

Capítulo 5

Arquitectura Hadoop

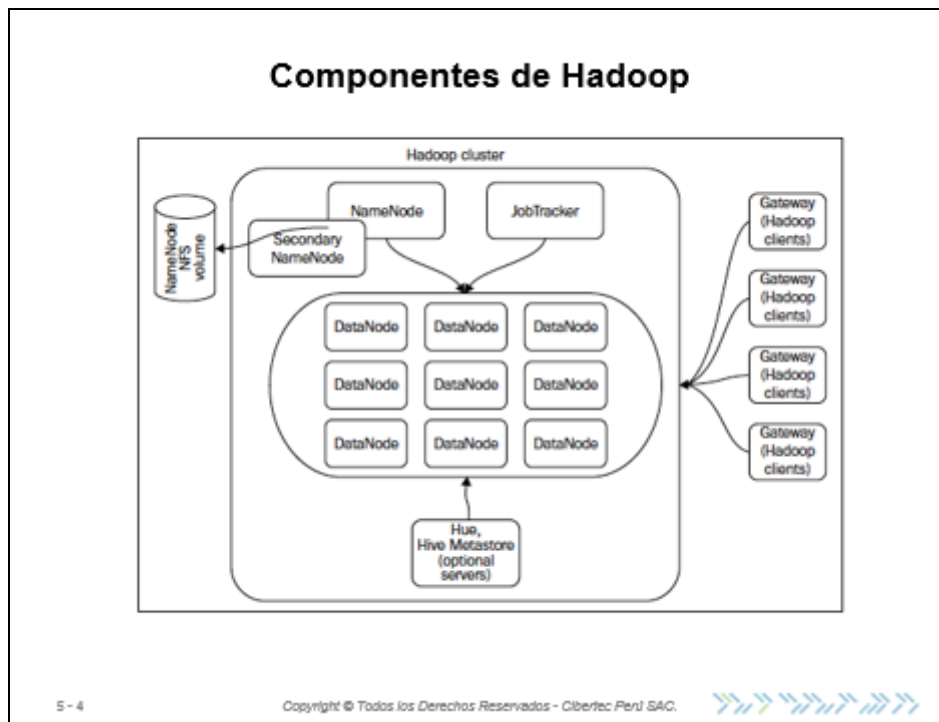
Al finalizar el capítulo, el alumno podrá:

- Comprender y desplegar la arquitectura distribuida de hadoop
- Implementar un cluster Hadoop.
- Manejar el sistema de archivos HDFS

Temas

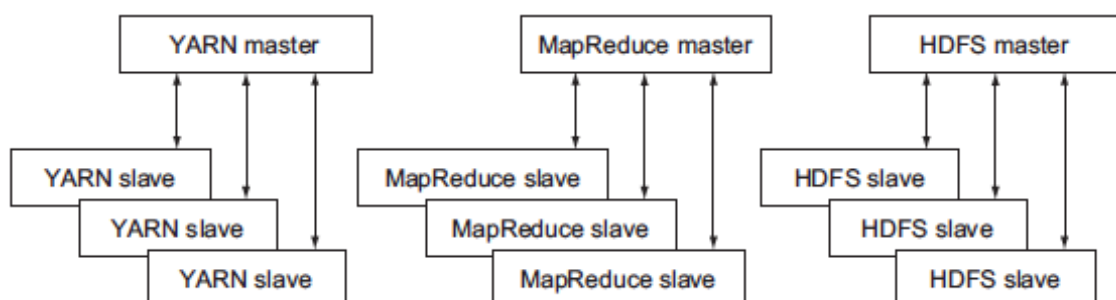
1. Componente de Hadoop
2. Implementación de un Cluster Hadoop
3. HDFS

1. Componente de Hadoop



Entre los principales componentes de Hadoop, tenemos:

- HDFS ó Hadoop Distributed File System, es un sistema de almacenamiento distribuido para almacenamiento de información estructurada o no estructurada.
- YARN ó Yet Another Resource Negotiator, incluido en la versión 2.0 de Hadoop, es un administrador de recursos mediante la calendarización de actividades.
- MapReduce es un motor de procesamiento batch, que en la versión 2.0 de Hadoop, se implementa con YARN.



Hadoop HDFS

El Sistema de archivos distribuidos de Hadoop (HDFS) es un sistema de archivos distribuidos diseñado para ejecutarse en hardware básico. Tiene muchas similitudes con los sistemas de archivos distribuidos existentes. Sin embargo, las diferencias con respecto a otros sistemas de archivos distribuidos son significativas. HDFS es altamente tolerante a fallas y está diseñado para implementarse en hardware de bajo costo. HDFS proporciona acceso de alto rendimiento a los datos de la aplicación y es adecuado para aplicaciones que tienen grandes conjuntos de datos. HDFS relaja algunos requisitos POSIX para permitir el acceso continuo a los datos del sistema de archivos. HDFS fue originalmente construido como infraestructura para el proyecto de motor de búsqueda web Apache Nutch. HDFS es parte del proyecto Apache Hadoop Core. La URL del proyecto es <http://hadoop.apache.org/>

