Universidad Icesi

Facultad de Ingeniería, Diseño y Ciencias Aplicadas

Departamento de Computadores y Sistemas Inteligentes

Ingeniería de Software IV

Informe:

Comparación de rendimiento en sorting entre sistemas monolíticos y distribuidos.

Integrantes:

Diana Lorena Balanta Solano

Carlos Javier Bolaños Riascos

Danna Alexandra Espinosa Arenas

Cristian Felipe Perafan Chilito

Samuel Adrian Soto Ruiz

Profesores:

Ph.D Gabriel Tamura Morimitsu.

Msc. Alejandro Muñoz Bravo.

Santiago de Cali, 4 de diciembre de 2023.

UNIVERSIDAD ICESI

Tabla de contenido

Introducción

El manejo eficiente de grandes volúmenes de datos se ha convertido en un desafío crítico en la era de la información. En este contexto, la capacidad de ordenar conjuntos masivos de información de manera rápida y efectiva es fundamental para numerosas aplicaciones y sistemas.

El presente informe presenta el diseño, implementación y evaluación de un sistema de ordenamiento distribuido, destinado a abordar esta necesidad creciente de procesamiento escalable. Nuestro objetivo principal reside en explorar estrategias arquitectónicas que permitan el ordenamiento efectivo de grandes cantidades de datos mediante un enfoque distribuido.

El sistema propuesto consta de múltiples componentes diseñados para trabajar en conjunto, aprovechando patrones de diseño y estilos arquitectónicos específicos para optimizar la eficiencia y escalabilidad. A lo largo de este informe, se detalla la estructura general del sistema, sus fundamentos teóricos y la implementación práctica que permite el ordenamiento distribuido de datos.

Además, se llevan a cabo experimentos exhaustivos con el objetivo de evaluar el rendimiento del sistema en diferentes configuraciones y tamaños de conjuntos de datos. La comparación con un enfoque monolítico sirve como punto de referencia fundamental para analizar y cuantificar la mejora de rendimiento obtenida mediante la distribución del proceso de ordenamiento.

En última instancia, este informe proporciona una visión detallada del diseño arquitectónico, la implementación técnica y los resultados experimentales, con el objetivo de demostrar la viabilidad y eficacia del sistema de ordenamiento distribuido en la gestión eficiente de grandes volúmenes de datos.

Descripción del sistema

Arquitectura general del sistema

La arquitectura de nuestro sistema distribuido para el ordenamiento de datos se focaliza en la eficiencia del procesamiento paralelo, la gestión óptima de recursos y la coordinación efectiva entre componentes clave. Para alcanzar estos objetivos, hemos integrado diversos elementos que trabajan en conjunto para optimizar el rendimiento y la capacidad de manejar grandes volúmenes de datos. En el corazón de nuestra arquitectura se encuentra un enfoque Master-Slave, donde un componente maestro orquesta la distribución de tareas entre los nodos esclavos. Este enfoque asegura la ejecución paralela de las fases del algoritmo Merge Sort, aprovechando la capacidad de MIMD (Multiple Instruction, Multiple Data) para maximizar el procesamiento concurrente.

Siguiendo un enfoque distribuido, designamos un controlador único encargado de gestionar las solicitudes del cliente y llevar a cabo el proceso de ordenamiento distribuido. El sistema gestiona la carga distribuida mediante la división de la carga de clasificación en fragmentos

asignados a cada nodo, distribuyendo así la responsabilidad de procesar datos específicos. Además, hemos incorporado el paradigma de MapReduce para optimizar el procesamiento paralelo y la gestión eficiente de grandes volúmenes de datos. Esta estrategia implica dividir la carga de clasificación en fragmentos asignados a cada nodo, facilitando el procesamiento en paralelo y reconociendo las siguientes fases del proceso.

Durante la fase de clasificación, implementamos el algoritmo Mergesort, permitiendo que cada nodo (worker) inicie la clasificación tan pronto como extrae datos localmente. Esto minimiza el tiempo de inactividad y reduce la latencia en la transferencia de datos entre nodos. La fase de fusión K-way es supervisada por el controlador (master), quien se encarga de la fusión para garantizar un flujo ordenado de datos que será procesado posteriormente. Para mejorar la comunicación entre nodos y prevenir posibles cuellos de botella, cada nodo almacena localmente los datos necesarios, evitando una dependencia excesiva del líder. Adicionalmente, hemos integrado callbacks para manejar eventos específicos, notificar finalizaciones de tareas o sincronizar acciones entre los distintos componentes. Estos callbacks aseguran la gestión eficiente de eventos relevantes en la arquitectura.

