Esteban Vergara Giraldo

Jonathan Betancur Espinosa

Miguel Ángel Cock Cano

Introducción

- **1.1 Definición de un Sistema Distribuido**
  - Un sistema distribuido es un modelo en el cual componentes localizados en computadoras conectadas en red se comunican y coordinan sus acciones mediante el intercambio de mensajes actuando a manera del usuario final, de entregar un único resultado transparente. El objetivo es permitir la compartición de recursos asegurando, al mismo tiempo, la tolerancia a fallos, la escalabilidad y la optimización del rendimiento. Los sistemas distribuidos soportan aplicaciones a gran escala distribuyendo tareas de almacenamiento de datos y procesamiento entre múltiples nodos, asegurando eficiencia y confiabilidad.
- **1.2 Propósito de los Sistemas Distribuidos**
  - Los sistemas distribuidos están diseñados para manejar grandes cantidades de datos y tareas de procesamiento distribuyéndolos entre varios servidores. Mejoran el rendimiento del sistema, proporcionan redundancia y permiten el procesamiento en paralelo. Ejemplos de uso común incluyen sistemas de archivos distribuidos como HDFS y GFS, computación en la nube y análisis de datos a gran escala.
- **1.3 Ventajas de los Sistemas Distribuidos**
  - **Tolerancia a fallos:** Son capaces de manejar fallos de nodos individuales sin que afecte al sistema en su conjunto, gracias a la replicación de datos en varios nodos.
  - **Escalabilidad:** Pueden crecer fácilmente añadiendo más nodos.
  - **Eficiencia:** Las tareas se ejecutan más cerca de donde se encuentran los datos, reduciendo la latencia y aumentando el rendimiento.

- **1.4 Desventajas de los Sistemas Distribuidos**
  - **Complejidad de implementación:** Los sistemas distribuidos son más difíciles de diseñar, construir, depurar y mantener sin encontrarse múltiples errores de conexión.
  - **Mantenibilidad:** Si no se tiene un manejo de errores en caso de pérdida de un nodo, o de un archivo, o de una instancia, encontrar el causante del error puede ser complejo de encontrar

2. Descripción del Sistema de Archivos Distribuido de Hadoop (HDFS)

- **2.1 Arquitectura y Componentes**
  - HDFS es un sistema de archivos altamente confiable y escalable, diseñado para almacenar datos distribuidos en grandes clústeres de hardware de bajo costo. Divide los datos en grandes bloques (típicamente de 128 MB) y replica estos bloques en varios nodos para mejorar la fiabilidad y el rendimiento.
  - **NameNode**: Administra el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los clientes. Mantiene los metadatos y coordina la distribución de datos entre los DataNodes.
  - **DataNodes**: Almacenan los bloques de datos reales. Cada DataNode informa periódicamente al NameNode y envía señales de vida (heartbeats) para indicar su disponibilidad.
  - **Clientes HDFS**: Acceden a los datos a través de una API, interactuando con el NameNode para recuperar metadatos y con los DataNodes para obtener los datos reales.
- **2.2 Manejo de Datos y Comunicación**
  - **Operaciones de Lectura y Escritura**: Los clientes de HDFS contactan primero al NameNode para identificar qué DataNodes almacenan los datos necesarios. Los datos se escriben de manera canalizada en múltiples DataNodes para asegurar un alto rendimiento.
  - **Replicación de Datos**: Los bloques de datos se replican en varios DataNodes (tres réplicas) para garantizar la tolerancia a fallos.
  - **Mecanismo de Heartbeat**: Los DataNodes envían señales de vida al NameNode a intervalos regulares. Si el NameNode no recibe una señal en un plazo de 10 minutos, marca el DataNode como no disponible.
  - **Checkpoint y Nodos de Respaldo**: Periódicamente se crea un punto de control (checkpoint) para proteger los metadatos del sistema de archivos.
- **2.3 Comunicación Entre Componentes**
  - **NameNode (Líder) y DataNodes**: El NameNode gestiona los metadatos, mientras que los DataNodes almacenan los datos. La comunicación es mayormente unidireccional, donde los DataNodes informan su estado mediante heartbeats y reciben comandos de replicación.
  - **Clientes**: Los clientes interactúan directamente con el NameNode para recuperar metadatos y con los DataNodes para leer y escribir los datos reales.