Supuestos y metas

- **Fallo de hardware:** La falla de hardware es la norma más que la excepción. Una instancia de HDFS puede consistir en cientos o miles de máquinas servidor, cada una de las cuales almacena parte de los datos del sistema de archivos. El hecho de que haya una gran cantidad de componentes y de que cada componente tenga una probabilidad de falla no trivial significa que algún componente de HDFS siempre es no funcional. Por lo tanto, la detección de fallas y la recuperación rápida y automática de ellos es un objetivo arquitectónico central de HDFS.
- **Acceso a datos en tiempo real:** Las aplicaciones que se ejecutan en HDFS necesitan acceso de transmisión a sus conjuntos de datos. No son aplicaciones de propósito general que normalmente se ejecutan en sistemas de archivos de propósito general. HDFS está diseñado más para procesamiento por lotes en lugar de uso interactivo por los usuarios. El énfasis está en el alto rendimiento del acceso a los datos en lugar de la baja latencia del acceso a los datos. POSIX impone muchos requisitos difíciles que no son necesarios para las aplicaciones que están destinadas a HDFS. La semántica POSIX en algunas áreas clave se ha comercializado para aumentar las tasas de rendimiento de datos.
- **Grandes conjuntos de datos:** Las aplicaciones que se ejecutan en HDFS tienen grandes conjuntos de datos. Un archivo típico en HDFS es de un tamaño de gigabytes a terabytes. Por lo tanto, HDFS está sintonizado para admitir archivos de gran tamaño. Debería proporcionar un gran ancho de banda de datos agregados y escalar a cientos de nodos en un único clúster. Debe admitir decenas de millones de archivos en una sola instancia.
- **Modelo de Coherencia Simple:** Las aplicaciones HDFS necesitan un modelo de acceso de escritura para lectura de varios para los archivos. Un archivo una vez creado, escrito y cerrado no necesita ser cambiado, excepto para agregar y truncar. Se admite el agregado del contenido al final de los archivos, pero no se puede actualizar en un punto arbitrario. Esta suposición simplifica los problemas de coherencia de datos y permite el acceso a datos de alto rendimiento. Una aplicación MapReduce o una aplicación de rastreo web se ajusta perfectamente con este modelo.

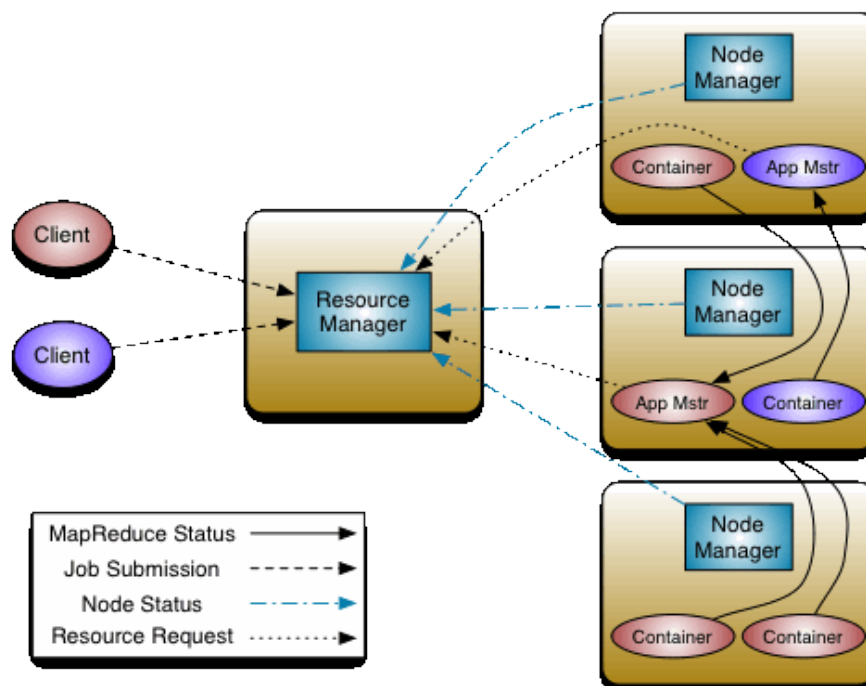
- "Mover el cálculo es más barato que mover datos": Un cómputo solicitado por una aplicación es mucho más eficiente si se ejecuta cerca de los datos en los que opera. Esto es especialmente cierto cuando el tamaño del conjunto de datos es enorme. Esto minimiza la congestión de la red y aumenta el rendimiento general del sistema. La suposición es que a menudo es mejor migrar el cálculo más cerca de donde se encuentran los datos en lugar de mover los datos a donde se ejecuta la aplicación. HDFS proporciona interfaces para que las aplicaciones se muevan más cerca del lugar donde se encuentran los datos.
- Portabilidad a través de plataformas de hardware y software heterogéneas: HDFS ha sido diseñado para ser fácilmente portátil de una plataforma a otra. Esto facilita la adopción generalizada de HDFS como plataforma de elección para un gran conjunto de aplicaciones.

Hadoop Yarn

La idea fundamental de YARN es dividir las funcionalidades de la gestión de recursos y la programación / supervisión del trabajo en demonios separados. La idea es tener un ResourceManager (RM) global y ApplicationMaster (AM) por aplicación. Una aplicación es un trabajo único o un DAG de trabajos.

ResourceManager y NodeManager forman el marco de cálculo de datos. ResourceManager es la máxima autoridad que arbitra recursos entre todas las aplicaciones en el sistema. NodeManager es el agente de marco por equipo que es responsable de los contenedores, monitoreando el uso de recursos (CPU, memoria, disco, red) y reportando lo mismo a ResourceManager / Scheduler.

El ApplicationMaster por aplicación es, en efecto, una biblioteca específica de framework y tiene la tarea de negociar recursos del ResourceManager y trabajar con los NodeManager para ejecutar y monitorear las tareas.



ResourceManager tiene dos componentes principales: Scheduler y ApplicationsManager.