En cuanto a la gestión de la memoria distribuida, hemos considerado la elección de la arquitectura NUMA (Non-Uniform Memory Access). Esta elección se ajusta a las necesidades específicas del sistema, optimizando el acceso a la memoria según la localidad de los datos y ofreciendo flexibilidad en la asignación de recursos.

Implementación

Código fuente

GitHub - DistributedSorting: This repository contains the implementation of a distributed sorting using zeroC and Java

Instrucciones de compilación y ejecución

1. En cada PC Remoto, abre un terminal y crea un directorio llamado Datamining y entra en este directorio.
2. Una vez creados todos los directorios, copia los ficheros del buiild/libs del proyecto, dependiendo del módulo que quieras arrancar en la máquina.
3. En la carpeta Datamining del servidor y de los trabajadores debe estar también el fichero que quieres ordenar.
4. En el servidor, ejecute el siguiente comando:
   java -jar master.jar
   Archivo de configuración del servidor:
   Callback.Server.Endpoints=default -h * -p 9091

   Ice.Warn.Connections=1
5. En los workers, ejecute el siguiente comando

java -jar worker.jar
Archivo de configuración del worker:
CallbackSender.Proxy=callbackSender:default -h hgrid1 -p 9091
Callback.Worker.Endpoints=por defecto -h * -p 9092
Ice.Warn.Connections=1

6. En el cliente, ejecute el siguiente comando
   java -jar client.jar
   Archivo de configuración del cliente:
   CallbackSender.Proxy=callbackSender:default -h hgrid1 -p 9091
   Callback.Client.Endpoints=por defecto -h *
   Ice.Warn.Connections=1
7. En el cliente, introduzca el nombre del fichero a clasificar, en el siguiente formato:
   dist_sorter:<nombre_archivo>.txt

Experimentos y resultados

Tabla con tiempos medidos vs tamaños de conjuntos de datos para las diferentes configuraciones

Configuración 1 (Monolítica)

Tamaño el conjunto de datos (en millones)	Tiempo (milisegundos)
1,194,045	735
2,388,090	1733
3,582,135	2970
4,776,180	3994
5,970,225	4835

Configuración 2 (4 máquinas)

Tamaño el conjunto de datos (en millones)	Tiempo (milisegundos)
1,194,045	1136
2,388,090	2250
3,582,135	2504
4,776,180	3058
5,970,225	4023

Configuración 3 (8 máquinas)

Tamaño el conjunto de datos (en	Tiempo (milisegundos)

millones)	
1,194,045	1153
2,388,090	1678
3,582,135	2300
4,776,180	2914
5,970,225	3862

Configuración 4 (12 máquinas)

Tamaño el conjunto de datos (en millones)	Tiempo (milisegundos)
1,194,045	1025
2,388,090	1843
3,582,135	2240
4,776,180	3481
5,970,225	3696

Gráfico comparativo de tiempos de ejecución para las distintas configuraciones

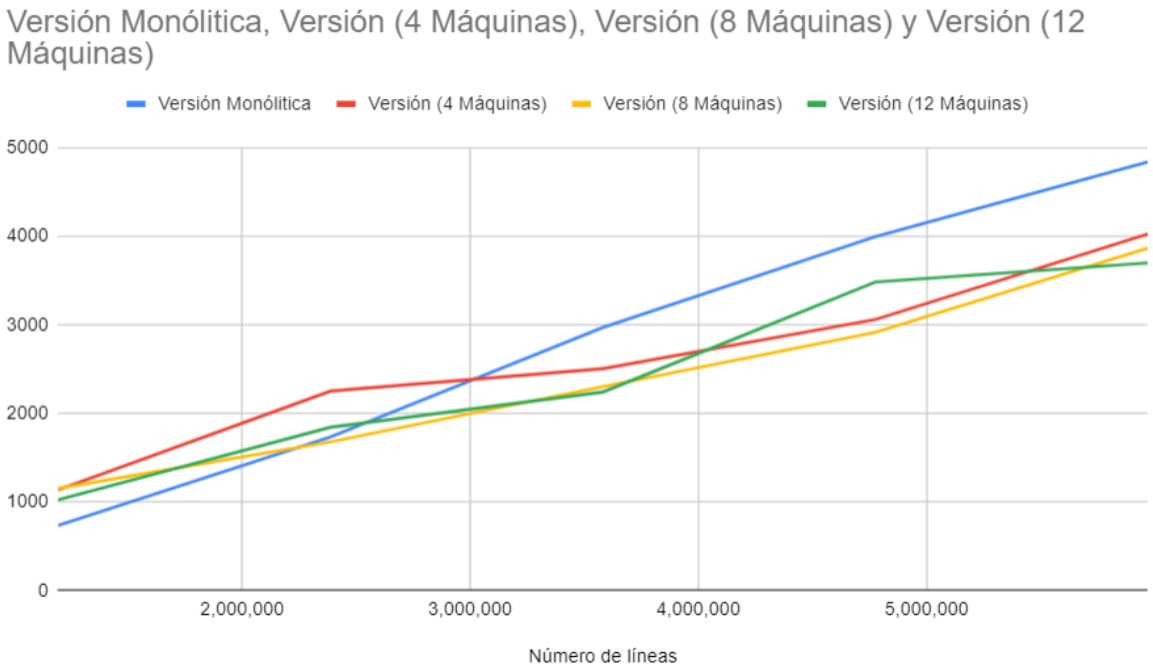
Versión monolítica:

