Simulación de Sistemas de Archivos: Impacto de la Política de Asignación en Rendimiento y Fragmentación

Marco Castilla Axel Cueva Gabriel Poma Angel de la Cruz Marcos Basualdo
UNMSM — Curso de Sistemas Operativos
{marco.castilla,axel.cueva,gabriel.poma,marcos.basualdo,angel.delacruz}@unmsm.edu.pe

Abstract

Este trabajo estudia, mediante simulación, el efecto de tres esquemas clásicos de asignación de archivos (*contiguous*, *linked* y *indexed*) sobre métricas clave: tiempo de acceso, throughput, uso de espacio, y fragmentación (externa e interna). Se diseñaron cargas de trabajo que mezclan archivos grandes/pequeños, patrones secuenciales/aleatorios y escenarios con creación/borrado intensivo para inducir fragmentación. Reportamos resultados agregados y discusión comparativa por escenario. El código y los datos de los experimentos están disponibles en un repositorio público.

1 Introducción

Los sistemas de archivos deben balancear eficiencia de acceso, utilización de espacio y control de la fragmentación. Aunque existen técnicas avanzadas de gestión de disco, la fragmentación de archivos sigue siendo una condición medible y prevalente en el uso activo de sistemas modernos, lo que justifica el análisis de sus consecuencias en el rendimiento **(Hoogstraaten & Z, 2021)**.

La política de asignación de bloques condiciona latencia, número de *seeks* lógicos, y la probabilidad de fragmentación externa. Para lograr esto, el principal reto de la implementación es mantener el registro de los bloques de disco asignados a cada archivo **(Tanenbaum & Bos, 2014, Secc. 4.3)**. En este trabajo analizamos tres políticas ampliamente descritas: asignación *contigua*, *enlazada (linked)* e *indexada (indexed)*. A partir de un simulador reproducible, medimos su impacto en tareas de creación, lectura y escritura bajo distintos perfiles de carga **(Silberschatz et al., 2018, Cap. 11)**.

Nuestra contribución es doble: (i) un *framework* de simulación con contratos claros por capa (disco, gestor de espacio libre, y estrategia de asignación) que permite UI o CLI intercambiables; (ii) un estudio experimental con 30 ejecuciones (barridos de las tres estrategias) que caracterizan el comportamiento de cada política por escenario. Los resultados crudos provienen del agregado de ejecuciones por `seed` y están disponibles como archivo JSON de resultados experimentales.

2 Metodología

2.1 Algoritmos implementados

Asignación contigua. Cada archivo ocupa un rango físico contiguo; el directorio guarda $(start, length)$. Ventajas: acceso secuencial óptimo y cero *seeks* lógicos intra–archivo; desventajas: sensibilidad a la fragmentación externa y a crecimiento in–place, un problema que motivó el desarrollo de políticas de asignación más sofisticadas para mitigar sus efectos en el rendimiento **(Smith & Seltzer, 1996)**.

Asignación enlazada (linked). El directorio guarda el bloque inicial; cada bloque reserva cabecera para un puntero al siguiente. Pros: baja presión de contigüidad (crece donde haya huecos); contras: coste de *seek* lógico para saltos/offsets y menor localidad **(Silberschatz et al., 2018, Cap. 11)**.

Asignación indexada (indexed, nivel único). Cada archivo reserva un bloque de índice con punteros a sus bloques de datos. Ofrece acceso directo por offset; la cabida máxima depende de la relación `block_size / sizeof(pointer)`. Reduce *seeks* lógicos respecto a *linked* y tolera no contigüidad **(Silberschatz et al., 2018, Cap. 14, pp. 575–577)**.

2.2 Arquitectura de simulación

El simulador separa responsabilidades: `Disk` (bloques lógicos con `block_size`), `FreeSpaceManager` (bitmap con *first–fit* contiguo/no–contiguo y métricas de huecos), y una clase `FilesystemBase` con `create/delete/read/write` como contrato.

