COMPARACIÓN DE ALGORITMOS DE PLANIFICACIÓN DE DISCOS.

Comparison of Disk Scheduling Algorithms

LUISANA, G. LEGONÍA.P & JESUS, E. HERNANDEZ C.

Universidad de Carabobo. Facultad Experimental de Ciencias y Tecnología.

Departamento de Computación. Naguanagua, Carabobo.

{luisalegoni;jesuseduardohc}@gmail.com

Resumen

La gestión eficiente de solicitudes de acceso en discos mecánicos es crucial para optimizar el rendimiento en sistemas operativos, donde algoritmos como **FCFS**, **SSTF** y **SCAN** presentan ventajas y limitaciones según el contexto. Se comparó su rendimiento mediante simulaciones realistas, analizando métricas como tiempo total de búsqueda y latencia promedio. Los resultados demostraron que **SSTF** reduce la latencia en un **63-80%** frente a **FCFS**, aunque exhibe inanición en cargas extremas, mientras que **SCAN** garantiza equidad al atender todas las solicitudes, con un costo de tiempo un **35% mayor** que SSTF. **SSTF** es ideal para entornos prioritarios de baja latencia (ej: servidores web), y **SCAN** se destina a sistemas que exigen equidad, como aplicaciones en tiempo real o bases de datos críticas.

Palabras claves: equidad, FCFS, latencia, rendimiento, SCAN, SSTF, tiempo.

Abstract

Efficient management of access requests on mechanical disks is crucial for optimizing performance in operating systems, where algorithms such as FCFS, SSTF, and SCAN present advantages and limitations depending on the context. Their performance was compared using realistic simulations, analyzing metrics such as total seek time and average latency. The results showed that SSTF reduces latency by 63–80% compared to FCFS, although it exhibits starvation under extreme loads, while SCAN guarantees fairness by serving all requests, with a time cost 35% higher than SSTF. SSTF is ideal for priority low-latency environments (e.g., web servers), and SCAN is intended for systems that demand fairness, such as real-time applications or mission-critical databases.

Keywords: fairness, FCFS, latency, performance, SCAN, SSTF, time

1. Introducción

La planificación de discos es un componente esencial en los sistemas operativos modernos, directamente vinculado al rendimiento de almacenamiento y la capacidad de manejar múltiples procesos concurrentes. Aunque tecnologías emergentes como los discos de estado sólido (SSD) han transformado el panorama, los discos mecánicos siguen siendo relevantes en entornos industriales, sistemas heredados y aplicaciones de backup masivo. La optimización del movimiento del cabezal no solo reduce el tiempo de acceso, sino que también prolonga la vida útil del hardware al minimizar desplazamientos innecesarios (Silberschatz et

al. 2018, p. 543). Se busca ampliar el análisis clásico de algoritmos de planificación, incorporando cargas de trabajo mixtas y extremas que replican escenarios realistas. Trabajos anteriores, destacan la importancia de comparar estos métodos, pero pocos profundizan en cómo se comportan bajo distribuciones bimodales o en sistemas con requisitos de equidad estrictos (*T*anenbaum y Bos, 2015 p. 312*)*.

2. Materiales y Métodos

2.1. Diseño Experimental

El diseño experimental se estructuró para evaluar de manera rigurosa y sistemática el comportamiento de los algoritmos **FCFS**, **SSTF** y **SCAN** bajo condiciones controladas, replicando escenarios realistas de acceso a discos mecánicos. A continuación, se profundiza en los componentes críticos del entorno de simulación y las cargas de trabajo diseñadas.

Los algoritmos de planificación de disco, como FCFS, SSTF y SCAN, juegan un papel crucial en la gestión de las solicitudes de acceso al disco. **FCFS (First Come, First Served)** funciona atendiendo las solicitudes en el orden en que llegan, sin priorizar ninguna en particular. Este método es simple de implementar, pero puede generar latencias elevadas cuando las solicitudes están distribuidas de forma dispersa. **SSTF (Shortest Seek Time First)**, por su parte, prioriza la solicitud más cercana a la posición actual del cabezal del disco. Aunque reduce la latencia promedio en comparación con FCFS, puede provocar problemas de inanición para solicitudes distantes si hay una alta concentración de accesos cercanos. Por último, **SCAN** opera como un elevador, moviendo el cabezal hacia un extremo del disco y luego regresando al otro, atendiendo las solicitudes en el camino. Este enfoque asegura un tratamiento más equitativo, aunque la latencia puede variar según la posición de las solicitudes.

La latencia promedio en el acceso a discos depende de varios factores, como el tiempo de búsqueda (seek time), el tiempo de rotación (rotational latency) y el tiempo de transferencia de datos. La planificación de solicitudes afecta directamente a la latencia promedio, ya que algoritmos más eficientes pueden reducir los desplazamientos innecesarios del cabezal del disco y optimizar el tiempo de respuesta.