Versión 4 nodos de procesamiento:

Versión 8 nodos de procesamiento:

Versión 12 nodos de procesamiento:

Comparativa de todas las versiones de la implementación:



Herramientas utilizadas

Tecnologías

GitHub: Utilizado como plataforma de control de versiones para el desarrollo colaborativo del código fuente. GitHub proporciona herramientas para el control de versiones, seguimiento de problemas (issues), gestión de ramas (branches), colaboración entre desarrolladores y almacenamiento del código fuente de manera segura en la nube.

Lenguajes

Java: Utilizado como el lenguaje principal de programación para el desarrollo del sistema. Java es conocido por su portabilidad, orientación a objetos y su capacidad para construir sistemas escalables.

Framework

ZeroC Ice (ZeroIce): Utilizado como framework de comunicación para facilitar la interacción entre componentes distribuidos en el sistema. ZeroIce es una plataforma que permite la comunicación entre diferentes partes de un sistema distribuido, proporcionando mecanismos para el intercambio de datos entre componentes remotos de manera eficiente y segura.

Resultados y Conclusiones

- **Eficiencia del Procesamiento Paralelo:**

  La adopción de una arquitectura distribuida, con un enfoque Master-Slave y la implementación de los algoritmos merge y k-merge sorting, demuestra una eficiencia significativa en el procesamiento paralelo. La capacidad de ejecutar tareas de clasificación y fusión en paralelo contribuye a la reducción efectiva de los tiempos de ejecución.

- **Escalabilidad del Sistema:**

  La escalabilidad del sistema se evidencia en las configuraciones 2, 3 y 4, donde el tiempo de procesamiento no aumenta proporcionalmente al tamaño del conjunto de datos. Aunque los tiempos no disminuyen de manera lineal con el aumento del número de máquinas, sigue habiendo mejoras significativas en la eficiencia, indicando que el sistema es capaz de escalar para manejar conjuntos de datos más grandes.

- **Rendimiento Comparativo de Configuraciones:**

  La comparación entre las configuraciones 2, 3 y 4 muestra que agregar más máquinas no siempre resulta en una mejora proporcional en el tiempo de procesamiento. Puede haber costos adicionales asociados con la coordinación y comunicación entre un mayor número de nodos, lo que podría influir en el rendimiento.

- **Optimización de Recursos:**

  Aunque agregar máquinas puede mejorar la velocidad de procesamiento, también es esencial tener en cuenta la optimización de recursos. La configuración que utiliza 8 máquinas logra un buen equilibrio en términos de tiempos de ejecución y puede representar una optimización eficiente de recursos

Referencias

ZeroC. (2023). Documentación de Ice 3.7 Middleware. ZeroC Documentation: https://doc.zeroc.com/ice/3.7
Oracle. (2018). Java™ Platform, Standard Edition 11 Documentation. Oracle Documentation: https://docs.oracle.com/en/java/javase/11/
Wikipedia contributors. (2023). K-way merge algorithm. En Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/K-way_merge_algorithm

K-Way Merge Sort. Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2001). *Introducción a los algoritmos*. Prensa del MIT. págs. 28 y 29. ISBN 978-0-262-03293-3.