Universidad Nacional de Lanús, UNLA

Carrera de Licenciatura en sistemas

Informe de Comparación:

Implementación Secuencial y Concurrente de Algoritmos de Ordenamiento de Árboles Binarios.

Alumno: Fereyro Barnes, Martin Alejandro.

Profesor: Pérez, Nicolas.

Materia: Programación concurrente.

Fecha de entrega: 11/06/2024.

Índice

I. RESUMEN

El informe compara la eficacia de dos enfoques de ordenamiento de árboles binarios: secuencial y concurrente. Se evaluaron los tiempos de ejecución y la eficiencia en el uso de recursos. Los resultados indican que el enfoque concurrente mejora significativamente con grandes volúmenes de datos, aunque se observaron anomalías en el rendimiento que podrían atribuirse a la optimización del sistema operativo y la dinámica del procesador.

II. INTRODUCCIÓN

El ordenamiento de datos es esencial en ciencias de la computación y tiene una amplia gama de aplicaciones. En este estudio, comparamos dos enfoques para el ordenamiento de árboles binarios: secuencial y concurrente. Nuestro objetivo es determinar cuál de estos métodos ofrece un mejor rendimiento, especialmente a medida que aumenta el tamaño del conjunto de datos. Este análisis proporcionará una comprensión más clara sobre qué enfoque es más adecuado para diferentes necesidades computacionales.

III. METODOLOGIA

Detalles de Procesador

Tabla 1

Procesador de pruebas.

Características	**Detalle**
Generación	11th Gen
Marca	Intel(R)
Modelo	Core(TM) i5-1135G7
Frecuencia Base	2.40 GHz

Nota: Características del procesador con las cuales se hicieron las pruebas

3.1 Implementación de los Algoritmos en Java

Para llevar a cabo este estudio, se implementaron dos algoritmos de ordenamiento de árboles binarios en el lenguaje de programación Java. La implementación se realizó en un entorno de desarrollo integrado (IDE) compatible con Java, como IntelliJ IDEA o Eclipse.

3.2 Generación de Conjuntos de Datos

Se emplearon dos métodos para generar conjuntos de datos para la evaluación de los algoritmos:

1. **Carga Manual:** Se permitió al usuario especificar la cantidad de valores que se cargarían manualmente en el árbol binario y llenar cada uno de ellos.

2. **Carga Aleatoria:** Se desarrolló una función que generaba de forma aleatoria conjuntos de datos de tamaños variables. Para ello, se utilizó el método **random.nextInt(n)** de la clase **java.util.Random**, donde 'n' representa el tamaño del conjunto de datos, vinculado con un “for” para mayor eficiencia.

3.3 Medición de los Tiempos de Ejecución

Antes de iniciar el proceso de ordenamiento, se utilizó el método **System.nanoTime()** para obtener el tiempo actual del sistema en nanosegundos. Este tiempo se registró como el punto de inicio del proceso de ordenamiento.

3.4 Ejecución de los Algoritmos Concurrente y Secuencial

Una vez cargados los datos en el árbol binario, se procedió a instanciar y ejecutar los algoritmos de ordenamiento concurrente y secuencial. Para la versión concurrente, se crearon múltiples hilos para aprovechar la concurrencia y la capacidad de los sistemas de hilos.

3.5 Medición de los Tiempos de Ejecución

Tras la finalización del proceso de ordenamiento, se volvió a utilizar el método **System.nanoTime()** para registrar el tiempo actual del sistema en nanosegundos. La diferencia entre este tiempo y el tiempo registrado al inicio del proceso proporcionó el tiempo total de ejecución del algoritmo.

3.6 Tamaños de los Conjuntos de Datos

Se llevaron a cabo pruebas utilizando conjuntos de datos de tres tamaños diferentes: mil, diez mil y cien mil elementos numéricos. Estos tamaños se seleccionaron

para abarcar una gama de escenarios de carga de datos y permitir una comparación significativa entre los algoritmos en diferentes niveles de complejidad.

IV. RESULTADOS

4.1 Descripción del Entorno de Pruebas

Las pruebas se realizaron en un sistema con un procesador de velocidad promedio de 1.70 GHz. Se evaluaron los tiempos de ejecución de los algoritmos de ordenamiento de árboles binarios en sus formas concurrente y secuencial, utilizando conjuntos de datos de diferentes tamaños (mil, diez mil y cien mil elementos).

4.2 Resultados de la Ejecución Concurrente

El algoritmo concurrente se ejecutó utilizando dos hilos, y los tiempos de ejecución fueron los siguientes:

- **Mil números:** 3.865900 nanosegundos
- **Diez mil números:** 8.785300 nanosegundos
- **Cien mil números:** 3.953700 nanosegundos

Es notable que al ordenar cien mil números, el tiempo de ejecución fue menor que al ordenar diez mil números, lo cual sugiere un comportamiento anómalo posiblemente debido a la capacidad de optimización del sistema operativo o del hardware. Durante esta prueba, se observó que la velocidad del procesador se incrementó significativamente hasta alcanzar los 3.70 GHz, lo que puede haber contribuido a la reducción del tiempo de ejecución.