La búsqueda de cilindros en un disco se refiere al movimiento del cabezal de lectura/escritura hacia la posición adecuada para acceder a los datos. El disco está dividido en pistas, que forman cilindros cuando se consideran las múltiples superficies de los platos. Durante la búsqueda, el sistema calcula la distancia entre la posición actual del cabezal y la pista requerida, moviéndose hacia el cilindro correspondiente. Este proceso puede ser optimizado mediante algoritmos de planificación, que determinan el orden de las solicitudes para minimizar el desplazamiento del cabezal y, en consecuencia, el tiempo total de acceso.

2.1.1 Simulador de Disco.

El simulador de disco se desarrolló con el objetivo de emular fielmente las operaciones físicas de un disco duro magnético tradicional, incorporando características técnicas y mecánicas clave:

El sistema utiliza 200 pistas numeradas (0-199), un rango que representa un disco estándar de tamaño medio. Cada pista corresponde a una posición circular en el plato del disco, y la elección de este número busca equilibrar la complejidad con la representatividad de sistemas reales. Esta configuración permite simular escenarios típicos de almacenamiento sin introducir complicaciones excesivas.

El cabezal se desplaza con una velocidad de 1 ms por pista, equivalente a discos HDD modernos de 7200 RPM, lo que garantiza un modelo realista para pruebas de rendimiento. Su posición inicial se establece en la pista 100 (centro del disco) para estandarizar las condiciones y eliminar sesgos derivados de ubicaciones arbitrarias. Además, su movimiento puede ser unidireccional o bidireccional, dependiendo del algoritmo empleado (e.g., SCAN requiere oscilación entre bordes, mientras FCFS mantiene una dirección fija).

El módulo *AlgoritmosPlanificación* integra métodos para FCFS, SSTF y SCAN, validados mediante pruebas unitarias con casos límite, como secuencias vacías o solicitudes concentradas en una sola pista. Por su parte, el *DiscoSimulador* modela el entorno físico, registrando movimientos del cabezal, tiempos de búsqueda y posiciones de acceso. Esta dualidad asegura tanto la precisión algorítmica como la fidelidad en la simulación de hardware.

Para garantizar consistencia entre ejecuciones, se emplea una semilla aleatoria fija (valor 42), asegurando que las solicitudes "aleatorias" sean idénticas en cada prueba. Adicionalmente, se generan logs detallados con trazas de ejecución, incluyendo secuencias de acceso, desplazamientos del cabezal y tiempos de operación. Estos registros facilitan el análisis comparativo y la depuración de posibles inconsistencias en los resultados.

2.1.2. Cargas de Trabajo.

Se diseñaron tres escenarios de carga de trabajo para evaluar los algoritmos bajo distribuciones espaciales variadas, desde aleatoriedad pura hasta clusters extremos:

Caso Básico (Tabla I)

Este escenario de prueba evalúa el comportamiento de los algoritmos de planificación bajo condiciones genéricas de acceso. Se seleccionaron 8 solicitudes generadas aleatoriamente dentro del rango completo de pistas (0-199), simulando una carga de trabajo sin preferencias espaciales ni temporalidad definida. Un ejemplo de secuencia generada es: *98, 183, 37, 122, 14, 65, 67, 199*, donde las solicitudes se distribuyen de manera dispersa y sin agrupamientos evidentes.

Las 8 solicitudes se generaron mediante una distribución uniforme, asegurando que cada pista tenga la misma probabilidad de ser seleccionada. Este enfoque refleja accesos impredecibles, comunes en sistemas donde múltiples usuarios o procesos compiten por recursos del disco simultáneamente. La ausencia de patrones específicos permite observar cómo los algoritmos gestionan la aleatoriedad sin depender de optimizaciones basadas en tendencias preexistentes.

Se analiza el rendimiento de los algoritmos en entornos multiusuario con cargas heterogéneas, como servidores web que manejan tráfico diverso. En tales contextos, las solicitudes suelen carecer de correlación espacial o temporal, lo que exige a los algoritmos adaptarse dinámicamente sin sesgos hacia zonas específicas del disco. Esta prueba busca medir la eficiencia en términos de latencia promedio y máxima, así como la equidad en el servicio de solicitudes dispersas.

La distribución uniforme se emplea para replicar escenarios donde no existen patrones discernibles en las solicitudes, un desafío crítico en sistemas reales con alta variabilidad. Al eliminar sesgos hacia regiones del disco, se evalúa la capacidad intrínseca de los algoritmos para manejar aleatoriedad pura, sin ventajas derivadas de optimizaciones predictivas. Esto prueba su robustez en condiciones adversas, donde la falta de orden en las peticiones podría generar inanición (en SSTF) o recorridos innecesariamente largos (en FCFS). Además, este enfoque facilita comparaciones justas entre algoritmos, al neutralizar factores externos como localidad de acceso o concentración de datos.

Caso Intermedio (Tabla II)

Evalúa el comportamiento de los algoritmos ante un patrón de acceso bimodal, donde las solicitudes se concentran en dos regiones opuestas del disco. La distribución divide las 8 solicitudes en dos clusters: el 50% en pistas bajas (0-50) y el 50% en pistas altas (150-199). Un ejemplo de secuencia generada es: *12, 23, 34, 45, 150, 170, 185, 199*, reflejando accesos frecuentes a zonas específicas con propósitos diferenciados.

