del cluster. En caso de que el namespace ID del DataNode no coincida con el del NameNode, el nodo no se unirá al sistema de ficheros distribuido por motivos de integridad.

También tienen asociado un storage ID, que es asignado por el NameNode la primera vez que un DataNode se registra, y no vuelve a cambiar.

D. *Entrada/Salida del SFD*

HDFS sigue un modelo *single-writer/multiple-reader*, es decir, está más orientado a sistemas que tengan que escribir pocas veces y leer muchas.

Cuando un cliente abre un fichero para escribir, ningún otro puede escribir en él. HDFS no da ninguna garantía de que los datos que se escriben en un fichero sean visibles hasta que el cliente haya cerrado el fichero, o haya ejecutado una orden *hflush* que fuerza el envío de la caché local del bloque. Si no hay ningún error la escritura de ficheros se produce de la siguiente forma, donde la líneas continuas son los paquetes enviados y las discontinuas los ACK, se puede ver el pipeline explicado anteriormente.



Cuando un cliente abre un fichero para escribir en él, éste se bloquea y ningún otro cliente puede escribir en el fichero. Al escribir a un fichero, el cliente tiene que pedir un "candado" al NameNode, e irlo renovando cada cierto tiempo.

Para evitar que un cliente bloquee permanentemente un fichero, hay dos límites, el *soft limit* y el *hard limit*.

Hasta que no se pasa el *soft limit*, el cliente tiene acceso

exclusivo al fichero, una vez superado, otro cliente puede pedir el candado al NameNode y éste lo reservará, pero aún no se lo dará.
Cuando el tiempo del *hard limit* sea superado (por defecto 1 hora), HDFS asume que el cliente no está activo, cierra el fichero y le da el candado al siguiente cliente.

IV. HDFS vs GFS (Google File System)

A. Arquitectura

dos ficheros en su sistema de ficheros local, uno para almacenar los propios datos de HDFS y GFS tienen una arquitectura muy similar, de hecho GFS fue el SFD que originalmente implementó el algoritmo MapReduce y fué hadoop quien decidió acogerlo también y usarlo.
Podemos apreciar la misma estructura master/slave solo que google llama al nodo maestro MasterNode y a los slaves ChunkNodes, por el hecho que los trozos en los que parte los archivos son denominados chunks.
Otra diferencia seria el software que implementa esto. Para HDFS tendríamos una plataforma con diferentes SO y el uso de java para el sistema. Mientras que para google usaremos específicamente linux y el lenguaje a usar será C o C++.

En ambos tanto la comunicación, como el funcionamiento de los slaves o el ámbito donde se usan son los mismos.

B. Indexación de bloques

En HDFS la posición de los bloques de un archivo y como están guardados en memoria se guarda continuamente en la memoria del NameNode. Mientras que en google el tamaño del chunk es utilizado para calcular un índice junto con el nombre del archivo. Con ese índice será con el que haremos al request al MasterNode para pedir el archivo.

C. Integridad de los datos

En HDFS los datos que escrive un cliente genera un cheksum que se pasa por un pipe de DataNodes que lo verifican de forma que esté bien escrito en todos ellos.

En google se puede hacer de dos formas. Una es utilizando un checksum de cada ChunkNode que verifica si hay corrupción en los datos del nodo. La otra es comparando los chunks de datos entre los nodos que los poseen.

D. Permisos

HDFS utiliza el mecanismo de candados en objetos de forma que cuando un cliente quiere modificar un archivo este se bloquea. En cambio google tiene el sistema de “leases” o permisos temporales, lo que da permisos a un cliente para escribir donde él decida.

E. Balanceo de la carga de datos

GFS pone las nuevas réplicas en los ChunkNodes con el mínimo espacio de disco utilizado y el master se encarga de rebalancear las réplicas periódicamente. En HDFS es el NameNode el que se encarga de hacer todo el trabajo de balanceo manteniendo un valor medio de carga entre los nodos.