El Scheduler es responsable de asignar recursos a las diversas aplicaciones en ejecución, sujeto a limitaciones familiares de capacidades, colas, etc. El Scheduler es un programador puro en el sentido de que no realiza monitoreo o seguimiento del estado de la aplicación. Además, no ofrece garantías sobre el reinicio de tareas fallidas debido a fallas de la aplicación o fallas de hardware. El Scheduler realiza su función de programación en función de los requisitos de recursos de las aplicaciones; lo hace en función de la noción abstracta de un Contenedor de recursos que incorpora elementos como memoria, CPU, disco, red, etc.

El Scheduler tiene una política conectable que se encarga de dividir los recursos del clúster entre varias colas, aplicaciones, etc. Los programadores actuales, como CapacityScheduler y FairScheduler, serían algunos ejemplos de complementos.

El ApplicationsManager es responsable de aceptar envíos de trabajos, negociar el primer contenedor para ejecutar el ApplicationMaster específico de la aplicación y proporciona el servicio para reiniciar el contenedor de ApplicationMaster en caso de falla. El ApplicationMaster por aplicación tiene la responsabilidad de negociar contenedores de recursos apropiados desde el Scheduler, hacer un seguimiento de su estado y monitorear el progreso.

MapReduce en hadoop-2.x mantiene la compatibilidad API con la versión estable anterior (hadoop-1.x). Esto significa que todos los trabajos de MapReduce aún se deben ejecutar sin cambios sobre YARN con solo una recompilación.

YARN admite la noción de reserva de recursos a través de ReservationSystem, un componente que permite a los usuarios especificar un perfil de recursos en tiempo y restricciones temporales (por ejemplo, plazos) y recursos de reserva para garantizar la ejecución predecible de trabajos importantes. The ReservationSystem rastrea los recursos. a lo largo del tiempo, realiza el control de admisión para las reservas e instruye dinámicamente al planificador subyacente para garantizar que la reserva se cumpla.

Para escalar YARN más allá de unos pocos miles de nodos, YARN admite la noción de Federación a través de la función de Federación YARN. La federación permite enlazar de forma transparente múltiples hilos (sub- clusters) y hacer que aparezcan como un único grupo masivo. Esto se puede usar para lograr una escala mayor y / o permitir que múltiples clusters independientes se utilicen en conjunto para trabajos muy grandes o para inquilinos que tienen capacidad en todos ellos.

Hadoop MapReduce

Hadoop MapReduce es un marco de software para escribir fácilmente aplicaciones que procesan grandes cantidades de datos (conjuntos de datos de varios terabytes) en paralelo en clústeres grandes (miles de nodos) de hardware básico de una manera confiable y tolerante a fallas.

Un trabajo de MapReduce generalmente divide el conjunto de datos de entrada en trozos independientes que son procesados por las tareas del mapa de una manera completamente paralela. El marco ordena los resultados de los mapas, que luego se ingresan a las tareas de reducción. Normalmente, tanto la entrada como la salida del

trabajo se almacenan en un sistema de archivos. El marco se encarga de programar las tareas, supervisarlas y volver a ejecutar las tareas fallidas.

Normalmente, los nodos de cálculo y los nodos de almacenamiento son los mismos, es decir, el marco MapReduce y el sistema de archivos distribuidos de Hadoop (consulte la Guía de arquitectura HDFS) se están ejecutando en el mismo conjunto de nodos. Esta configuración permite que el marco planifique efectivamente las tareas en los nodos donde los datos ya están presentes, lo que resulta en un ancho de banda agregado muy alto en todo el clúster.

El marco MapReduce consta de un único ResourceManager maestro, un NodeManager de trabajador por cluster-node y MRAppMaster por aplicación (ver la Guía de Arquitectura de YARN).

Como mínimo, las aplicaciones especifican las ubicaciones de entrada / salida y el mapa de suministro y reducen las funciones a través de implementaciones de interfaces apropiadas y / o clases abstractas. Estos y otros parámetros del trabajo comprenden la configuración del trabajo.

El cliente de trabajo de Hadoop luego envía el trabajo (jar / ejecutable, etc.) y la configuración al ResourceManager, que luego asume la responsabilidad de distribuir el software / configuración a los trabajadores, programar las tareas y monitorearlos, proporcionando información de estado y diagnóstico al trabajo. cliente.

Aunque el marco de Hadoop se implementa en Java TM, las aplicaciones de MapReduce no necesitan estar escritas en Java.

Hadoop Streaming es una utilidad que permite a los usuarios crear y ejecutar trabajos con cualquier ejecutable (por ejemplo, utilidades de shell) como el asignador y / o el reductor.

Hadoop Pipes es una API C ++ compatible con SWIG para implementar aplicaciones MapReduce (no basadas en JNI TM).

