TEC | Tecnológico de Costa Rica

Tercer Proyecto

Escuela de Ingeniería en Computadores

Algoritmos y Estructuras de Datos II

Profesor:

Leonardo Araya

Estudiantes:

Esteban Campos Cerdas | 202207705
Jafet Díaz Morales | 2023053249
Steven Pérez Aguilar | 2024118003
Paula Melina Porras Mora | 2023082886

I Semestre, 2025.

Tabla de contenido

Introducción

Este documento presenta la documentación del proyecto TEC Media File System, desarrollado como parte del curso Algoritmos y Estructuras de Datos II del Instituto Tecnológico de Costa Rica. El objetivo principal del proyecto es diseñar e implementar un sistema de archivos distribuido bajo un esquema RAID 5, aplicando los conocimientos adquiridos en el curso, en especial aquellos relacionados con estructuras de datos, organización de archivos y computación distribuida.

En esta documentación se detallan los aspectos clave del desarrollo, incluyendo una descripción del problema y la solución propuesta. Se explican las decisiones de diseño e implementación tomadas para cumplir con cada uno de los requerimientos funcionales, así como las herramientas y tecnologías utilizadas. Además, se discuten las alternativas consideradas durante el proceso de desarrollo, las limitaciones identificadas y los principales desafíos enfrentados.

También se incluye una descripción de la arquitectura del sistema, un resumen de la comunicación entre componentes mediante HTTP, y se presenta una interfaz gráfica diseñada para facilitar la interacción con el sistema. Finalmente, se proporciona un enlace al repositorio de GitHub donde se encuentra el código fuente del proyecto.

Descripción del problema

El proyecto TEC Media File System desarrolla un sistema de archivos distribuido capaz de almacenar, recuperar y gestionar documentos de forma eficiente, utilizando un esquema de RAID 5 y una arquitectura distribuida conocida como Shared Disk Architecture. El sistema está conformado por múltiples nodos de disco (Disk Nodes) y un nodo controlador (Controller Node) encargado de coordinar la distribución, lectura y recuperación de los datos.

Cada archivo que se desea almacenar se divide en bloques, los cuales se distribuyen entre los distintos nodos del sistema. Para garantizar la tolerancia a fallos, se implementa un mecanismo de paridad que permite recuperar la información incluso si uno de los nodos deja de funcionar. Los bloques almacenados pueden contener datos o información de paridad, según el esquema de distribución definido.

El Controller Node, desarrollado en C++, es responsable de administrar la lógica del RAID: decide la ubicación de cada bloque, calcula la paridad correspondiente y reconstruye archivos al momento de descargarlos. Este nodo también expone una interfaz de comunicación mediante HTTP, permitiendo a los usuarios agregar, buscar, eliminar o descargar documentos. El sistema incluye tolerancia a fallos, permitiendo la operación normal aún si uno de los nodos se vuelve inaccesible.

Además, se desarrolla una aplicación con interfaz gráfica que permite a los usuarios interactuar con el sistema de forma amigable y consultar cargas, descargas y borrado de archivos.

El principal reto del proyecto radica en la correcta implementación del esquema RAID 5 en un entorno distribuido, la gestión eficiente de los bloques y paridades, y la sincronización entre múltiples nodos a través de la red. Esto implica aplicar conocimientos de estructuras de datos, sistemas de archivos, algoritmos de codificación de paridad, comunicación en red y desarrollo de interfaces gráficas. El sistema es implementado mayoritariamente en C++, complementado con tecnologías para la comunicación remota y la gestión visual del sistema.

Descripción de la solución

001. Instalación y configuración de Disk Nodes mediante la aplicación TECMFS-Disk, especificando IP/Puerto y ruta de almacenamiento a través de archivo XML.

El sistema TECMFS-Disk gestiona los Disk Nodes mediante un archivo de configuración que especifica los nombres de los discos en el sistema. La configuración de la IP/Puerto y la ruta de almacenamiento de los discos se realiza de forma rígida en el código, utilizando el array disk_names para definir las rutas y los archivos binarios para cada nodo de disco. Al inicio, los discos son verificados, y si no existen, se crean con un tamaño fijo de 32,768 bytes. Esto asegura que el sistema sea escalable y configurable sin necesidad de intervención manual.

