Sistema de búsqueda y almacenamiento de archivos eficiente mediante tabla hash

Luis Javier palacio
Universidad Eafit
Colombia
ljpalaciom@eafit.edu.co

Santiago Castrillón Galvis
Universidad Eafit
Colombia
scastrillg@eafit.edu.co

Kevyn Santiago Gómez
Universidad Eafit
Colombia
ksgomezp@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

RESUMEN

Hoy en día existen muchos sistemas de búsqueda y almacenamiento de datos mediante diversas estructuras de datos, sin embargo muchas de ellas no pueden suplir la gran cantidad de información que conlleva el amplio volumen de los archivos, puesto que estamos en la era de la información, por lo que cada vez hay más información que necesitamos almacenar y procesar, esto quiere decir que cada vez se hace más necesario idear sistemas mucho más óptimos y duraderos que cumplan con las necesidades tanto actuales como futuras, evitando así tener que lidiar a futuro con un posible espacio poco eficiente debido a una estructura poco eficiente así como ha sucedido en el pasado, esto se puede evidenciar en estructuras como el ext2, en el cual todavía no se había implementado el journaling, posteriormente la estructura ext3 comenzó a implementarlo, pero una vez más esto no fue suficiente, ya que carecía de extensiones, localización dinámica de inodos, sublocalización de bloques, herramienta de desfragmentación, compresión, y además de esto al no tener una suma de verificación existía el riesgo de que los archivos se corrompen por fallos del hardware.

Palabras clave

Registro por diario, sublocalización, desfragmentación, inodos, compresión, sistemas de búsqueda y almacenamiento, tabla de hash, árbol B, listas enlazadas.

Palabras clave de la clasificación de la ACM

Theory of computation → Data management systems → Data structures → Data access methods → Data scans

Theory of computation → Information storage systems → Record storage systems → Directory structures → Extent-based file structures

1.INTRODUCCIÓN

En el mundo real se está viendo una masificación de la información de una forma exponencial, la globalización ha permitido que la comunicación a nivel mundial este a disposición de cualquier persona, y de una manera muy sencilla y rápida, por esto la proliferación de la información es un fenómeno que ha llevado a los sistemas de archivos a estar en constante evolución, ya que a medida que pasa el tiempo las exigencias son mayores, por lo que se hace necesario cambiar a estructuras de datos más eficientes, con mejores herramientas que faciliten y permitan el manejo óptimo de la información. En este proyecto se busca estudiar el proceso por el que han pasado las diversas estructuras de datos que han hecho parte de los sistemas de archivos para analizar los problemas que se han presentado y de esta manera tener una visión más general y completa para buscar una solución que satisfaga las necesidades actuales y tenga en cuenta futuros problemas que se puedan presentar y la forma en la que deben ser afrontados.

2. PROBLEMA

El avance de la tecnología ha provocado que los sistemas de archivos estén en constante evolución, esto significa que el sistema que hace unos años parecía ser el definitivo, ahora no es igual de efectivo, y esto se ha convertido en una tendencia, así que el problema radica en encontrar la forma de optimizar el sistema para que se ajuste a las exigencias de la actualidad y que tenga un funcionamiento a largo plazo, esto tiene una trascendencia de alto impacto pues la tecnología sigue en evolución, pero si no hay una forma de ordenarla entonces habrá un estancamiento en el progreso, lo que puede repercutir en ámbitos sociales y económicos de todo el mundo.

3. TRABAJOS RELACIONADOS

3.1 XFS

Es uno de los primeros sistemas de archivos de 64 bits que incorporaron el registro por diario, y a partir de este, todos los siguientes sistemas de archivos comenzaron a implementarlo debido a que éste evita los engorrosos y largos chequeos de disco que efectúan los sistemas al apagarse bruscamente. Este sistema funciona con la estructura árbol B. Es un sistema estable, sin embargo no es de los sistemas más eficientes en cuanto a tiempo de ejecución, y su mayor problema es que carece de cifrado de datos transparente, por lo que su seguridad es superada por la de otros sistemas.

3.2 ReiserFS

Es un sistema de archivos de propósito general que almacena metadatos sobre los ficheros, entradas de directorio y listas de inodos en un único árbol B+ cuya clave principal es un identificador único. Posee una capacidad de 4 mil millones de archivos. Su problema radica en que suele ser muy inestable y no tiene un sistema de desfragmentación de archivos.