F. Estructura de ficheros

GFS divide los archivos en chunks de 64MB. Estos son replicados por defecto un total de 3 veces y tienen un identificador de 64 bits. Luego cada uno es dividido en bloques de 64Kb y cada bloque tiene un checksum.
En HDFS los bloques son de 128MB y el NameNode tiene en cuenta la réplica como 2 archivos: uno para los datos y el otro para el checksum.

V. HDFS vs CFS (Cassandra File System)

A. Arquitectura

La arquitectura en ambos sistemas es completamente diferente. HDFS está implementado con un sistema master/slave donde el NameNode trabaja como master y el DataNode como slave. En cambio e Cassandra funciona con un sistema peer-to-peer que distribuye los datos entre todos los nodos los cuales son iguales entre ellos.

B. Modelo de almacenamiento de datos

HDFS parte los datos en bloques más pequeños y los almacena copiados varias veces en los sistemas de ficheros de diferentes nodos, Mientras que Cassandra tiene un tipo de almacenamiento similar a la de las bases de datos, donde para acceder al dato necesitamos índices primario y secundario para acceder a una columna.

C. Protocolo de lectura/escritura

En HDFS al tener un nodo master que contiene los metadatos, las lecturas y escrituras son administradas por este y se hacen de forma mapreduce. En cambio cassandra tiene un sistema de lectura escritura donde leemos y escribimos en cualquier nodo.

D. Tolerancia de fallos

HDFS tiene una tolerancia de fallos en los DataNodes muy alta debido a la política de réplicas, pero es vulnerable ante la

caída del nodo master. En cambio Cassandra tiene un buen sistema de recuperación igual que HDFS pero sin tener la debilidad del nodo maestro, ya que los metadatos están en todos los nodos

E. Área donde se utiliza cada uno

HDFS es más útil cuando hacemos peticiones que requieren un montón de datos, en cambio Cassandra es más eficiente cuando hacemos queries rápidas, simples y queremos una respuesta en tiempo real.

F. Comunicación

HDFS utiliza RPC con TCP normalmente y en caso de mensajes rápidos utilizará UDP. En cambio Cassandra utiliza lo que llamamos protocolo gossip, que consiste en comunicaciones ordenador a ordenador.

G. Consistencia de los datos

En HDFS por cada bloque de un archivo calculamos los cheksums y guardamos estos checksums en el propio sistema en un archivo oculto separado.
En Cassandra separamos la consistencia de escritura de la de lectura. En la de escritura, cuando esta se realiza debemos recibir tantas señales de acknowledge como réplicas tengamos. Y en la lectura antes de que esta suceda se devuelve al cliente cuantas réplicas deberían responder.

H. Balance de datos

HDFS si un nodo contiene más réplicas de las que un valor medio entre los nodos indica, estas son movidas a otro nodo que tenga menos carga y no contenga ya esas réplicas.
En Cassandra cuando añadimos nodos nuevos no se reparte automáticamente la carga entre ellos sino que desde los nodos debemos ejecutar una comanda que mueva esta información a los nodos sin carga.

I. Identificador de archivos

HDFS utiliza los metadatos que se guardan en el NameNode.
Cassandra tiene un sistema de inodos para guardar sus metadatos.

J. Throughput y latencia

HDFS al ser un sistema con particiones de datos, la lectura puede llegar a ser inmediata o puede tardar relativamente bastante tiempo. Pero al escritura suele ser inmediata debido a la gran cantidad de DataNodes.
Cassandra al ser un sistema como una base de datos siempre tiene una respuesta de latencia y throughput excelente.

K. Mecanismo de permisos

HDFS utiliza candados en los objetos del sistema mientras, que Cassandra crea transacciones atómicas que crean consistencia durable.

L. Persistencia de los datos

HDFS escribe directamente en los DataNodes, mientras que Cassandra primero escribe los datos en memoria, en una estructura llamada mem-table y cuando está llena finalmente la escribe en disco.

VI. Comparación general entre SFD

Como no podemos comparar exhaustivamente cada SFD con HDFS a continuación presentamos una tabla que compara los conceptos generales de cada SFD con otros.