4.3 Resultados de la Ejecución Secuencial

El algoritmo secuencial mostró los siguientes tiempos de ejecución:

- **Mil números:** 3.178100 nanosegundos
- **Diez mil números:** 2.04847E7 nanosegundos
- **Cien mil números:** 1.29965E8 nanosegundos

En comparación, el algoritmo secuencial mostró un incremento considerable en el tiempo de ejecución a medida que aumentaba el tamaño del conjunto de datos. La velocidad del procesador también se incrementó durante estas pruebas, alcanzando los 3.00 GHz al ordenar diez mil números y 3.60 GHz al ordenar cien mil números.

4.4 Análisis Comparativo

4.4.1 Rendimiento

Hasta los mil datos, el algoritmo secuencial mostró tiempos de ejecución ligeramente mejores que el concurrente. Sin embargo, al superar este umbral, el algoritmo concurrente demostró ser más eficiente, especialmente notable al comparar los tiempos para ordenar diez mil y cien mil números. Esta mejora en el rendimiento concurrente puede atribuirse a la capacidad de aprovechar los múltiples núcleos del procesador, reduciendo así el tiempo total de ejecución.

4.4.2 Utilización del Procesador

Se observó un comportamiento interesante en la utilización del procesador durante las pruebas. En ambas metodologías, la velocidad del procesador aumentó significativamente durante la ejecución de las tareas de ordenamiento, lo cual es un indicativo de la carga de trabajo intensiva que estas operaciones imponen al sistema. En particular, la ejecución concurrente para cien mil números resultó en un incremento sustancial de la velocidad del CPU a 3.70 GHz, lo que puede haber contribuido a los tiempos de ejecución más bajos observados.

4.4.3 Interpretación de los Resultados

Los resultados obtenidos sugieren que la concurrencia ofrece una ventaja significativa en escenarios con grandes volúmenes de datos. Sin embargo, es importante tener en cuenta las posibles anomalías y optimizaciones que pueden ocurrir en sistemas de hardware y software, como el aumento dinámico de la velocidad del procesador, que pueden influir en los tiempos de ejecución medidos.

En resumen, el algoritmo concurrente se muestra como una opción más eficiente para el ordenamiento de árboles binarios cuando se trata de conjuntos de datos grandes, mientras que el algoritmo secuencial puede ser más adecuado para conjuntos de datos más pequeños.

V. DISCUSIÓN

5.1 Comportamiento Anómalo en la Ejecución Concurrente

Uno de los hallazgos más sorprendentes fue el comportamiento anómalo observado en los tiempos de ejecución del algoritmo concurrente al ordenar cien mil números. A pesar de la mayor complejidad esperada, el tiempo de ejecución fue menor que al ordenar diez mil números. Esta anomalía podría explicarse por varios factores:

1.1.1 Optimización del Sistema Operativo

Los sistemas operativos modernos optimizan la asignación de recursos y pueden priorizar tareas de alto rendimiento, lo que podría haber beneficiado la ejecución concurrente.

1.1.2 Gestión de Caché

La estructura de acceso a la memoria caché puede influir significativamente en los tiempos de ejecución. Un acceso más eficiente al caché en la prueba de cien mil números podría haber reducido los tiempos.

1.1.3 Aceleración Dinámica del CPU

La tecnología de turbo boost puede haber aumentado temporalmente la velocidad del procesador para manejar la carga de trabajo intensiva, como se observó con el incremento a 3.70 GHz.

5.2 Impacto de la Estructura de Datos

La estructura de los datos en el árbol binario juega un papel crucial en el rendimiento de los algoritmos. Los árboles binarios balanceados pueden mejorar la eficiencia del ordenamiento, ya que las operaciones de inserción y búsqueda tienen una complejidad promedio de O(log n). Sin embargo, si el árbol está desequilibrado, estas operaciones pueden degradarse a O(n). En el contexto concurrente, mantener el balance del árbol es aún más crítico, ya que la concurrencia puede introducir complejidades adicionales en la gestión del estado del árbol, que pueden ser bien manejadas o traer más complicaciones según este desarrollado el algoritmo.

5.3 Ventajas del Enfoque Concurrente

El enfoque concurrente muestra ventajas claras en términos de escalabilidad y manejo de grandes volúmenes de datos. A medida que el tamaño del conjunto de datos aumenta, la capacidad de distribuir la carga de trabajo entre múltiples hilos permite una mejor utilización de los recursos del sistema. Esta distribución reduce el tiempo total de ejecución y mejora la eficiencia global del sistema.

