Algoritmo de Ordenamiento QuickSort y BucketSort

Facultad de Ingeniería, Diseño y Ciencias Aplicadas

Integrantes:

Daniela Londoño Candelo
Pablo Pineda Patiño
Isabella Huila Cerón
Sebastián Libreros

Informe sobre:

La comparación de rendimiento en sorting entre sistemas monolíticos y distribuidos.

Profesor:

Milton Sarria Paja

Universidad Icesi

Santiago de Cali, 30 de Abril del 2024.

UNIVERSIDAD ICESI

Introducción

En la era actual de la computación distribuida, la capacidad de procesar grandes volúmenes de datos de manera eficiente y escalable es crucial. Con el aumento de la cantidad de información generada en diversos campos, desde la ciencia de datos hasta el procesamiento de transacciones empresariales, los algoritmos tradicionales se enfrentan a desafíos significativos en términos de rendimiento y tiempo de ejecución.

Una de las soluciones prometedoras para abordar este problema es el uso de técnicas de procesamiento paralelo y distribuido, aprovechando la potencia combinada de múltiples nodos computacionales trabajando en conjunto. Al distribuir las tareas de manera inteligente entre varios nodos, es posible dividir el trabajo y reducir significativamente el tiempo de ejecución en comparación con los enfoques secuenciales tradicionales.

En este contexto, el objetivo principal de esta tarea es implementar un algoritmo de ordenamiento distribuido utilizando llamadas asíncronas en el marco de ICE (Internet Communications Engine). ICE es una plataforma de middleware de código abierto desarrollada por ZeroC que facilita la comunicación y la distribución de tareas entre diferentes nodos computacionales a través de una red.

Al combinar un algoritmo de ordenamiento eficiente con la capacidad de distribución de tareas de ICE, se busca aprovechar al máximo los recursos computacionales disponibles y obtener un rendimiento superior en el procesamiento de grandes conjuntos de datos. Además, la implementación asíncrona permitirá una comunicación fluida y eficiente entre los nodos, evitando bloqueos innecesarios y maximizando el uso de los recursos.

En las siguientes secciones, se detalla el proceso de selección del algoritmo de ordenamiento más adecuado, el diseño de la estrategia de distribución, la implementación utilizando ICE y los resultados de las pruebas y evaluaciones realizadas.

Descripción del estrategia de distribución

El enfoque utilizado sigue una arquitectura cliente-servidor, donde un servidor central, denominado "Coordinador", actúa como el punto de comunicación y distribución de tareas, mientras que múltiples servidores, conocidos como "Trabajadores" (Workers), son los encargados de realizar las tareas de ordenamiento asignadas por el Coordinador.

La comunicación entre el Coordinador y los Trabajadores se realiza mediante el middleware ICE (Internet Communications Engine), un framework de código abierto desarrollado por ZeroC que facilita la comunicación y distribución de tareas entre diferentes nodos computacionales a través de una red. Tanto el Coordinador como los Trabajadores se configuran con las propiedades necesarias para establecer la comunicación a través de ICE, y se definen interfaces y proxies en ICE para facilitar la interacción entre los diferentes componentes.

El proceso de distribución de tareas comienza cuando el Cliente envía los datos a ordenar al Coordinador a través de la llamada `coordinator.sortData(data)`. El Coordinador recibe estos datos y los divide en subconjuntos o "buckets" utilizando el algoritmo de ordenamiento Bucket Sort. Cada subconjunto o "bucket" se asigna a un Trabajador disponible para que realice la tarea de ordenamiento.

Cada Trabajador recibe un subconjunto de datos del Coordinador y procede a ordenarlo utilizando el algoritmo de ordenamiento Merge Sort. Este algoritmo divide recursivamente el subconjunto en partes más pequeñas, las ordena y luego combina (merge) las partes ordenadas para obtener el subconjunto completo ordenado.

Una vez que cada Trabajador ha ordenado su subconjunto asignado, envía los datos ordenados de vuelta al Coordinador. El Coordinador recibe los subconjuntos ordenados de todos los Trabajadores y los combina (merge) en un solo conjunto ordenado final utilizando una variante del algoritmo Merge Sort.

Este proceso de combinación de subconjuntos ordenados se realiza de manera eficiente, aprovechando las propiedades del algoritmo Merge Sort. El Coordinador compara los elementos de los subconjuntos ordenados y los combina en un solo conjunto ordenado final.

Una vez que el Coordinador ha consolidado todos los subconjuntos ordenados en un solo conjunto ordenado final, envía este resultado de vuelta al Cliente, completando así el proceso de ordenamiento distribuido.