Una alternativa era implementar una configuración directamente en el código (sin archivo XML), pero se desestimó debido a la falta de flexibilidad y la dificultad para modificar configuraciones sin recompilar el proyecto. Mientras que una limitación fue si el archivo XML está mal formado o contiene datos incorrectos, el sistema no podrá iniciar correctamente.

002. Definición y verificación del tamaño fijo del Disk Node y de los bloques de datos/paridad para garantizar uniformidad en todo el RAID 5.

El tamaño fijo del Disk Node y de los bloques de datos/paridad se define estáticamente en el código. El tamaño de cada nodo de disco es de 32,768 bytes, lo cual asegura la uniformidad en todo el sistema RAID 5. Además, el código garantiza que cada bloque de datos y de paridad mantenga un tamaño consistente a través de todos los discos. Se verifican las operaciones de lectura y escritura en los discos mediante el tamaño fijo especificado para garantizar que los discos operen de manera coherente y que los bloques de datos y paridad sean uniformes.

Se evaluó permitir tamaños de bloque configurables, pero se desestimó por la complejidad adicional en la gestión de bloques de tamaño variable, lo que afectaría la eficiencia y la tolerancia a fallos del sistema RAID. Además, inicialmente hubo problemas de sincronización al asegurar que todos los discos operaran con el mismo tamaño de bloque.

003. Implementación del Controller Node en C++ para gestionar distribución de bloques, cálculo de paridad y recuperación de datos con tolerancia a fallos.

El Controller Node es responsable de gestionar la distribución de bloques entre los discos y el cálculo de la paridad utilizando el esquema RAID 5. Para cada bloque de datos, se calcula la paridad utilizando un XOR entre los bloques de datos, y esta paridad se distribuye entre los discos. El cálculo de paridad se realiza en la función CalculateParityByte, la cual utiliza el operador XOR para combinar los bytes de datos y generar el byte de paridad correspondiente.

El manejo de la paridad fue un reto, ya que cualquier fallo en un disco requiere reconstruir los datos utilizando la paridad, esto requiere un sistema eficiente de manejo de excepciones para garantizar que el sistema recupere los datos correctamente. Se eligió RAID 5 por su eficiencia y tolerancia a fallos.

004. Desarrollo de interfaz GRPC/HTTP en el Controller Node que permita agregar, eliminar, buscar y descargar documentos con recuperación automática ante fallo de un disco.

La interfaz HTTP fue implementada usando el framework Crow, permitiendo que el Controller Node maneje solicitudes de agregar, eliminar, buscar y descargar documentos. Las rutas HTTP proporcionan las funcionalidades necesarias para interactuar con el sistema: Ruta /upload para cargar archivos, Ruta /get_file para obtener un archivo, Ruta /delete_file para eliminar archivos. Además, el Controller Node maneja los fallos de los discos a través de la función RepairDamagedDisk, asegurando que las operaciones continúen incluso si un disco falla.