5.4 Desventajas del Enfoque Concurrente

Aunque la programación concurrente ofrece numerosas ventajas, también presenta importantes desventajas. La programación concurrente es significativamente más compleja que la secuencial, ya que los desarrolladores deben entender cómo

interactúan los procesos para evitar conflictos y asegurar que se ejecuten en el orden correcto. Esta complejidad se incrementa con el número de procesos concurrentes.

Además, la sincronización adecuada de procesos es crucial para evitar interferencias, especialmente al acceder a recursos compartidos. Implementar mecanismos de sincronización, como semáforos o bloqueos, es necesario, pero añade complejidad al código y puede dar lugar a errores difíciles de detectar y corregir.

5.5 Influencia del Hardware en los Resultados

El rendimiento de los algoritmos de ordenamiento, tanto secuenciales como concurrentes, está fuertemente influenciado por las características del hardware en el que se ejecutan. La capacidad del CPU para manejar múltiples hilos es un factor crucial en la eficiencia de los algoritmos concurrentes. Los procesadores modernos, con múltiples núcleos y capacidades de hyper-threading, pueden ejecutar varios hilos simultáneamente, distribuyendo la carga de trabajo de manera más eficaz y reduciendo significativamente los tiempos de ejecución en comparación con los procesadores más antiguos o menos potentes.

La velocidad del procesador también juega un papel fundamental. Un CPU con mayor velocidad de reloj puede procesar instrucciones más rápidamente, lo que se traduce en un rendimiento superior para ambos tipos de algoritmos. Sin embargo, en escenarios de alta carga, la velocidad del procesador puede variar dinámicamente debido a tecnologías como el turbo boost, que aumentan temporalmente la velocidad de reloj para mejorar el rendimiento en tareas intensivas. Este comportamiento puede llevar a resultados aparentemente anómalos, como se observó en la ejecución concurrente con cien mil números, donde el tiempo de ejecución fue menor de lo esperado debido a un incremento sustancial en la velocidad del CPU.

VI. CONCLUSIÓN

En este estudio, se compararon dos versiones de algoritmos de ordenamiento de árboles binarios: uno secuencial y otro concurrente. Los resultados obtenidos revelaron diferencias significativas en el rendimiento de ambos enfoques, especialmente en relación con el tamaño del conjunto de datos y la capacidad del sistema para aprovechar la concurrencia.

El análisis de los datos mostró que, hasta un cierto umbral (aproximadamente hasta los mil datos), el algoritmo secuencial demostró tiempos de ejecución ligeramente

mejores que su contraparte concurrente. Sin embargo, a medida que el tamaño del conjunto de datos aumentó, la versión concurrente mostró una mejora significativa en el rendimiento, superando al enfoque secuencial.

El comportamiento anómalo observado en los tiempos de ejecución concurrente al ordenar cien mil números sugiere la influencia de factores como la optimización del sistema operativo, la gestión de caché y la aceleración dinámica del CPU. Estas observaciones resaltan la importancia de considerar el entorno de ejecución y las características del hardware al interpretar los resultados.

La ventaja de la concurrencia radica en su capacidad para distribuir la carga de trabajo entre múltiples hilos, lo que permite una mejor utilización de los recursos del sistema y una reducción significativa en los tiempos de ejecución, especialmente en escenarios con grandes volúmenes de datos. Sin embargo, se debe tener en cuenta que la implementación y gestión eficientes de la concurrencia son fundamentales para maximizar sus beneficios y minimizar la sobrecarga asociada.

VII. ACLARACIONES

7.1 **Reopsitorio Git**

Barnes, M. (2024). Trabajo Practico Concurrencia. GitHub. https://github.com/MBarness/TrabajoPracticoConcurrencia.git

7.2 Video explicativo

https://www.youtube.com/watch?v=uaye6aIH6WA&ab_channel=MartinBarnes

VIII. REFERENCIAS

AppMaster. (s.f.). Control de concurrencia. https://appmaster.io/es/glossary/control-de-concurrencia

De la Rosa, A. (2015, febrero). *Investigación sobre bases de datos distribuidas*. Slideshare. https://es.slideshare.net/slideshow/invetigacion-base-de-datos-dis-para-subir/44161901

Marconi, E. (2007). *Ordenamiento y búsqueda en árboles binarios*. [Tesis de grado, Universidad Nacional de La Plata]. SEDICI. https://sedici.unlp.edu.ar/bitstream/handle/10915/3947/Tesis_.pdf?sequence=1&isAllowed=y

Radio Michi. (s.f.). Ventajas y desventajas de la programación concurrente. https://radiomichi.com/ventajas-y-desventajas-de-la-programacion-concurrente/

Santamaría, J., Villalobos, R., & Acuña, G. (2017). Ventajas de la programación concurrente en sistemas distribuidos. Revista Espacios, 38(24). https://www.revistaespacios.com/a17v38n24/17382414.html