Entradas y salidas

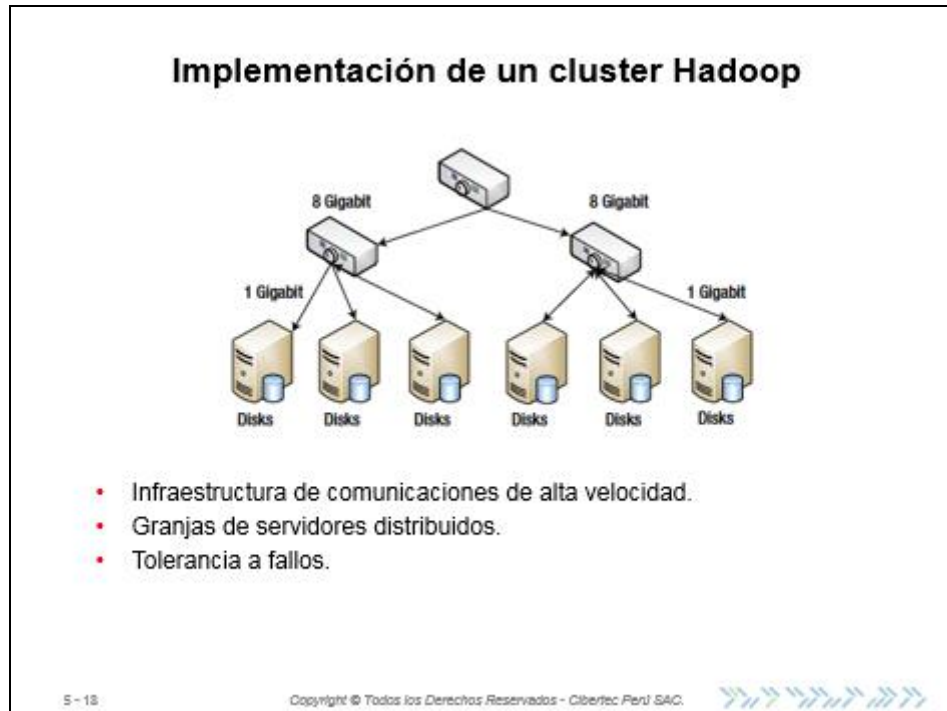
El marco MapReduce opera exclusivamente en pares de <clave, valor>, es decir, el marco visualiza la entrada al trabajo como un conjunto de pares de <clave, valor> y produce un conjunto de pares de <clave, valor> como el resultado del trabajo, concebiblemente de diferentes tipos.

Las clases clave y de valor tienen que ser serializables por el marco y, por lo tanto, deben implementar la interfaz Escribir. Además, las clases clave deben implementar la interfaz WritableComparable para facilitar la clasificación por el marco.

Tipos de entrada y salida de un trabajo de MapReduce:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** ->
<k3, v3> (output)

2. Implementación de un Cluster Hadoop




Para realizar la implementación de un cluster de Hadoop, esta será realizada en el laboratorio.

3. HDFS

HDFS

- HDFS, es un sistema de almacenamiento tolerante a fallos que puede almacenar gran cantidad de datos, escalar de forma incremental y sobrevivir a fallos de hardware sin perder datos.
- HDFS gestionar el almacenamiento en el cluster, dividiendo los ficheros en bloques y almacenando copias duplicadas a través de los nodos.
- Por defecto se replican en 3 nodos distintos.

5 - 19Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

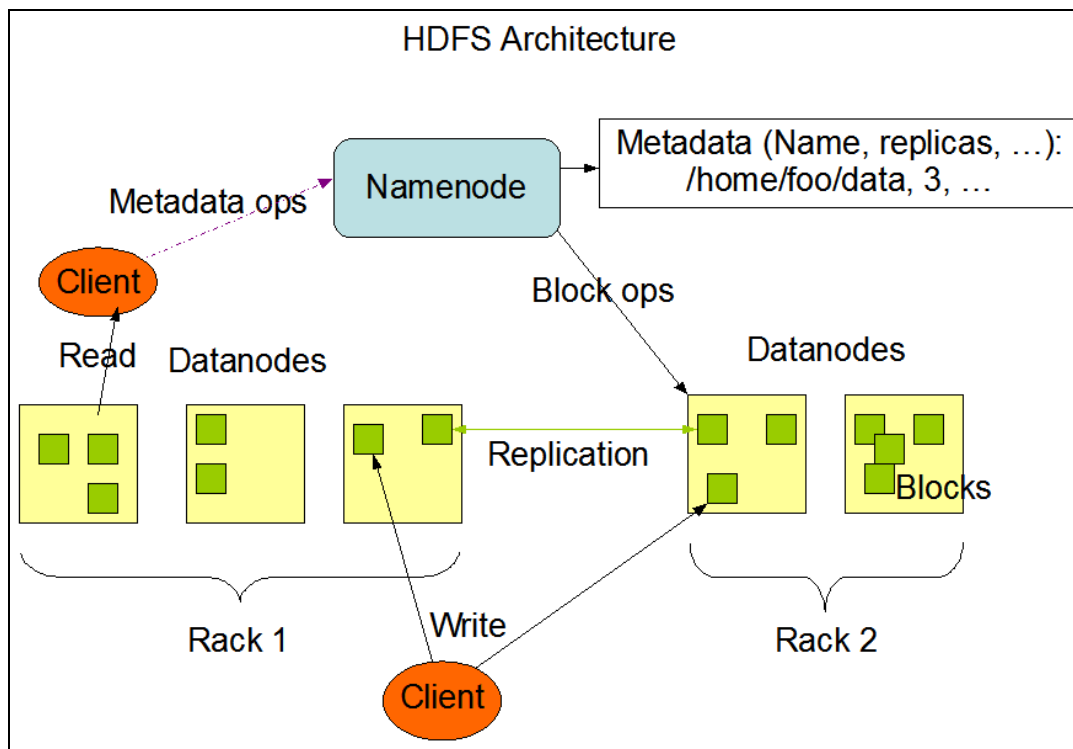
NameNode y DataNodes

HDFS tiene una arquitectura maestra / esclavo. Un clúster HDFS consta de un único NameNode, un servidor maestro que administra el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los clientes. Además, hay una cantidad de DataNodes, generalmente uno por nodo en el clúster, que administran el almacenamiento adjunto a los nodos en los que se ejecutan. HDFS expone un espacio de nombres del sistema de archivos y permite que los datos del usuario se almacenen en archivos. Internamente, un archivo se divide en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes. NameNode ejecuta operaciones del espacio de nombres del sistema de archivos como abrir, cerrar y cambiar el nombre de archivos y directorios. También determina la asignación de bloques a DataNodes. Los DataNodes son responsables de atender las solicitudes de lectura y escritura de los clientes del sistema de archivos. Los DataNodes también realizan la creación, eliminación y replicación de bloques con instrucciones del NameNode.