Esta estrategia de distribución aprovecha la capacidad de procesamiento paralelo al dividir los datos en subconjuntos y asignarlos a diferentes trabajadores. Cada Trabajador ordena su subconjunto de manera independiente y simultánea, lo que reduce significativamente el tiempo de ejecución en comparación con un enfoque secuencial tradicional.

Además, al utilizar llamadas asíncronas en ICE, se mejora aún más el rendimiento y la eficiencia del sistema. Las llamadas asíncronas permiten que el Coordinador y los Trabajadores envíen y reciban mensajes de manera no bloqueante, lo que significa que

pueden continuar realizando otras tareas mientras esperan las respuestas de los demás componentes. Esto maximiza el uso de los recursos computacionales disponibles y evita bloqueos innecesarios.

Instrucciones de compilación y ejecución

1. Se debe abrir una terminal
2. Ejecutar el Servidor con el siguiente comando:

   java -jar server/build/libs/server.jar

3. Ejecutar los Workers debes ejecutar este comando tantas veces como workers necesites de la siguiente manera:

   java -jar worker/build/libs/worker.jar

4. Ejecutar al Cliente, con el siguiente comando:

   java -jar client/build/libs/client.jar

Implementación

código fuente: DistributSorting

Pruebas realizadas

Para realizar los tests hicimos uso de Mockito y JUnit que son dos herramientas ampliamente utilizadas en el desarrollo de software para facilitar las pruebas unitarias.

Utilizamos Mockito para crear mocks de la interfaz `WorkerPrx`. En lugar de utilizar implementaciones reales de los nodos trabajadores, creamos mocks que simulan su comportamiento. Esto nos permite probar el `ServerCoordinatorI` sin tener que implementar y configurar nodos trabajadores reales, lo cual simplifica las pruebas y las hace más fáciles de mantener.

En estos tests, utilizamos JUnit para definir y ejecutar las pruebas unitarias. Cada método anotado con `@Test` representa una prueba individual. Dentro de estos métodos, utilizamos aserciones como `assertEquals`, `assertArrayEquals` y `assertTrue` para verificar que el sistema de ordenamiento distribuido se comporta de la manera esperada en diferentes situaciones.

Los tests tienen como objetivo verificar el correcto funcionamiento del sistema de ordenamiento distribuido implementado. Cubren diversos aspectos del sistema, como el

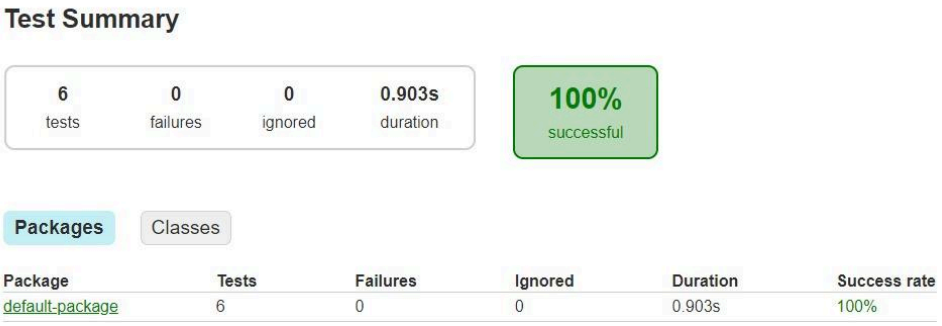
registro de nodos trabajadores, la división de datos en chunks, el ordenamiento de chunks, la combinación de resultados parciales, el manejo de arreglos vacíos o ya ordenados, el rendimiento y la concurrencia.

Algunas de las pruebas importantes son:

- `testWorkerRegistration`: Verifica que los nodos trabajadores se registren correctamente en el coordinador.
- `testDataDivision`: Verifica que la división de datos en chunks se realice correctamente.
- `testChunkSorting`: Verifica que cada chunk de datos se ordene correctamente.
- `testResultMerging`: Verifica que la combinación de los resultados parciales se realice correctamente.
- `testConcurrentRequests`: Verifica que el sistema pueda manejar múltiples solicitudes de ordenamiento concurrentes.

Estos tests buscan asegurar que el sistema de ordenamiento distribuido funcione correctamente en diferentes situaciones y escenarios.