Las estrategias (`ContiguousFS`, `LinkedFS`, `IndexedFS`) instrumentan un canal de eventos para recolectar *seeks* lógicos estimados y bloques tocados. Una CLI guía barridos (*sweep*) y la UI puede subscribirse a callbacks del bitmap para visualización.

2.3 Escenarios de prueba

Se usaron tres escenarios canónicos, definidos por parámetros de tamaño de disco/bloque, mezcla de tamaños de archivo y patrón de acceso. Cada ejecución toma una `seed` y aplica la misma carga a las tres estrategias mediante barrido:

- **Mezcla Pequeños/Grandes** (`mix-small-large`): mezcla de archivos pequeños (1–16 bloques) y grandes (256–2048), con acceso 60% secuencial / 40% aleatorio; $\sim$1230 operaciones por corrida.
- **Secuencial vs Aleatorio** (`seq-vs-rand`): comparación controlada con 90% secuencial o 90% aleatorio; $\sim$970 operaciones por corrida.
- **Fragmentación Intensiva** (`frag-intensive`): tasa de creación/borrado elevada para inducir huecos; $\sim$1760 operaciones por corrida.

Plan experimental. Se realizaron **30 experimentos** en total, todos como *sweeps* (las tres estrategias por corrida). Cada integrante ejecutó **6 experimentos** (= 2 seeds por escenario). Las `seeds` se tomaron de `random.org`. (rango 1–1.000.000) y se distribuyeron en los siguientes cinco conjuntos utilizados para la inicialización:

1. Set 1: 44552, 262655, 768582, 801240, 879263, 912983
2. Set 2: 215431, 383868, 516415, 677082, 842372, 974120
3. Set 3: 589265, 620050, 678083, 769217, 913182, 962938
4. Set 4: 55907, 81652, 159233, 346606, 697256, 795519
5. Set 5: 22973, 153259, 242552, 245635, 548373, 888898

Cada corrida produce un resumen por estrategia y un *bitmap* final del espacio libre.

2.4 Métricas

Reportamos:

- **Tiempo de acceso promedio (ms)** por operación.
- **Throughput** (operaciones/segundo) agregado por corrida.
- **Uso de espacio (%)** = bloques usados / bloques totales.
- **Fragmentación externa (%)** mediante $1 - \frac{\text{mayor run libre}}{\text{total libre}}$ (0% = sin fragmentación visible).
- **Seeks lógicos (estimados)** a partir de discontinuidades en el mapeo lógico$\rightarrow$físico (proxy del coste de salto).
- **Hit/Miss** (proporción de operaciones exitosas), y **uso de CPU (%)** medido con tiempo de proceso relativo al tiempo de pared.

3 Resultados y Discusión

Para facilitar la comprensión, preparamos **cinco tablas** con promedios $\pm$ desviación por escenario y estrategia: (i) tiempo de acceso, (ii) throughput, (iii) fragmentación externa, (iv) uso de espacio y (v) seeks estimados.

Escenario	Contigua	Enlazada	Indexada
Mezcla Pequeños/Grandes	0.17 ± 0.06	0.47 ± 0.19	0.05 ± 0.02
Secuencial vs Aleatorio	0.13 ± 0.05	0.28 ± 0.13	0.09 ± 0.05
Fragmentación Intensiva	0.08 ± 0.04	0.31 ± 0.16	0.03 ± 0.01

Table 1: Tiempo de acceso promedio (ms) por estrategia y escenario (n=10 por combinación). Valores reportados como media $\pm$ desviación estándar.

Escenario	Contigua	Enlazada	Indexada
Mezcla Pequeños/Grandes	279.56 ± 87.40	250.38 ± 82.29	272.47 ± 98.28
Secuencial vs Aleatorio	347.91 ± 103.02	318.99 ± 106.14	327.25 ± 121.97
Fragmentación Intensiva	277.68 ± 93.75	249.00 ± 90.80	273.72 ± 99.61

Table 2: Throughput (operaciones/segundo) por estrategia y escenario (n=10 por combinación). Valores reportados como media $\pm$ desviación estándar.