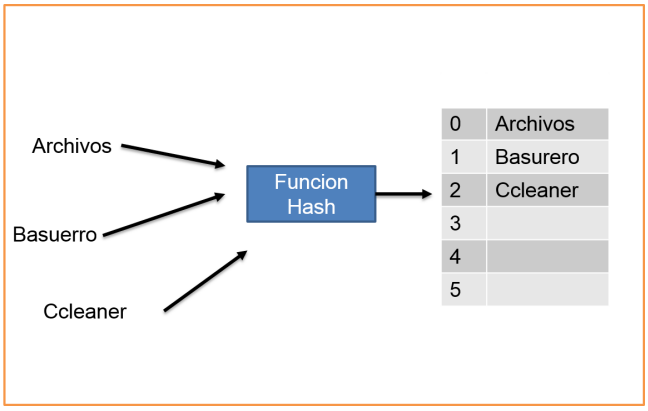
3.3 ext3

Este sistema se caracteriza por su evolución directa del ext2, mejorando muchos de los aspectos anteriores e implementando el registro por diario, sus cualidades son el crecimiento en línea,la estructura de árbol AVL, y el bajo consumo de cpu, sin embargo su funcionalidad ha quedado obsoleta debido a que no ha implementado las extensiones, la localización dinámica de los inodos, y la sublocalización de los bloques, que son aspectos indispensables en los sistemas más novedosos como el ext4.

3.4 HFS

Se caracteriza por ser un sistema de archivos jerárquico lo que permitió que el código del programa fuese almacenado por separado de recursos tales como iconos que pudiesen necesitar ser localizados, el tamaño máximo de archivos es de 66 mil y su estructura es árbol B. Su problema radica en que almacena todos los registros de archivos y directorios en una estructura de datos sencilla, lo que provoca bajo rendimiento cuando el sistema se encuentra realizando múltiples tareas.

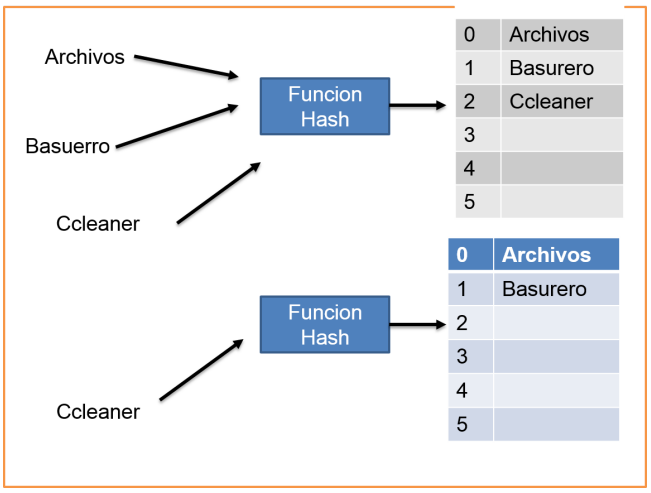
4. Tabla de Hash



Gráfica 1: Una tabla de Hash simple, un arreglo que promete eficacia gracias a la función hash

4.1 Operaciones de la estructura de datos

Gráfica 2: Imagen de una operación de agregación en una tabla de hash.



Gráfica 3: Imagen de una de añadir en una tabla de Hash

4.2 Criterios de diseño de la estructura de datos

Para lograr este objetivo se tomaron en cuenta varias estructuras de almacenamiento de datos como lo son listas enlazadas, árbol b, tabla hash, arreglo y pila. Después de analizar las complejidades de estas estructuras en los objetivos específicos de inserción y búsqueda, se llegó a la conclusión de que la estructura más eficaz para esto es la tabla hash, debido a que en la inserción, lo único que se requiere es ingresar una clave, y la función hash se encarga de determinar el lugar idóneo para almacenarla, de tal forma que para búsquedas posteriores, el tiempo de ejecución sea considerablemente más bajo.