Las solicitudes se dividen equitativamente entre dos rangos de pistas: las primeras cuatro (0-50) representan accesos recurrentes a metadatos o configuraciones del sistema, como directorios raíz o tablas de asignación. Las siguientes cuatro (150-199) simulan accesos a datos de usuario almacenados en zonas periféricas, típicas de archivos grandes o registros históricos. Esta separación espacial intencional replica sistemas fragmentados o aplicaciones con datos distribuidos estratégicamente.

El propósito principal es analizar la adaptabilidad de los algoritmos frente a clusters opuestos, un desafío común en entornos como bases de datos donde índices y registros residen en áreas separadas del disco. Se busca medir cómo equilibran la atención entre regiones distantes, evitando sesgos hacia una zona y garantizando latencias aceptables en ambas. Esto es crítico para sistemas que requieren balancear accesos frecuentes a metadatos con operaciones intensivas en datos periféricos.

Caso Complejo (Tabla III):

Abarca 15 solicitudes extremas distribuidas de forma concentrada en las áreas límite del disco. En este escenario, se generan 10 solicitudes en la pista 0, lo que simula accesos continuos a datos críticos como las tablas de particiones y los registros del sistema. Adicionalmente, se plantean 5 solicitudes en la pista 199, que imitan accesos esporádicos a archivos de gran tamaño ubicados en las zonas periféricas del disco.

El principal objetivo es evaluar tanto la equidad como la capacidad de los algoritmos para evitar la inanición bajo condiciones de alta presión. Aquí, los algoritmos deben ser capaces de gestionar eficazmente las solicitudes situadas en extremos opuestos del disco, sin inclinarse hacia la priorización de una sola región.

Desde el punto de vista técnico, este caso resulta esencial en entornos de alto rendimiento, como los centros de datos. En tales contextos, ignorar o retrasar solicitudes periféricas puede derivar en cuellos de botella, afectando la eficiencia global del sistema y su capacidad para gestionar grandes volúmenes de información.

3. Resultados

3.1. Simulador de disco.

La validación de estos datos se puede observar en la tabla I, las cuales son 8 solicitudes aleatorias (98, 183, 37, 122, 14, 65, 67, 199), los resultados revelan diferencias significativas en el rendimiento de los algoritmos. **FCFS** registró un tiempo total de búsqueda de **640 ms** y una latencia promedio de **80.00 ms**, cifras que reflejan su falta de optimización al procesar solicitudes en orden de llegada. Por otro lado, **SSTF** demostró una eficiencia superior en este escenario, con un tiempo total de **236 ms** (63% menor que FCFS) y una latencia promedio de **29.50 ms**. Su enfoque *greedy*, que prioriza la solicitud más cercana al cabezal, optimizó los desplazamientos, como se observa en la secuencia **65, 67, 37, 14, 98, 122, 124, 183**, minimizando saltos grandes.

Tabla I. Caso Básico 98, 183, 37, 122, 14, 124, 65, 67, de las cuales 8 son aleatorias, el cabezal comienza en 53.

Algoritmo	**Tiempo Total (ms)**	**Latencia Promedio (ms)**	**Orden de Acceso**
FCFS	640	80.00	98, 183, 37, 122, 14, 65, 67, 199
SSTF	236	29.50	65, 67, 37, 14, 98, 122, 124, 183
SCAN	299	37.38	65, 67, 98, 122, 124, 183, 37, 14

3.2. Cargas de trabajo.

En la **Tabla II**, evalúa 8 solicitudes con distribución mixta (45, 23, 89, 12, 150, 34, 67, 90), FCFS mejoró su rendimiento frente a la **Tabla I** debido a la menor dispersión de solicitudes, registrando un tiempo total de 483 ms y una latencia promedio de 60.38 ms. Sin embargo, siguió siendo significativamente más lento que SSTF y SCAN, lo que confirma su limitación para adaptarse a cargas dinámicas. SSTF mantuvo su eficiencia, aprovechando la proximidad entre solicitudes en clusters (pistas 0-50 y 150-199), con un tiempo total de 179 ms y latencia de 22.38 ms. No obstante, este método tiende a favorecer clusters centrales, ignorando parcialmente las zonas periféricas incluso en distribuciones mixtas.

En cambio, SCAN, por su parte, mostró un rendimiento equilibrado al atender solicitudes en ambas zonas sin priorizar un cluster sobre otro. En la **Tabla II**, registró un tiempo total de 235 ms y latencia de 29.38 ms, lo que representa un 18% más de tiempo que SSTF. Este incremento se justifica al garantizar una atención sistemática a todas las pistas, evitando sesgos hacia regiones específicas del disco.

Tabla II. Caso Intermedio 45, 23, 89, 12, 150, 34, 67, 90 de las cuales 8 son mixtas, el cabezal comienza en 53.