Escenario	Contigua	Enlazada	Indexada
Mezcla Pequeños/Grandes	48.7 ± 6.1	19.4 ± 4.8	0.5 ± 0.3
Secuencial vs Aleatorio	10.0 ± 3.5	2.6 ± 1.4	0.3 ± 0.1
Fragmentación Intensiva	13.7 ± 3.5	10.0 ± 2.2	1.0 ± 0.1

Table 3: Fragmentación externa (%) por estrategia y escenario (n=10 por combinación). Valores reportados como media $\pm$ desviación estándar.

Escenario	Contigua	Enlazada	Indexada
Mezcla Pequeños/Grandes	0.17 ± 0.06	0.47 ± 0.19	0.05 ± 0.02
Secuencial vs Aleatorio	0.13 ± 0.05	0.28 ± 0.13	0.09 ± 0.05
Fragmentación Intensiva	0.08 ± 0.04	0.31 ± 0.16	0.03 ± 0.01

Table 4: Tiempo de acceso promedio (ms) por estrategia y escenario (n=10 por combinación). Valores reportados como media $\pm$ desviación estándar.

Escenario	Contigua	Enlazada	Indexada
Mezcla Pequeños/Grandes	0 ± 0	29 ± 8	34 ± 7
Secuencial vs Aleatorio	0 ± 0	13 ± 3	14 ± 5
Fragmentación Intensiva	0 ± 0	104 ± 27	94 ± 16

Table 5: Seeks estimados (conteo) por estrategia y escenario (n=10 por combinación). Valores reportados como media $\pm$ desviación estándar.

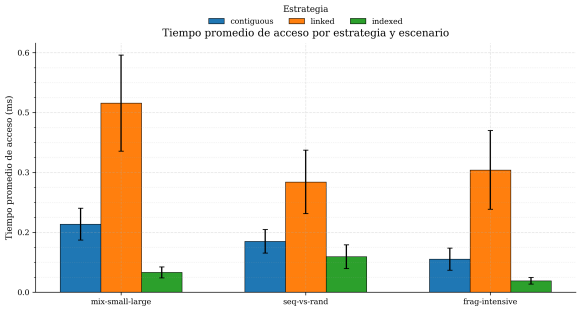

Figure 1: Tiempo de acceso promedio por estrategia y escenario.

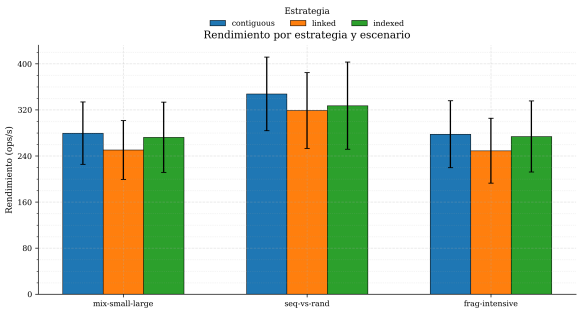

Figure 2: Throughput (ops/s) por estrategia y escenario.

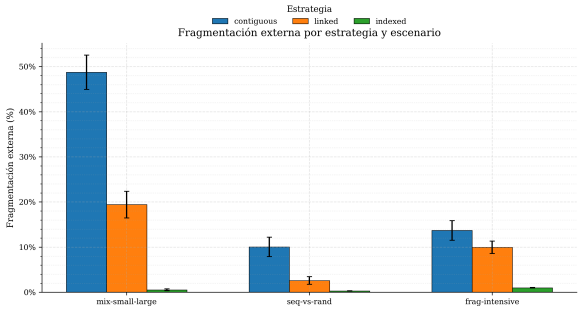

Figure 3: Fragmentación externa (%) por estrategia y escenario.