File system	GFS	KFS	Hadoop
Architecture	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based
Processes	Stateful	Stateful	Stateful
Communication	RPC/TCP	RPC/TCP	RPC/TCP&UDP
Naming	Central metadata server	Central metadata server	Central metadata server
Synchronization	Write-once-read-many, Multiple-producer/single-consumer, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases
Consistency and Replication	Server side replication, Asynchronous replication, checksum, relax consistency among replications of data objects	Server side replication, Asynchronous replication, checksum	Server side replication, Asynchronous replication, checksum
Fault tolerance	Failure as norm	Failure as norm	Failure as norm
Security	No dedicated security mechanism	No dedicated security mechanism	No dedicated security mechanism

Lustre	Panasas	PVFS2	RGFS
Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, symmetric, parallel, aggregation-based	Clustered-based, symmetric, parallel, block-based
Stateful	Stateful	Stateless	Stateful
Network Independence	RPC/TCP	RPC/TCP	RPC/TCP
Central metadata server	Central metadata server	Metadata distributed in all nodes	Metadata distributed in all nodes
Hybrid locking mechanism, using leases	Give locks on objects to clients	No locking method, no leases	Give locks on objects to clients
Server side replication – Only metadata replication, Client side caching, checksum	Server side replication – Only metadata replication	No replication, relaxed semantic for consistency	No replication
Failure as exception	Failure as exception	Failure as exception	Failure as exception
Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy

VII. Conclusiones

Después de dar un paseo por las principales problemáticas que genera extraer un sistema de ficheros local a uno distribuido conservando la transparencia. Hemos podido ver como HDFS afronta estos problemas con soluciones que son comunes con otros sistemas de ficheros, como por ejemplo el de google. Pero por el contrario también existen algunos que lo resuelven de forma completamente diferente como Cassandra. Podemos ver una relación del uso de cada sistema respecto a la forma en que resuelven los problemas. Esto es debido a que no hay soluciones perfectas ni absolutas, sino que cada solución favorece en un ámbito diferente. Es por eso que en caso de duda de que tipo de SFD usar, debemos comparar sus características para ver cual seria el adecuado para nosotros.

Por último podemos concluir que realmente HDFS es una herramienta potente para manejar Big Data debido al uso de su algoritmo MapReduce, y que es adecuado para grandes ficheros. Una de sus debilidades es que al ser un sistema de ficheros distribuido centralizado, toda el SFD depende del master (un *single point of failure*). También ofrece menos rapidez que otros SFD que a la vez actúen como sistema de ficheros locales, como Lustre, ya que HDFS no puede interactuar directamente con el disco, sino que tiene que hacerlo a través del SO y el sistema de ficheros local (ext4, FAT32… etc).

VIII. References

Libros:

[1] T. White, "Title of chapter in the book," in *Hadoop: The Definitive Guide*, *4*th ed. USA

Publicaciones:

[2] Kalpana Dwivedi and Sanjay Kumar Dubey, "A taxonomy and comparison of hadoop distributed file system with cassandra file system" *ARPN Journal of Engineering and Applied Sciences,* VOL. 10, NO. 16, SEPTEMBER 2015

[3] *Tran Doan Thanh , Subaji Mohan , Eunmi Choi , SangBum Kim , Pilsung Kim,* "A Taxonomy and Survey on Distributed File Systems" *Fourth International Conference on Networked Computing and Advanced Information Management ,* School of Business IT, Kookmin University, Seoul, Korea

[4] *D. Sathian, R. Ilamathi , R. Praveen Kumar , J. Amudhavel and P. Dhavachelvan, "A Comprehensive Survey on Taxonomy and Challenges of Distributed File Systems" Indian Journal of Science and Technology, Vol 9(11), DOI: 10.17485/ijst/2016/v9i11/89268, March 2016*

Recursos online:

[5] Robert Chansler, Hairong Kuang, Sanjay Radia, Konstantin Shvachko, and Suresh Srinivas. The Architecture of Open Source Applications Hadoop Distributed File System [Online]. Available: http://www.aosabook.org/en/hdfs.html

[6] Aditya Kulkarni. Play with the Big Yellow Elephant Toy[Online]. Available:https://www.quora.com/What-is-the-difference-between-the-Hadoop-file-distributed-system-and-the-Google-file-system