4.3 Análisis de Complejidad

Metodo	Complejidad
actualizarDirección	O(n) “siendo n el numero de la profundidad del directorio”
listarContenido	O(n)
añadir	O(n) “siendo n el numero de la profundidad del directorio”
obtener	O(1)

Tabla 1: Tabla para reportar la complejidad

4.4 Tiempos de Ejecución

Operacion	Mejor tiempo	Peor tiempo	Tiempo promedio
leerArchivo	653 ms	1.9 s	1.3 s
buscar con nombre	0 ms	0.04 ms	0 .00 ms
buscar con ruta	0.07ms	0.23 ms	0.013ms
imprimir	0.87ms	4ms	2.8 ms

Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con treeEtc.txt

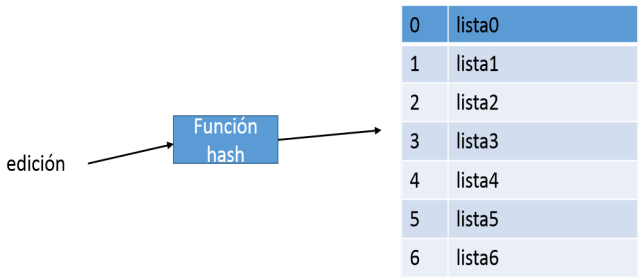
4.6 Análisis de los resultados

Operacion	Mejor tiempo	Peor tiempo	Tiempo promedio
leerArchivo	653 ms	1.9 s	1.3 s
buscar con nombre	0 ms	0.04 ms	0 .00 ms
buscar con ruta	0.07ms	0.23 ms	0.013ms
imprimir	0.87ms	4ms	2.8 ms

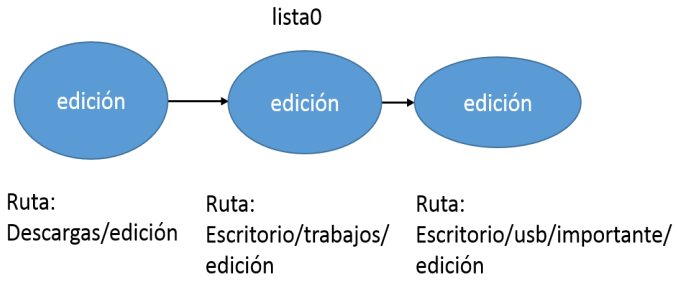
Tabla 4: Análisis de los resultados obtenidos con la implementación de la estructura de datos

5. Funcionamiento de la estructura de datos:

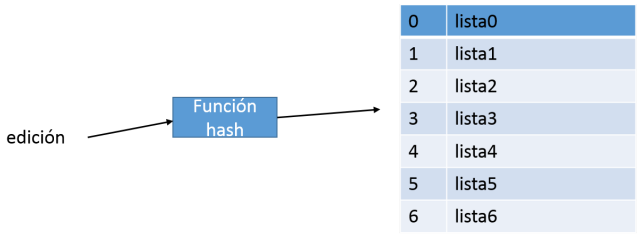
Consiste en una tabla hash de listas enlazadas de carpetas. La clave y el valor de la tabla hash son respectivamente el nombre de la carpeta y una lista de carpetas que poseen el mismo nombre; esto se hizo con el propósito de controlar las carpetas que tienen ruta distinta pero nombres iguales.



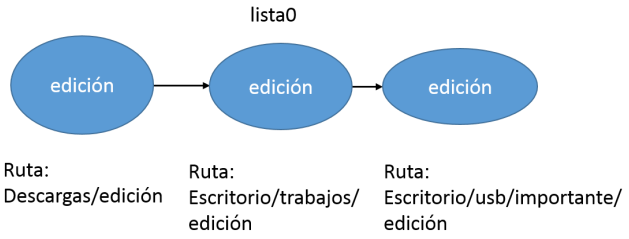
Gráfica 4: Listas enlazadas que contienen carpetas y/o archivos con nombre idéntico. Para diferenciarlos se usan sus rutas puesto que estas siempre son distintas. El algoritmo de búsqueda con ruta es eficiente siempre y cuando no hayan demasiadas carpetas y/o archivos con el mismo nombre.



5.1 Operaciones de la estructura de datos



Gráfica 5: Método de agregación de carpetas y búsquedas de las mismas. Se ingresa como clave el nombre de la carpeta y mediante la función hash obtenemos una lista enlazada de carpetas con el mismo nombre. De este modo podemos añadir la nueva carpeta a la lista, o imprimir la lista como las coincidencias de una búsqueda.



Gráfica 6: Lista enlazada de carpetas o archivos.