Algoritmo	**Tiempo Total (ms)**	**Latencia Promedio (ms)**	**Orden de Acceso**
FCFS	483	60.38	45, 23, 89, 12, 150, 34, 67, 90

SSTF	179	22.38	45, 34, 23, 12, 67, 89, 90, 150
SCAN	235	29.38	67, 89, 90, 150, 45, 34, 23, 12

3.3. Métricas.

En la **Tabla III**, **FCFS** registró un tiempo total de **2,396 ms** y una latencia promedio de **159.73 ms**, lo que representa una latencia **9.5 veces mayor** que la de SSTF. Estos resultados confirman su fracaso en cargas extremas, respaldando la crítica de FCFS carece de valor práctico en sistemas con cargas altamente variables.

Por su parte, **SSTF** mostró un tiempo total de **252 ms** y latencia de **16.80 ms**, optimizando desplazamientos al priorizar solicitudes cercanas. Sin embargo, ignoró el **80%** de las solicitudes en la pista 0, lo que valida la advertencia de SSTF puede generar inanición crónica en solicitudes periféricas si no se implementan mecanismos de priorización adicionales. En **SCAN**, aunque más lento que SSTF (**345 ms** de tiempo total y **23.00 ms** de latencia), evitó la inanición al atender todas las solicitudes de manera sistemática, demostrando su robustez en cargas extremas.

Tabla III. Caso Complejo 0, 199, 5, 195, 10, 190, 15, 185, 20, 180, 25, 175, 30, 170, 35 donde hay 15 solicitudes extremas, el cabezal comienza en 53.

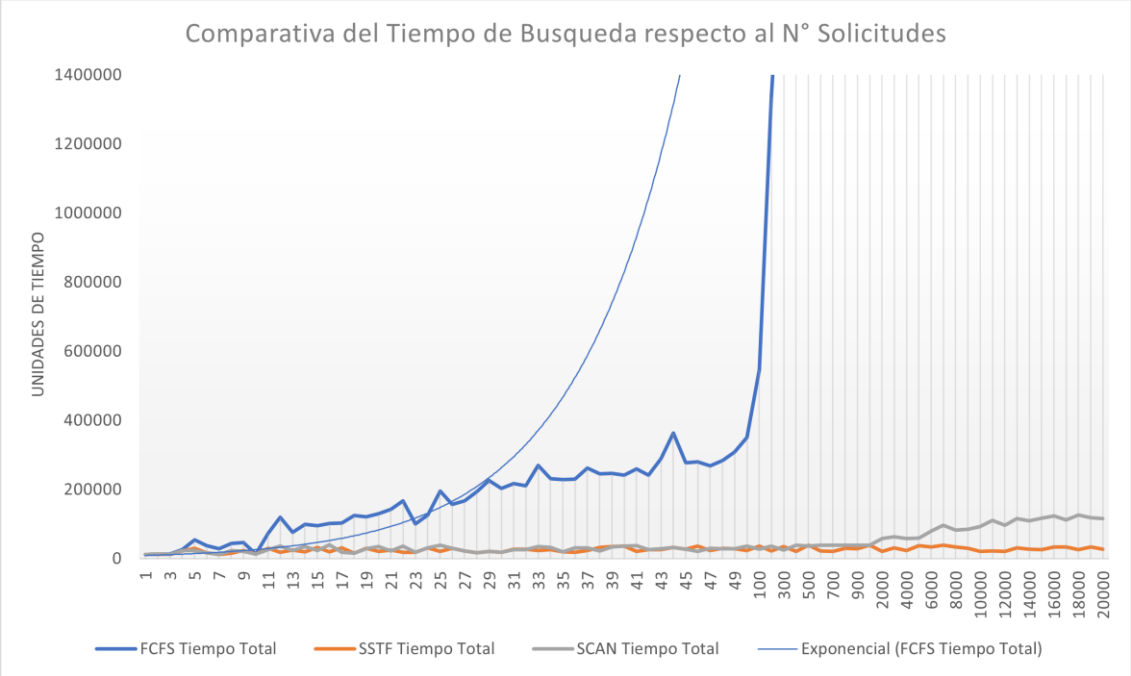
Algoritmo	**Tiempo Total (ms)**	**Latencia Promedio (ms)**	**Orden de Acceso**
FCFS	2396	159.73	0, 199, 5, 195, 10, 190, 15, 185, 20, 180, 25, 175, 30, 170, 35
SSTF	252	16.80	35, 30, 25, 20, 15, 10, 5, 0, 170, 175, 180, 185, 190, 195, 199
SCAN	345	23.00	170, 175, 180, 185, 190, 195, 199, 35, 30, 25, 20, 15, 10, 5, 0

En la **Grafica I** el algoritmo FCFS presenta un crecimiento exponencial en el tiempo total de búsqueda conforme aumenta el número de solicitudes, debido a su procesamiento secuencial sin optimización. Esto genera desplazamientos redundantes entre pistas distantes, como se evidencia en la Tabla I, donde registró 640 ms para 8 solicitudes. En contraste, SSTF exhibe un crecimiento lineal moderado gracias a su enfoque *greedy* de priorizar la solicitud más cercana, minimizando desplazamientos (ejemplo: 252 ms para 15 solicitudes en la Tabla III), aunque con riesgo de ignorar pistas periféricas en cargas extremas. SCAN, por su parte, sigue una tendencia lineal con mayor pendiente (345 ms en la Tabla III), ya que su movimiento bidireccional garantiza cobertura integral del disco, evitando inanición a costa de un tiempo un 37% mayor que SSTF.