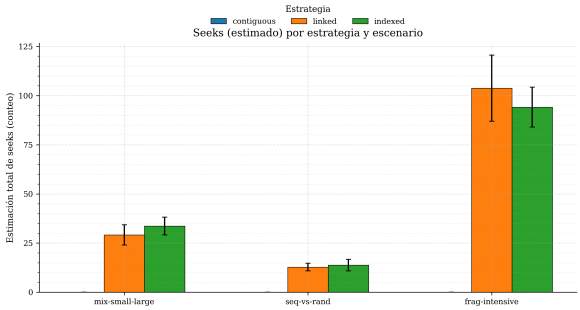

Figure 4: Seeks lógicos estimados por estrategia y escenario.

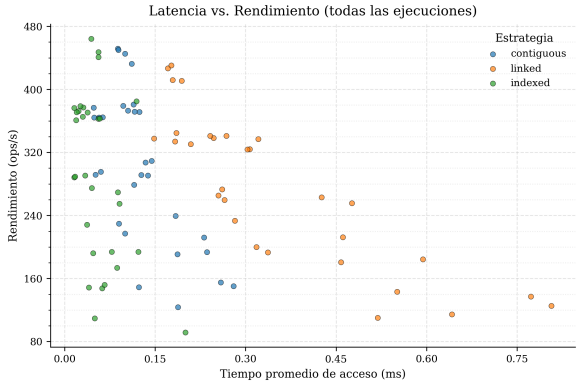

Figure 5: Relación latencia–rendimiento.

Tendencias generales. (i) **Indexed** exhibe las menores latencias promedio y muy baja fragmentación externa en los tres escenarios, coherente con su acceso directo por índice y tolerancia a no contigüidad. (ii) **Contiguous** muestra throughput alto y *seeks* lógicos ≈ 0, pero sufre fragmentación externa elevada en escenarios con mezcla y borrado, lo que tensiona la reserva contigua para archivos grandes. (iii) **Linked** mantiene uso de espacio competitivo y fragmentación externa moderada, a costa de mayor latencia y *seeks* lógicos en accesos aleatorios debido al *pointer chasing*.

Mezcla Pequeños/Grandes. La presión por bloques contiguos expone a *contiguous* a huecos pequeños que no acomodan archivos grandes; *indexed* conserva latencias bajas y fragmentación

mínima al dispersar datos sin contiguidad. *linked* se posiciona en medio: robusto a huecos pero penalizado en saltos.

Secuencial vs Aleatorio. Cuando predomina acceso secuencial, *contiguous* y *indexed* convergen en throughput alto (lectura lineal y búsqueda por índice); en acceso aleatorio la ventaja de *indexed* es mayor por direccionamiento directo, mientras *linked* acumula más *seeks*.

Fragmentación intensiva. El patrón de crear/borrar acelera la aparición de runs libres pequeños. *indexed* mantiene baja fragmentación externa y latencias cortas; *linked* degrada menos que *contiguous* en espacio, pero su latencia crece por enlaces largos.

4 Conclusiones

El estudio confirma intuiciones clásicas: **indexed** es una opción robusta frente a fragmentación con latencias bajas y buena utilización; **contiguous** domina en accesos secuenciales sin churn, pero es sensible a borrado/crecimiento y a la necesidad de runs contiguos; **linked** es flexible en espacio pero paga en latencia por *pointer chasing*. Como trabajo futuro, se propone añadir índices multinivel, *extents* y políticas alternativas de *free–space* (best–fit, buddy) para explorar compromisos adicionales.

Repositorio. El código fuente, scripts para reproducibilidad, tablas LaTeX y figuras están disponibles en: https://github.com/mrcastilla8/Sistema-de-Archivos-simulados

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne. 2018. *Operating System Concepts* (10th ed.). Wiley. https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf.

[2] A. S. Tanenbaum and H. Bos. 2014. *Modern Operating Systems* (4th ed.). Pearson.

[3] Stallings, W. (2017). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

[4] H. Hoogstraaten and L. Z. 2021. *A contemporary investigation of NTFS file fragmentation*. https://dfrws.org/wp-content/uploads/2021/01/2021_APAC_paper-a_contemporary_investigation_of_ntfs_file_fragmentation.pdf.