NameNode y DataNode son piezas de software diseñadas para ejecutarse en máquinas básicas. Estas máquinas suelen ejecutar un sistema operativo (SO) GNU / Linux. HDFS está construido usando el lenguaje Java; cualquier máquina que admita Java puede ejecutar NameNode o el software DataNode. El uso del lenguaje Java altamente portátil significa que HDFS se puede implementar en una amplia gama de máquinas. Una implementación típica tiene una máquina dedicada que ejecuta solo el software NameNode. Cada una de las otras máquinas en el clúster ejecuta una instancia del software DataNode. La arquitectura no impide ejecutar múltiples

DataNodes en la misma máquina, sino en una implementación real que rara vez es el caso.

La existencia de un NameNode único en un clúster simplifica enormemente la arquitectura del sistema. NameNode es el árbitro y repositorio de todos los metadatos HDFS. El sistema está diseñado de tal manera que los datos del usuario nunca fluyan a través del NameNode.



El espacio de nombres del sistema de archivos

HDFS admite una organización de archivos jerárquica tradicional. Un usuario o una aplicación pueden crear directorios y almacenar archivos dentro de estos directorios. La jerarquía del espacio de nombres del sistema de archivos es similar a la mayoría de los demás sistemas de archivos existentes; uno puede crear y eliminar archivos, mover un archivo de un directorio a otro o renombrar un archivo. HDFS admite cuotas de usuario y permisos de acceso. HDFS no es compatible con enlaces duros o enlaces blandos. Sin embargo, la arquitectura HDFS no impide la implementación de estas características.

NameNode mantiene el espacio de nombres del sistema de archivos. Cualquier cambio en el espacio de nombres del sistema de archivos o sus propiedades es registrado por NameNode. Una aplicación puede especificar el número de réplicas de un archivo que HDFS debe mantener. La cantidad de copias de un archivo se llama factor de replicación de ese archivo. Esta información es almacenada por NameNode.

Replicación de datos

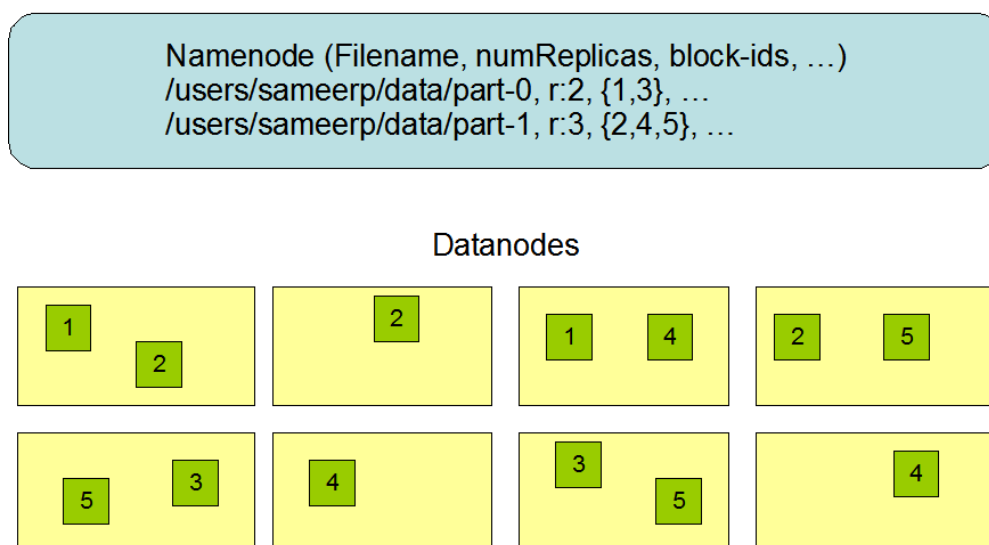
HDFS está diseñado para almacenar de manera confiable archivos muy grandes en máquinas de un gran clúster. Almacena cada archivo como una secuencia de bloques. Los bloques de un archivo se replican para la tolerancia a fallas. El tamaño del bloque y el factor de replicación son configurables por archivo.

Todos los bloques en un archivo excepto el último bloque tienen el mismo tamaño, mientras que los usuarios pueden comenzar un nuevo bloque sin completar el último bloque al tamaño de bloque configurado después de que se agregó el soporte para bloque de longitud variable para append y hsync.

Una aplicación puede especificar el número de réplicas de un archivo. El factor de replicación se puede especificar en el momento de creación del archivo y se puede cambiar más adelante. Los archivos en HDFS son de escritura única (excepto para anexos y truncados) y tienen estrictamente un escritor en cualquier momento.

NameNode toma todas las decisiones con respecto a la replicación de bloques. Recibe periódicamente un Heartbeat y un Blockreport de cada uno de los DataNodes en el clúster. La recepción de un latido significa que el DataNode funciona correctamente. Un informe de bloque contiene una lista de todos los bloques en un nodo de datos.