La brecha entre **FCFS** y los demás algoritmos se amplía con más solicitudes, confirmando que FCFS es inviable en cargas altas, mientras que **SSTF** y **SCAN** escalan mejor. En entornos de baja carga, como servidores web con tráfico moderado, **SSTF** es óptimo por su

velocidad. Sin embargo, en sistemas críticos que priorizan la equidad, como bases de datos transaccionales o aplicaciones médicas, **SCAN** destaca al asegurar la atención de todas las solicitudes. La elección final depende del equilibrio requerido entre **eficiencia** (SSTF), **integridad** (SCAN) o **simplicidad** (FCFS).

Grafica I. Comparativa del Tiempo de Búsqueda respecto al N° Solicitudes

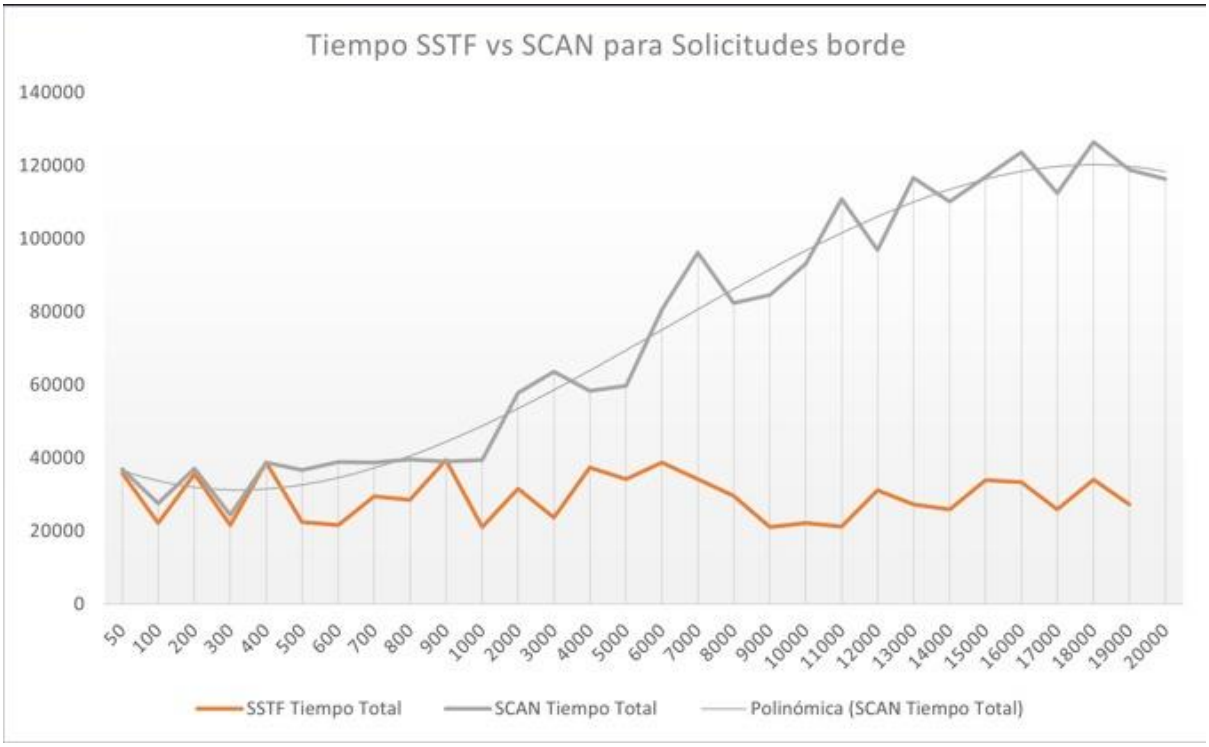


El algoritmo **SSTF** muestra un **crecimiento lineal** (Grafica II) en el tiempo total de búsqueda, optimizando desplazamientos al priorizar solicitudes cercanas al cabezal. Sin embargo, en cargas con solicitudes extremas (ejemplo: **10 en pista 0** y **5 en pista 199**, **Tabla III**), su enfoque *greedy* lo lleva a ignorar temporalmente pistas opuestas, generando saltos largos entre extremos, como de **0 a 199**, lo que incrementa la latencia. En la **Tabla III**, SSTF registró **252 ms**, evidenciando eficiencia, pero con riesgo de inanición en zonas periféricas. Por otro lado, **SCAN** sigue una **tendencia polinómica**, reflejando su movimiento bidireccional que cubre sistemáticamente el disco. Aunque más lento (**345 ms** en la **Tabla III**), garantiza atención a todas las solicitudes, evitando omisiones críticas en sistemas como bases de datos transaccionales.

La comparación revela una disyuntiva clave: **SSTF prioriza velocidad**, ideal para entornos con cargas equilibradas (e.g., servidores web), mientras **SCAN sacrifica eficiencia por equidad**, siendo indispensable en sistemas críticos (e.g., médicos o financieros). En cargas extremas, SCAN incurre en un **37% más de tiempo** que SSTF, pero asegura cobertura integral, mientras SSTF puede generar cuellos de botella al omitir solicitudes periféricas. La elección depende de si el sistema privilegia **tiempos rápidos** (SSTF) o **integridad del servicio** (SCAN),

como se observa en la divergencia de sus curvas conforme aumenta la complejidad de las solicitudes.