3. Descripción del Sistema de Archivos de Google (GFS)

- **3.1 Arquitectura y Componentes**
  - GFS es un sistema de archivos distribuido diseñado para gestionar grandes conjuntos de datos en miles de máquinas utilizando hardware de bajo costo. Su arquitectura es simple pero efectiva, proporcionando tolerancia a fallos y alta disponibilidad a través de la replicación
  - **Máster**: GFS utiliza un único maestro para mantener los metadatos, gestionar las solicitudes de los clientes y coordinar los chunkservers
  - **Chunkservers**: Almacenan fragmentos de archivos de tamaño fijo (chunks). Cada chunk se replica en varios chunkservers (generalmente tres réplicas) para garantizar la fiabilidad.
  - **Clientes**: Los clientes interactúan con el master para obtener metadatos y con los chunkservers para obtener los datos del archivo.
- **3.2 Manejo de Datos y Comunicación**
  - **Operaciones de Archivos**: Similar a HDFS, los clientes de GFS contactan primero al máster para recuperar metadatos y luego se comunican directamente con los chunkservers para obtener los datos del archivo.
  - **Replicación de Datos**: Los chunks se replican en varios chunkservers para garantizar la disponibilidad de los datos, incluso si fallan algunos nodos.
  - **Recuperación ante Fallos**: GFS asume que los fallos de nodos son comunes y utiliza una supervisión constante y replicación para recuperarse rápidamente de los errores.
- **3.3 Comunicación Entre Componentes**
  - **Máster (Líder) y Chunkservers (Seguidores)**: El máster controla los metadatos y coordina la ubicación de los chunks. Los chunkservers almacenan los chunks de datos y envían reportes de estado al máster mediante mensajes de Heartbeat.
  - **Clientes**: Los clientes se comunican con el máster para localizar los chunks y luego con los chunkservers para obtener los datos reales

4. Comparación entre HDFS y GFS

- **4.1 Diferencias Arquitectónicas**
  - HDFS utiliza NameNode y DataNodes, mientras que GFS utiliza un único máster y chunkservers.
  - GFS utiliza chunks más grandes (64MB) en comparación con HDFS, lo que optimiza para archivos grandes y lecturas secuenciales.
- **4.2 Comunicación y Tolerancia a Fallos**

- Ambos sistemas priorizan la tolerancia a fallos, utilizando la replicación y mecanismos de heartbeat regulares para monitorear y gestionar la disponibilidad de nodos.
- GFS utiliza mecanismos de recuperación automática para manejar fallos, mientras que HDFS utiliza un sistema más complejo de checkpoint y journaling para asegurar la integridad de los metadatos.

5. Especificaciones del sistema

- **5.1 Arquitectura del sistema**

Nuestro sistema de archivos distribuido se diseñó con un enfoque simple y basado en RAM. Se compone de tres elementos principales: el NameNode, los DataNodes y los Clientes, cada uno con un rol específico en el sistema. A continuación, se describe cómo se gestionan estos componentes y cómo interactúan entre sí.

- **NameNode:** El NameNode es el componente central de nuestro sistema. Es responsable de gestionar toda la información sobre los archivos y su distribución en los DataNodes. Esta información se almacena en RAM utilizando un diccionario de listas, donde las claves son las IPs de los DataNodes y los valores son listas de archivos que están almacenados en cada uno de ellos. No se implementa ningún mecanismo de recuperación en caso de fallo del NameNode. Si el NameNode se cae, toda la información almacenada se pierde y el sistema debe reiniciarse. No hay un NameNode secundario, y el NameNode es considerado un punto crítico en la arquitectura.

- **DataNodes:** Los DataNodes se encargan de almacenar los bloques de los archivos en listas enlazadas. Cada archivo se divide en bloques, y estos bloques se distribuyen en varios DataNodes. Cada DataNode puede almacenar una lista de bloques, donde cada ítem tiene un tamaño máximo establecido de 1 MB. Si un bloque falla, se considera que el archivo está corrupto, y no se recupera. Sin embargo, debido a que cada archivo debe estar replicado en al menos tres DataNodes, el sistema intentará replicar el archivo fallido en otro nodo si detecta la corrupción.