Block Replication



- **Ubicación de réplica: Primeros pasos.**

La colocación de réplicas es crítica para la fiabilidad y el rendimiento de HDFS. La optimización de la ubicación de las réplicas distingue a HDFS de la mayoría de los demás sistemas de archivos distribuidos. Esta es una característica que necesita mucha afinación y experiencia. El propósito de una política de ubicación de réplicas con reconocimiento de bastidor es mejorar la confiabilidad de los datos, la disponibilidad y la utilización del ancho de banda de la red. La implementación actual de la política de ubicación de réplicas es un primer esfuerzo en esta dirección. Los objetivos a corto plazo de implementar esta política son validarla en sistemas de producción, aprender más sobre su comportamiento y construir una base para probar e investigar políticas más sofisticadas.

Las instancias grandes de HDFS se ejecutan en un grupo de computadoras que comúnmente se extienden a través de muchos racks. La comunicación entre dos nodos en diferentes racks tiene que pasar por switches. En la mayoría de los casos, el ancho de banda de red entre máquinas en el mismo rack es mayor que el ancho de banda de red entre máquinas en diferentes racks.

NameNode determina la id del rack a la que pertenece cada DataNode a través del proceso descrito en Hadoop Rack Awareness. Una política simple pero no óptima es colocar réplicas en únicos racks. Esto evita la pérdida de datos cuando falla todo un rack y permite el uso de ancho de banda de múltiples racks al leer datos. Esta política distribuye de manera uniforme las réplicas en el clúster, lo que facilita el equilibrio de la carga en la falla del componente. Sin embargo, esta política aumenta el costo de las escrituras porque una escritura necesita transferir bloques a múltiples racks.

Para el caso común, cuando el factor de replicación es tres, la política de colocación de HDFS es poner una réplica en la máquina local si el escritor está en un nodo de datos, de lo contrario en un nodo de datos aleatorio, otra réplica en un nodo en un estante diferente (remoto) y el último en un nodo diferente en el mismo rack remoto. Esta política reduce el tráfico de escritura entre racks, lo que generalmente mejora el rendimiento de escritura. La posibilidad de falla de rack es mucho menor que la falla de nodo; esta política no afecta las garantías de fiabilidad y disponibilidad de los datos. Sin embargo, sí reduce el ancho de banda de red agregado que se utiliza al leer datos, ya que un bloque se coloca en solo dos racks únicos en lugar de tres. Con esta política, las réplicas de un archivo no se distribuyen uniformemente en los racks. Un tercio de las réplicas están en un nodo, dos tercios de las réplicas están en un rack y el otro tercio están distribuidas uniformemente en los racks restantes. Esta política mejora el rendimiento de escritura sin comprometer la confiabilidad de los datos o el rendimiento de lectura.

Si el factor de replicación es mayor que 3, la ubicación de la 4ta y siguientes réplicas se determina aleatoriamente mientras se mantiene el número de réplicas por rack por debajo del límite superior (que es básicamente $(\text{réplicas} - 1) / \text{racks} + 2$).

Como NameNode no permite que DataNodes tenga varias réplicas del mismo bloque, la cantidad máxima de réplicas creadas es la cantidad total de DataNodes en ese momento.

Una vez que se agregó el soporte para los Tipos de Almacenamiento y las Políticas de Almacenamiento a HDFS, NameNode toma en cuenta la política para la ubicación de la réplica, además de la conciencia de rack descrita anteriormente. El NameNode elige nodos basados en la conciencia de rack al principio, luego verifica que el nodo candidato tenga el almacenamiento requerido por la política asociada con el archivo. Si el nodo candidato no tiene el tipo de almacenamiento, NameNode busca otro nodo. Si no se pueden encontrar suficientes nodos para colocar réplicas en la primera ruta, NameNode busca nodos que tengan tipos de almacenamiento de reserva en la segunda ruta.

La política actual de colocación de réplicas descrita aquí es un trabajo en progreso.

- **Selección de réplica**

Para minimizar el consumo de ancho de banda global y la latencia de lectura, HDFS intenta satisfacer una solicitud de lectura de una réplica más cercana al lector. Si existe una réplica en el mismo bastidor que el nodo lector, entonces se prefiere esa réplica para satisfacer la solicitud de lectura. Si el clúster HDFS

abarca varios centros de datos, se prefiere una réplica residente en el centro de datos local sobre cualquier réplica remota.

- **Modo seguro**

Al inicio, NameNode ingresa a un estado especial llamado Safemode. La replicación de bloques de datos no ocurre cuando el NameNode está en el estado Safemode. NameNode recibe mensajes Heartbeat y Blockreport de los DataNodes. Un informe de bloqueo contiene la lista de bloques de datos que aloja un nodo de datos. Cada bloque tiene un número mínimo especificado de réplicas. Se considera que un bloque se ha replicado con seguridad cuando el número mínimo de réplicas de ese bloque de datos se ha registrado con NameNode. Después de que un porcentaje configurable de bloques de datos replicados con seguridad ingresa con NameNode (más 30 segundos adicionales), NameNode sale del estado de Safemode. A continuación, determina la lista de bloques de datos (si corresponde) que aún tienen menos que el número especificado de réplicas. El NameNode luego replica estos bloques a otros DataNodes.