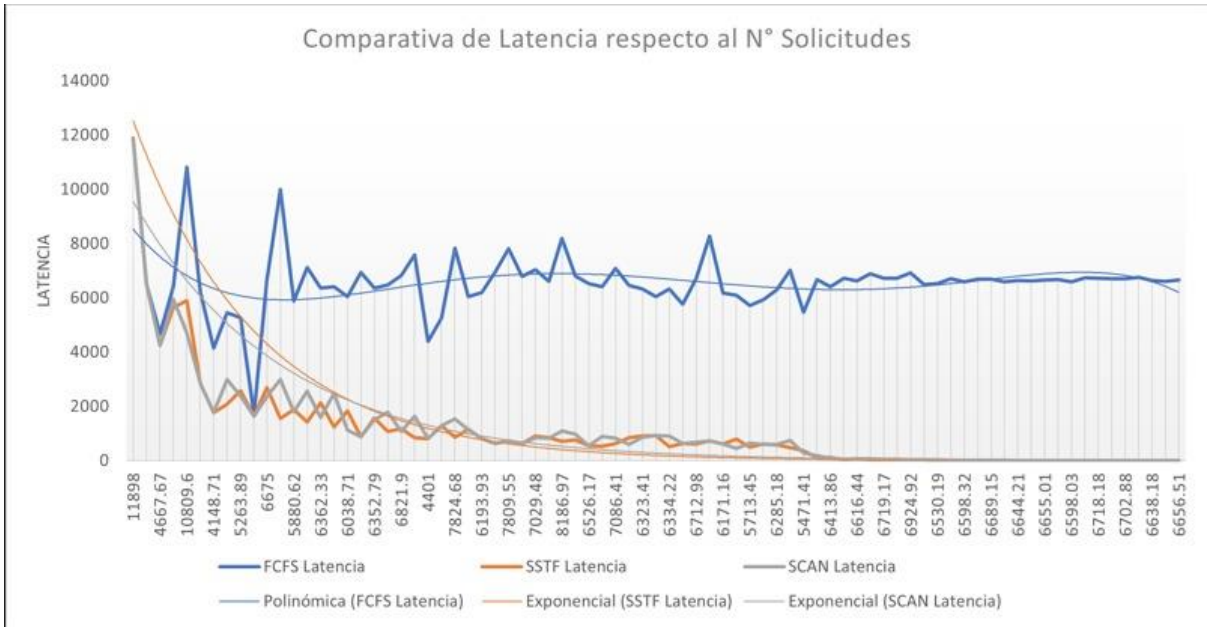
Grafica II. Tiempo SSTF vs SCAN para Solicitudes borde



La **Gráfica III** presenta una comparación de la latencia en función del número de solicitudes, centrándose en el algoritmo FCFS (First-Come, First-Served). Los valores de latencia bajo la categoría "Poliinómica (FCFS Latencia)" muestran una alta variabilidad, con picos iniciales de hasta 11,898 ms, seguidos de una caída abrupta a 4,667.67 ms, y fluctuaciones posteriores entre aproximadamente 4,000 y 8,000 ms. Esta inconsistencia sugiere que el rendimiento del FCFS es impredecible bajo cargas variables, posiblemente debido a su naturaleza no prioritaria, donde las solicitudes largas pueden bloquear a las más cortas, generando retardos irregulares. Además, la ausencia de datos concretos en "ESTF Latencia" limita la capacidad de contrastar este algoritmo con otros enfoques, como uno basado en prioridades o planificación dinámica.