5.2 Criterios de diseño de la estructura de datos

Para lograr este objetivo se tomaron en cuenta varias estructuras de almacenamiento de datos como lo son listas enlazadas, árbol b, tabla hash, arreglo y pila. Después de analizar las complejidades de estas estructuras en los objetivos específicos de inserción y búsqueda, se llegó a la conclusión de que la estructura más eficaz para esto es la tabla hash, debido a que en la inserción, lo único que se requiere es ingresar una clave, y la función hash se encarga de determinar el lugar idóneo para almacenarla, de tal forma que para búsquedas posteriores, el tiempo de ejecución sea considerablemente más bajo.

5.3 Análisis de la Complejidad

Metodo	Complejidad
actualizarDirección	O(n) "siendo n el número de la profundidad del directorio"
listarContenido	O(n)
añadir	O(n) "siendo n el número de la profundidad del directorio"
obtener con nombre	O(1)
obtener con ruta	O(n) "siendo n el número de las carpetas que tienen el mismo nombre"
buscar con nombre	O(1)
buscar "este buscar se emplea con una ruta"	O(n) "siendo n el número de archivos en la lista"

Tabla 5: Tabla para reportar la complejidad

5.4 Tiempos de Ejecución

Operacion	Mejor tiempo	Peor tiempo	Tiempo promedio
leerArchivo	653 ms	1.9 s	1.3 s
buscar con nombre	0 ms	0.04 ms	0 .00 ms
buscar con ruta	0.07ms	0.23 ms	0.013ms
imprimir	0.87ms	4ms	2.8 ms

Tabla 6: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos

5.6 Análisis de los resultados

Las operaciones de búsqueda tienen muy poco tiempo de ejecución y esto concuerda con los cálculos de complejidad. El algoritmo demuestra ser eficiente para situaciones de la vida real donde no hay mucha profundidad en los directorios (por ejemplo 50 carpetas cada una metida dentro de otra) o muchos archivos llamados igual (por ejemplo 50 archivos llamados a.txt) No tuvimos la oportunidad de medir la memoria puesto que no sabíamos interpretar los datos de visualVm correctamente, sin embargo presumimos que la estructura en el peor de los casos consume 25 megabytes de memoria ram, lo cual no es significativo ni para una ram de 4 gigabytes de capacidad.

Operación	Mejor tiempo	Peor tiempo	Tiempo promedio
leerArchivo	653 ms	1.9 s	1.3 s
buscar con nombre	0 ms	0.04 ms	0 .00 ms
buscar con ruta	0.07ms	0.23 ms	0.013ms
imprimir	0.87ms	4ms	2.8 ms

6. CONCLUSIONES

Consideramos que lo más importante al realizar este reporte fue la investigación que hicimos respecto a el funcionamiento de las estructuras de datos, en nuestro caso el método de búsqueda de las carpetas o archivos que tenían un nombre idéntico puesto que esto supuso un gran desafío para la estructura que elegimos: una tabla hash. Logramos solucionar este inconveniente haciendo, no una tabla hash de carpetas como teníamos pensado al principio, sino una tabla hash de listas de carpetas, donde cada lista contiene las carpetas que tienen igual nombre pero ruta distinta.

En relación a los resultados, lo más significativo fueron los tiempos de ejecución que sin contar la lectura del archivo son por debajo de los 3 ms y esto demuestra la buena implementación de nuestra estructura de datos.

La primera solución que implementamos estaba incompleta y presentaba muchos errores respecto a la lectura de el txt y la búsqueda de archivos y carpetas que tenían un mismo nombre. Ahora nuestra estructura de datos es muy rápida, con una buena implementación que controla eficazmente las colisiones de archivos y carpetas y, además, permite búsquedas veloces.

Una posible mejora que deseamos hacerle al programa es darle la posibilidad de leer el disco duro, para darle utilidad. Sin embargo, puesto que ya se trata de un dispositivo distinto, quizá la estructura de datos que usamos es completamente ineficiente en éste. Así que no estamos seguros de continuar con el desarrollo de esta aplicación, sin embargo surgió un interés por desarrollar aplicaciones con estructuras de alta eficiencia, y coincidimos en que ésta la eficiencia es un área de suma importancia para el futuro.

6.1 Trabajos futuros

En el proyecto, sentimos que a futuro lo mejor sería implementar un programa que funcionase con el disco duro y usar estructuras de datos más complejas como lo son el árbol b.

AGRADECIMIENTOS

Esta investigación fue soportada parcialmente por Andi-Eafit, Ministerio de Educación Nacional y la Fundación Suiza quienes permiten nuestra educación y nos dan su máximo apoyo para disfrutar de la experiencia del conocimiento.