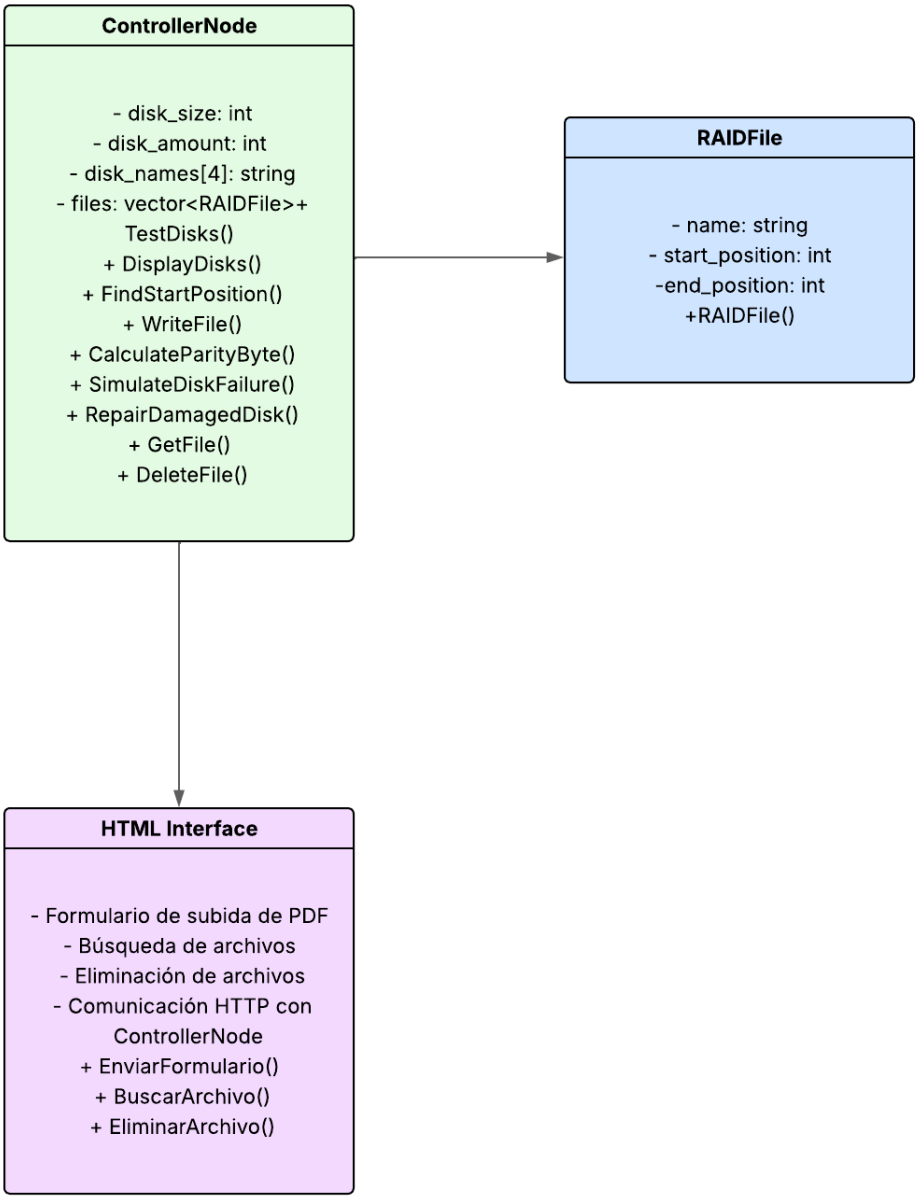
No se implementó GRPC, lo que afectó la eficiencia y la escalabilidad de la comunicación en un sistema distribuido. Si el sistema fuera a crecer en tamaño y nodos, la falta de GRPC podría convertirse en un cuello de botella.

005. Creación de una aplicación gráfica que permita listar, buscar, agregar y eliminar documentos PDF, mostrando además el estado de los bloques por nodo y comunicándose con el Controller Node vía GRPC/HTTP.

La interfaz gráfica se implementó como una aplicación web usando HTML y JavaScript. Se desarrollaron las siguientes funcionalidades: *Subida de archivos PDF:* El formulario permite subir archivos PDF a través de una solicitud POST a http://0.0.0.0:18080/upload. *Búsqueda de archivos:* Permite buscar archivos por nombre, realizando una solicitud GET a http://0.0.0.0:18080/get_file/<nombre_del_archivo>. *Eliminación de archivos:* Los archivos se eliminan mediante una solicitud DELETE a http://0.0.0.0:18080/delete_file/<nombre_del_archivo>. *Estado de los bloques:* Aunque la función DisplayDisks() muestra el estado de los bloques, no está completamente integrada en la interfaz gráfica.

Como alternativa se consideró usar una **aplicación de escritorio** con **Qt** o **Electron**, pero se optó por una interfaz **web** por su facilidad de acceso remoto. Se tuvo algunos problemas al integrar el **frontend** con el **backend,** fue un desafío en cuanto a la sincronización de operaciones asincrónicas. Así mismo la solución usa **Crow** para gestionar las rutas HTTP, facilitando la interacción con el **Controller Node**.

Diagrama UML



Enlace al repositorio

StevenTEC295/Proyecto_3_Datos_II