- **Clientes (API y CLI):** Los clientes interactúan con el sistema a través de un menú de consola, donde pueden ejecutar diferentes comandos como cargar, descargar o listar archivos. Estos comandos se traducen en acciones que se comunican con el NameNode utilizando API REST. En caso de operaciones relacionadas con archivos, como subir o descargar, el NameNode primero consulta su diccionario para determinar dónde están los bloques, y luego el

cliente se comunica con los DataNodes correspondientes usando gRPC para transferir los bloques.

A continuación, también se tienen algunas especificaciones sobre mecanismos de elección, replicación y autenticación plasmados en el funcionamiento del sistema:

Replicación y selección de nodos: Cuando un archivo se guarda en el sistema, el NameNode se encarga de replicarlo en tres nodos diferentes. La selección de estos nodos se hace de forma aleatoria, asegurando que el nodo original no sea uno de los replicados, y que tampoco se elija un nodo que ya tenga el archivo.

- **Mecanismo de autenticación:** Antes de interactuar con el sistema, los clientes deben iniciar sesión con un usuario y una contraseña. Todas las sesiones se gestionan a través del NameNode, lo que significa que cualquier acción que un cliente quiera realizar (como subir un archivo o listar los archivos) primero pasa por el NameNode para verificar la sesión.

- **Manejo de señales de vida (heartbeat):** Para asegurar que los nodos del sistema están activos, se implementa un mecanismo de "ping-pong" basado en API REST. Cada componente envía señales de vida cada 5 segundos para asegurarse de que los nodos siguen activos. Si un nodo no responde a una señal de vida, se marca como inactivo y se deja de utilizar para futuras operaciones.

- **Punto de fallo y recuperación:** El sistema no tiene mecanismos para recuperar datos si un nodo falla. Toda la información se maneja en RAM utilizando estructuras simples como listas enlazadas, diccionarios y composiciones de estas. Por lo tanto, si un nodo se cae, cualquier dato que estuviera en RAM se pierde.

- **5.2 Protocolos de comunicación entre los diferentes componentes del sistema**

En nuestro sistema de archivos distribuido, decidimos usar una combinación de API REST y gRPC para la comunicación entre los diferentes componentes. La idea es que REST maneje las operaciones generales (como pedir metadatos o listar archivos), y gRPC se encargue de la transferencia de archivos, ya que es más eficiente para mover grandes cantidades de datos.

- **Cliente <-> NameNode:** Cuando el cliente necesita hacer algo como ver la lista de archivos o saber dónde están almacenados los bloques de un archivo, se comunica con el NameNode usando API REST. Es decir, se mandan

peticiones HTTP y el NameNode responde con la información necesaria, generalmente en formato JSON, para que el cliente pueda saber en qué DataNodes están los bloques de datos que necesita.

- **Cliente <-> DataNode:** Para subir o descargar un archivo, el cliente se conecta directamente a los DataNodes usando gRPC. Esto es porque gRPC es mucho más rápido para transferir archivos grandes. Básicamente, el archivo se divide en bloques, y gRPC se encarga de enviarlos o recibirlos de manera eficiente desde los diferentes DataNodes.

- **NameNode <-> DataNode:** El NameNode se encarga de decirle a los DataNodes qué hacer (por ejemplo, replicar bloques, enviar señales de vida, etc.) y todo esto lo hace a través de API REST. Es una forma fácil de coordinar todo el sistema, y REST nos permite mantener todo más simple cuando no estamos transfiriendo datos pesados.

- **DataNode <-> DataNode:** Cuando un bloque necesita ser replicado de un DataNode a otro, usamos gRPC para la transferencia. Esto asegura que los bloques se muevan rápido entre nodos, garantizando que los datos estén siempre disponibles, incluso si uno de los DataNodes falla.

Así, combinamos lo mejor de ambos mundos: REST para lo más administrativo y el control, y gRPC para mover archivos de forma rápida y eficiente.