La persistencia de los metadatos del sistema de archivos

El espacio de nombre HDFS es almacenado por NameNode. NameNode utiliza un registro de transacciones llamado EditLog para registrar de forma persistente cada cambio que se produce en los metadatos del sistema de archivos. Por ejemplo, crear un nuevo archivo en HDFS hace que NameNode inserte un registro en EditLog que indica esto. De forma similar, al cambiar el factor de replicación de un archivo, se inserta un nuevo registro en EditLog. NameNode utiliza un archivo en su sistema de archivos local OS host para almacenar EditLog. Todo el espacio de nombres del sistema de archivos, incluida la asignación de bloques a los archivos y las propiedades del sistema de archivos, se almacena en un archivo llamado FsImage. FsImage también se almacena como un archivo en el sistema de archivos local de NameNode.

NameNode mantiene una imagen del espacio de nombres completo del sistema de archivos y el archivo Blockmap en la memoria. Cuando el NameNode se inicia o un punto de control se activa mediante un umbral configurable, lee FsImage y EditLog desde el disco, aplica todas las transacciones de EditLog a la representación en memoria de FsImage y vacía esta nueva versión en un nueva FsImage en el disco. A continuación, puede truncar el antiguo EditLog porque sus transacciones se han aplicado a la FsImage persistente. Este proceso se llama punto de control. El propósito de un punto de control es asegurarse de que HDFS tenga una vista coherente de los metadatos del sistema de archivos al tomar una instantánea de los metadatos del sistema de archivos y guardarlos en FsImage. Aunque es eficiente leer una imagen Fs, no es eficiente realizar ediciones incrementales directamente a una imagen Fs. En lugar de modificar FsImage para cada edición, persistimos las ediciones en el Registro. Durante el punto de control, los cambios de Registro se aplican a FsImage. Se puede activar un punto de control en un intervalo de tiempo dado (`dfs.namenode.checkpoint.period`) expresado en segundos, o después de que se haya acumulado una cantidad determinada de transacciones del sistema de archivos (`dfs.namenode.checkpoint.txns`). Si se establecen estas dos propiedades, el primer umbral que se alcanzará desencadena un punto de control.

DataNode almacena datos HDFS en archivos en su sistema de archivos local. DataNode no tiene conocimiento sobre los archivos HDFS. Almacena cada bloque de

datos HDFS en un archivo separado en su sistema de archivos local. DataNode no crea todos los archivos en el mismo directorio. En cambio, usa una heurística para determinar la cantidad óptima de archivos por directorio y crea subdirectorios de manera apropiada. No es óptimo crear todos los archivos locales en el mismo directorio porque el sistema de archivos local podría no ser capaz de admitir de manera eficiente una gran cantidad de archivos en un único directorio. Cuando un DataNode se inicia, escanea a través de su sistema de archivos local, genera una lista de todos los bloques de datos HDFS que corresponden a cada uno de estos archivos locales, y envía este informe al NameNode. El informe se llama Informe de bloque.

Los protocolos de comunicación

Todos los protocolos de comunicación HDFS están superpuestos sobre el protocolo TCP / IP. Un cliente establece una conexión a un puerto TCP configurable en la máquina NameNode. Habla ClientProtocol con NameNode. Los DataNodes hablan con el NameNode utilizando el protocolo DataNode. Una abstracción de llamada a procedimiento remoto (RPC) ajusta el protocolo de cliente y el protocolo de nodo de datos. Por diseño, NameNode nunca inicia ningún RPC. En cambio, solo responde a las solicitudes RPC emitidas por DataNodes o clientes.

Robustez

El objetivo principal de HDFS es almacenar datos de manera confiable incluso en presencia de fallas. Los tres tipos comunes de fallas son las fallas de NameNode, las fallas de DataNode y las particiones de red.

- **Falla de Disco de Datos, Heartbeats y Re-Replicación**

Cada DataNode envía un mensaje Heartbeat al NameNode periódicamente. Una partición de red puede hacer que un subconjunto de DataNodes pierda conectividad con NameNode. NameNode detecta esta condición por la ausencia de un mensaje Heartbeat. NameNode marca DataNodes sin Heartbeats recientes como muertos y no reenvía ninguna nueva solicitud de IO a ellos. Cualquier dato que haya sido registrado en un DataNode muerto ya no está disponible para HDFS. La muerte de DataNode puede causar que el factor de replicación de algunos bloques caiga por debajo de su valor especificado. NameNode rastrea constantemente qué bloques deben replicarse e inicia la replicación siempre que sea necesario. La necesidad de volver a replicar puede surgir debido a varias razones: un DataNode puede no estar disponible, una réplica puede dañarse, un disco duro en un DataNode puede fallar o puede aumentar el factor de replicación de un archivo.

El tiempo de espera para marcar Dead DataNodes es conservativamente largo (más de 10 minutos por defecto) para evitar la tormenta de replicación causada por el aleteo de DataNodes. Los usuarios pueden establecer intervalos más cortos para marcar DataNodes como obsoletos y evitar nodos obsoletos al leer y / o escribir por configuración para cargas de trabajo sensibles al rendimiento.

- **Rebalanceo de Cluster**

La arquitectura HDFS es compatible con esquemas de rebalanceo de datos. Un esquema puede mover automáticamente los datos de un DataNode a otro si el espacio libre en un DataNode cae por debajo de un cierto umbral. En el caso de una gran demanda repentina de un archivo en particular, un esquema podría crear dinámicamente réplicas adicionales y reequilibrar otros datos en el clúster. Estos tipos de esquemas de reequilibrio de datos aún no se han implementado.