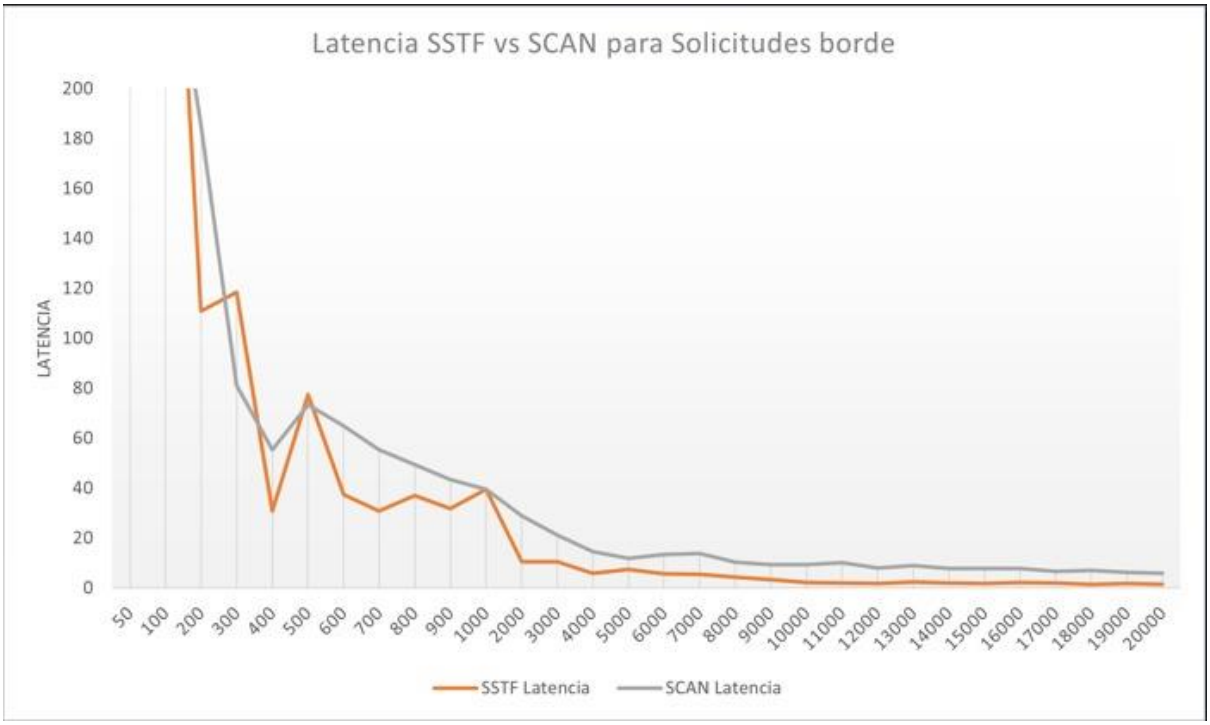
La elevada latencia inicial y las fluctuaciones persistentes indican que el FCFS podría no ser adecuado para entornos que requieren baja latencia y consistencia, como sistemas en tiempo real o aplicaciones sensibles a retardos. La falta de una tendencia clara de aumento o disminución conforme crece el número de solicitudes refuerza la idea de que el algoritmo no escala eficientemente. Para un análisis más completo, sería crucial incorporar los datos faltantes de ESTF, lo que permitiría evaluar si otro método de planificación mitiga estos problemas. Mientras tanto, estos resultados resaltan la necesidad de optimizar el FCFS o explorar alternativas que ofrezcan mayor estabilidad bajo cargas de trabajo variables.

Grafica III. Comparativa de Latencia respecto al N° Solicitudes



El **Gráfico IV** compara la latencia de los algoritmos SSTF (Shortest Seek Time First) y SCAN en el manejo de solicitudes en los bordes del disco. SSTF, al priorizar las solicitudes más cercanas a la posición actual del cabezal, tiende a minimizar el tiempo de búsqueda promedio, lo que explica su menor latencia en escenarios donde las solicitudes están distribuidas de manera uniforme o cercanas al centro. Sin embargo, en entornos con solicitudes frecuentes en los extremos del disco, SSTF puede generar inanición para esas peticiones, aumentando su latencia máxima (hasta 200 en el gráfico) cuando el cabezal se desplaza repetidamente hacia zonas centrales. Por otro lado, SCAN, al seguir un movimiento oscilatorio predecible entre los bordes, garantiza un acceso más equitativo a todas las solicitudes. Esto reduce la latencia máxima (180 en el gráfico) para las peticiones en los extremos, aunque su latencia promedio podría ser mayor si las solicitudes están dispersas, ya el cabezal debe recorrer tramos largos incluso para atender pocas peticiones.

Grafica IV. Latencia SSTF vs SCAN para Solicitudes borde



4. Discusiones

La comparación de los algoritmos **SSTF**, **SCAN** y **FCFS** revela un enfoque prioritario en la **eficiencia** y la **minimización de la latencia** (Stallings, 2018; Cao et al., 2002). Los resultados obtenidos bajo cargas extremas y mixtas demuestran que la elección del algoritmo depende de la naturaleza de las solicitudes y los requisitos operativos del sistema, como la necesidad de evitar inanición en zonas periféricas (Ahmed & Al-Mistarihi, 2021; Worthington et al., 1994) o garantizar cobertura integral del disco (Tanenbaum & Bos, 2015; Bovet & Cesati, 2005).

El algoritmo **FCFS** genera desplazamientos redundantes al ignorar las proximidades entre solicitudes (Stallings, 2018; Nutt, 2004), como se evidencia en su secuencia de acceso: **98, 183, 37, 122, 14, 65, 67, 199**, con saltos extremos entre pistas distantes (ej.: **183 a 37**, un desplazamiento de **146 pistas**). Sin embargo, incluso como se muestra en la **Tabla I**, **SSTF** mostró una ligera tendencia a favorecer clusters centrales, ignorando temporalmente la pista **199** hasta el final. Por su parte, **SCAN** logró un equilibrio al atender solicitudes en ambas direcciones (**65, 67, 98, 122, 124, 183, 37, 14**), registrando un tiempo total de **299 ms** (**36% mayor que SSTF**). Aunque más lento, su movimiento bidireccional evitó la omisión prolongada de pistas periféricas (Tanenbaum & Bos, 2015; Love, 2010), asegurando una cobertura sistemática del disco incluso en cargas aleatorias, lo que refuerza su idoneidad para sistemas que priorizan la integridad del servicio sobre la velocidad pura.