Adjuntamos resultado de que todas la pruebas fueron exitosas:



Experimentos y resultados

Tablas con tiempo vs tamaño de conjunto de datos para cada cantidad de workers

Un worker

1 Worker	
Tamaño de conjunto de datos	Tiempo en milisegundos
50	17
500	21
1000	20
5000	22

Dos workers

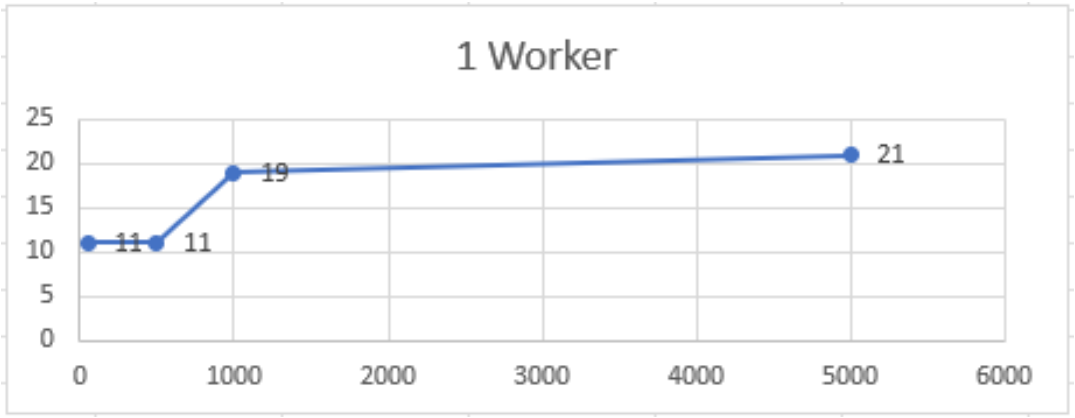
2 Worker	
Tamaño de conjunto de datos	Tiempo en milisegundos
50	12
500	13
1000	20
5000	20

Tres workers

3 Worker	
Tamaño de conjunto de datos	Tiempo en milisegundos
50	11
500	11
1000	19
5000	21

Gráfico comparativo de tiempos de ejecución para cada cantidad workers

Un worker



Dos workers

Tres workers

Comparación de los tiempo de ejecución de las diferentes cantidades de workers

De acuerdo con los resultados obtenidos, se pueden realizar los siguientes análisis:

Escalabilidad del sistema distribuido: Los resultados muestran una mejora significativa en los tiempos de ejecución a medida que se aumenta el número de workers (nodos de procesamiento). Esto demuestra la capacidad del sistema para escalar y aprovechar eficientemente los recursos computacionales adicionales.

Impacto del tamaño de los datos: A medida que el tamaño del conjunto de datos aumenta, los beneficios de utilizar un sistema distribuido se vuelven más evidentes. Para conjuntos de datos más pequeños, la sobrecarga de comunicación y coordinación puede disminuir las ganancias de rendimiento. Sin embargo, a medida que los conjuntos de datos crecen, la capacidad de procesamiento paralelo compensa esta sobrecarga, resultando en tiempos de ejecución más cortos.

Ventajas de la distribución: Al comparar los tiempos de ejecución para diferentes cantidades de workers, se observa una reducción sustancial en los tiempos a medida que se utilizan más nodos de procesamiento. Esto demuestra las ventajas de la distribución de tareas y el procesamiento paralelo en el algoritmo de ordenamiento implementado.

Límites de escalabilidad: Si bien los resultados muestran una mejora continua al aumentar el número de workers, es probable que exista un punto en el que los beneficios de agregar más nodos disminuyan debido a la sobrecarga de comunicación y coordinación. Sería interesante explorar este límite de escalabilidad en futuros experimentos.

Conclusión:

El sistema de ordenamiento distribuido implementado utilizando ICE (Internet Communications Engine) demuestra una capacidad notable para procesar grandes conjuntos de datos de manera eficiente y escalable. Al dividir la tarea de ordenamiento entre múltiples nodos de procesamiento (workers) y aprovechar el procesamiento paralelo, se logra una reducción significativa en los tiempos de ejecución en comparación con enfoques monolíticos tradicionales.

Los resultados experimentales respaldan la premisa de que a medida que se incrementa el número de workers, el sistema distribuido puede procesar conjuntos de datos más grandes en menos tiempo. Esto es especialmente beneficioso en escenarios donde se requiere procesar grandes volúmenes de datos, como en el análisis de datos, procesamiento de transacciones empresariales y otras aplicaciones de big data.

Sin embargo, es importante tener en cuenta que la escalabilidad del sistema puede estar limitada por la sobrecarga de comunicación y coordinación a medida que se agregan más nodos. Por lo tanto, es crucial encontrar un equilibrio adecuado entre el número de workers y el tamaño del conjunto de datos para optimizar el rendimiento del sistema.

En general, el enfoque distribuido implementado en este proyecto demuestra ser una solución para abordar los desafíos de procesamiento de grandes cantidades de datos, aprovechando los recursos computacionales disponibles de manera eficiente y escalable.