- **Integridad de los datos**

Es posible que un bloque de datos recuperados de un DataNode llegue dañado. Esta corrupción puede ocurrir debido a fallas en un dispositivo de almacenamiento, fallas de red o software defectuoso. El software de cliente HDFS implementa verificación de suma de comprobación en los contenidos de los archivos HDFS. Cuando un cliente crea un archivo HDFS, calcula una suma de comprobación de cada bloque del archivo y almacena estas sumas de comprobación en un archivo oculto separado en el mismo espacio de nombre HDFS. Cuando un cliente recupera el contenido del archivo, verifica que los datos que recibió de cada DataNode coincidan con la suma de comprobación almacenada en el archivo de suma de comprobación asociado. De lo contrario, el cliente puede optar por recuperar ese bloque de otro DataNode que tenga una réplica de ese bloque.

- **Error de disco de metadatos**

FsImage y EditLog son estructuras de datos centrales de HDFS. Una corrupción de estos archivos puede causar que la instancia de HDFS no sea funcional. Por este motivo, NameNode se puede configurar para admitir el mantenimiento de varias copias de FsImage y EditLog. Cualquier actualización de FsImage o EditLog hace que cada una de las FsImages y EditLogs se actualicen de forma sincronizada. Esta actualización sincrónica de varias copias de FsImage y EditLog puede degradar la tasa de transacciones de espacio de nombres por segundo que un NameNode puede admitir. Sin embargo, esta degradación es aceptable porque a pesar de que las aplicaciones HDFS son muy intensivas en datos por naturaleza, no son intensivas en metadatos. Cuando un NameNode se reinicia, selecciona la última FsImage y EditLog consistentes para usar.

Otra opción para aumentar la capacidad de recuperación contra fallas es habilitar Alta disponibilidad utilizando múltiples NameNodes con un almacenamiento compartido en NFS o usando un registro de edición distribuida (llamado Diario). Este último es el enfoque recomendado.

- **Snapshots**

Las instantáneas permiten almacenar una copia de datos en un instante determinado. Un uso de la función de instantánea puede ser retrotraer una instancia de HDFS dañada a un punto bueno conocido previamente en el tiempo.

Organización de datos

- **Bloques de datos**

HDFS está diseñado para admitir archivos muy grandes. Las aplicaciones que son compatibles con HDFS son aquellas que manejan grandes conjuntos de datos. Estas aplicaciones escriben sus datos solo una vez, pero lo leen una o más veces y requieren que estas lecturas se cumplan a velocidades de transmisión. HDFS admite la semántica escritura-una-vez-lectura-muchos en los archivos. Un tamaño de bloque típico utilizado por HDFS es 128 MB. Por lo tanto, un archivo HDFS se fragmenta en trozos de 128 MB y, si es posible, cada trozo residirá en un DataNode diferente.

- **Duplicación de replicación**

Cuando un cliente escribe datos en un archivo HDFS con un factor de replicación de tres, NameNode recupera una lista de DataNodes utilizando un

algoritmo de selección de destino de replicación. Esta lista contiene los DataNodes que alojarán una réplica de ese bloque. El cliente luego escribe en el primer DataNode. El primer DataNode comienza a recibir los datos en porciones, escribe cada parte en su repositorio local y transfiere esa parte al segundo nodo de datos en la lista. El segundo DataNode, a su vez comienza a recibir cada parte del bloque de datos, escribe esa parte en su repositorio y luego vacía esa porción al tercer DataNode. Finalmente, el tercer DataNode escribe los datos en su repositorio local. Por lo tanto, un DataNode puede estar recibiendo datos del anterior en el pipeline y al mismo tiempo reenviando datos al siguiente en la pipeline. Por lo tanto, los datos se canalizan desde un DataNode al siguiente.

Accesibilidad

Se puede acceder a HDFS desde las aplicaciones de diferentes maneras. Nativamente, HDFS proporciona una API de FileSystem Java para que las aplicaciones la utilicen. También está disponible un contenedor de lenguaje C para esta API Java y API REST. Además, un navegador HTTP y también se puede utilizar para navegar por los archivos de una instancia de HDFS. Mediante el uso de la puerta de enlace NFS, HDFS se puede montar como parte del sistema de archivos local del cliente.

- **FS Shell**
HDFS permite que los datos del usuario se organicen en forma de archivos y directorios. Proporciona una interfaz de línea de comando llamada FS shell que permite al usuario interactuar con los datos en HDFS.
- **DFSAdmin**
El conjunto de comandos DFSAdmin se usa para administrar un clúster HDFS.
- **Interfaz del navegador**
Una instalación HDFS típica configura un servidor web para exponer el espacio de nombre HDFS a través de un puerto TCP configurable. Esto le permite al usuario navegar por el espacio de nombre HDFS y ver el contenido de sus archivos usando un navegador web.

Recuperación

- **Recuperación de archivos eliminados**
Si la configuración de la papelera está habilitada, los archivos eliminados por FS Shell no se eliminan inmediatamente de HDFS. En cambio, HDFS lo mueve a un directorio de papelera (cada usuario tiene su propio directorio de papelera en /user/<username>/.Trash). El archivo se puede restaurar rápidamente siempre que permanezca en la papelera.
- **Disminuir el factor de replicación**
Cuando se reduce el factor de replicación de un archivo, NameNode selecciona el exceso de réplicas que se pueden eliminar. El siguiente Heartbeat transfiere esta información al DataNode. El DataNode luego elimina los bloques correspondientes y el espacio libre correspondiente aparece en el clúster. Una vez más, puede haber un retraso de tiempo entre la finalización de la llamada a la API setReplication y la aparición de espacio libre en el clúster.