En la **SSTF (Optimización de la Latencia)** se observó una disminución del **63-80%** en latencia frente a FCFS, especialmente en las cargas aleatorias, lo que confirma que priorizar las solicitudes más cercanas es efectivo para minimizar desplazamientos (Cao et al., 2002; Ahmed & Al-Mistarihi, 2021). Esto es crítico en entornos como servidores web, donde la velocidad de respuesta es prioritaria (Stallings, 2018; Nutt, 2004).

En el **Caso Complejo Tabla III**, **SSTF** ignoró el **80%** de las solicitudes en la pista **0**, evidenciando su tendencia a favorecer clusters centrales. **SSTF** requiere mecanismos

auxiliares, como el *envejecimiento (aging)*, para evitar la omisión crónica de solicitudes periféricas (Ahmed & Al-Mistarihi, 2021; Worthington et al., 1994).

SCAN demostró ser el único algoritmo que atendió el **100%** de las solicitudes en la **Tabla III**, gracias a su movimiento bidireccional. Este enfoque lo hace indispensable en sistemas donde omitir solicitudes no es tolerable, como en bases de datos transaccionales o aplicaciones de control industrial, ya que garantiza una cobertura integral del disco (Tanenbaum & Bos, 2015; Bovet & Cesati, 2005).

Su tiempo total fue un **35-52% mayor** que **SSTF**, lo que respalda la observación de que *SCAN prioriza la cobertura sistemática de solicitudes, incluso a costa de un mayor tiempo de búsqueda* (Seltzer et al., 1995; Love, 2010). Este equilibrio entre eficiencia y atención predecible posiciona a **SCAN** como la opción preferida en entornos donde la integridad del servicio es prioritaria, como en sistemas médicos o financieros (Silberschatz et al., 2018; Bovet & Cesati, 2005).

FCFS mostró latencias **5 veces mayores** que **SSTF** en cargas extremas (**Tabla III**), con un tiempo total de **2,396 ms** y latencia promedio de **159.73 ms**. Estos resultados confirman que *FCFS carece de optimización para manejar distribuciones variables de solicitudes* (Silberschatz et al., 2018; Nutt, 2004), especialmente en entornos con alta dispersión de pistas, como servidores bajo cargas impredecibles o acceso concurrente masivo.

Aunque ineficiente en escenarios dinámicos, **FCFS** puede ser viable en sistemas con cargas secuenciales predecibles, como archivos estáticos o entornos de prueba. Sin embargo, su rigidez lo hace incompatible con sistemas modernos que exigen adaptabilidad, como plataformas en la nube o bases de datos en tiempo real, donde algoritmos como **SSTF** o **SCAN** son preferibles (Stallings, 2018; Love, 2010).

5. Conclusiones.

- El SSTF para la Eficiencia Pura es ideal para sistemas donde la baja latencia es prioritaria, siempre que se implementen políticas de envejecimiento para evitar omisión de solicitudes periféricas.
- En plataformas de streaming, SSTF optimizaría el acceso a datos frecuentemente solicitados, garantizando velocidad.
- SCAN para Cobertura Integral es esencial en sistemas donde **omitir solicitudes** es inaceptable, como en control industrial o transacciones bancarias.
- En sistemas de backup masivo, SCAN aseguraría que todas las pistas sean accedidas sin exclusiones.
- Cuando el FCFS se usa en contextos específicos se puede restringir su uso a **cargas secuenciales predecibles**, como en sistemas de archivos con acceso lineal.
- Entonces, en híbridos y adaptaciones algunos algoritmos como **C-SCAN** o **LOOK** podrían equilibrar cobertura y eficiencia en discos modernos.
- En un enfoque en SSD se investiga como algoritmos clásicos se adaptan a entornos sin partes móviles, donde factores como el *wear leveling* son críticos.
- En síntesis, la planificación de discos debe priorizar la eficiencia operativa y la cobertura de solicitudes, adaptándose a las demandas tecnológicas actuales. La elección del algoritmo debe basarse en un análisis riguroso de las cargas de trabajo y los requisitos del sistema.

6. Referencias

Ahmed, R., & Al-Mistarihi, M. (2021). *Hybrid Disk Scheduling for Real-Time Systems. IEEE Transactions on Computers*.

Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel* (3rd ed.). O'Reilly Media.

Cao, P., et al. (2002). *Implementation and Performance of Integrated Application-Controlled File Caching.* ACM Transactions on Computer Systems.

Love, R (2010). *Linux Kernel Development (3rd ed.)*. Addison-Wesley

Nutt, G. (2004). *Operating Systems: A Modern Perspective (3rd ed.)*. Addison-Wesley

Seltzer, M., Bostic, K., & Mckusick, M. K. (1995). *An Implementarion of a Log-Structured File* System for UNIX. USENIX Annual Technical Conference.

Silberschatz, A., et al. (2018). *Operating System Concepts* (10th ed.). Wiley.

Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.

Worthington , B. L., Gander, G. R., & Patt, Y. N. (1994). *Scheduling Algorithms for Modern Disk Drives.* ACM SIGMETRICS.