Trabajo Práctico

Programación de Objetos Distribuidos

Integrantes del grupo:

Felipe Hiba (61219)
Juan Ignacio Matilla (60459)
Patricio Escudeiro (61156)

Informe sobre la Implementación de MapReduce y Análisis de Rendimiento para Consultas 1-4

Introducción

Este informe presenta el diseño e implementación de cuatro consultas MapReduce utilizando Hazelcast para procesar grandes conjuntos de datos de multas de estacionamiento, infracciones y agencias de la Ciudad de Nueva York (NYC) y Chicago (CHI). Cada consulta aborda necesidades específicas de análisis de datos y está optimizada para el rendimiento y la eficiencia de recursos. El informe también discute diseños alternativos considerados, análisis de rendimiento con predicciones sobre escalabilidad, posibles mejoras y comparaciones de técnicas de optimización.

Conjuntos de Datos Utilizados

- ticketsNYC.csv: 15,000,000 registros
- ticketsNYC100k.csv: 100,000 registros
- ticketsCHI.csv: 5,000,000 registros
- ticketsCHI1M.csv: 1,000,000 registros
- infractionsNYC.csv: 97 registros
- infractionsCHI.csv: 128 registros
- agenciesNYC.csv: 32 registros
- agenciesCHI.csv: 6 registros
- ticketsSample.csv: 10 registros, se utilizó para testear que funcionen correctamente la lógica las queries
- agenciesSample.csv: idem que el anterior
- infractiosSample.csv: idem que el anterior

Los subsets de datos se obtuvieron corriendo el siguiente comando:

head -n [rows deseadas] tickets[NYC|CHI].csv > tickets[NYC|CHI][rows deseadas].csv

Query 1: Número Total de Multas por Infracción y Agencia

Diseño de Componentes MapReduce

Mapper (Query1Mapper)

- **Entrada:** Key (String), Value (Ticket)
- **Salida:** Key (infractionDescription;agency), Value (Integer 1)

Decisiones de Diseño:

- **Composición de la Clave:** Combinar infractionDescription y agency en una sola clave asegura que las multas se agrupen por ambos atributos.
- **Emisión del Valor:** Emitir 1 simplifica el proceso de conteo.

- **Objetivo:** Contar el número de multas emitidas para cada infracción por cada agencia.

Combiner (Query1CombinerFactory)

- **Propósito:** Actúa como un mini-reductor para sumar los conteos localmente en cada nodo antes de que los datos se envíen por la red.
- **Decisión de Diseño:** Implementado para reducir el tráfico de red y mejorar el rendimiento.
- **Objetivo:** Agregar conteos localmente y minimizar la transferencia de datos a los reductores.

Reducer (Query1ReducerFactory)

- **Entrada:** Key (infractionDescription;agency), Values (Integer counts)
- **Salida:** Key (infractionDescription;agency), Value (Integer total count)
- **Decisión de Diseño:** Suma los conteos recibidos para obtener el total de multas por clave.
- **Objetivo:** Producir los conteos finales de multas por infracción y agencia.

Collator (Query1Collator)

- **Propósito:** Procesa los resultados reducidos en el lado del cliente, realizando la ordenación y el formato final.
- **Decisiones de Diseño:**
    - **Ordenación:** Los resultados se ordenan por conteo de multas descendente, luego alfabéticamente por infracción y agencia.
- **Objetivo:** Presentar los resultados en el orden y formato requeridos.

Diseños Alternativos Considerados y Descartados

- **No Usar un Combiner:** Inicialmente se consideró omitir el combiner, pero se descartó debido al aumento del tráfico de red.
- **Claves Separadas para Infracción y Agencia:** Se consideró dividir infraction y agency en claves separadas, pero se descartó porque complicaría el proceso de agrupamiento y agregación.

Query 2: Ingresos Acumulados del Año (YTD) por Agencia

Diseño de Componentes MapReduce

Mapper (Query2Mapper)

- **Entrada:** Key (String), Value (Ticket)
- **Salida:** Key (agency;year;month), Value (Double fine amount)

Decisiones de Diseño:

- **Composición de la Clave:** Incluir agency, year y month en la clave para agrupar los ingresos.
- **Emisión del Valor:** Emitir fineAmount para el cálculo de ingresos.
- **Objetivo:** Calcular los ingresos mensuales por agencia.

Combiner (Query2CombinerFactory)

- **Propósito:** Sumar los montos de las multas localmente por clave para reducir la transferencia de datos.
- **Decisión de Diseño:** Implementado para reducir el tráfico de red y mejorar el rendimiento.
- **Objetivo:** Minimizar la cantidad de datos enviados a los reductores.

Reducer (Query2ReducerFactory)

- **Entrada:** Key (agency;year;month), Values (Double fine amounts)
- **Salida:** Key (agency;year;month), Value (Double total monthly revenue)
- **Decisión de Diseño:** Sumar los montos de las multas para obtener el ingreso total por mes para cada agencia.
- **Objetivo:** Calcular los ingresos mensuales por agencia.

Collator (Query2Collator)

- **Propósito:** Calcula los ingresos acumulados del año, maneja la ordenación y filtra los meses con ingresos cero.
- **Decisiones de Diseño:**
    - **Cálculo de YTD:** Acumula los ingresos mensuales para calcular el ingreso YTD.
    - **Ordenación:** Los resultados se ordenan alfabéticamente por agencia, luego cronológicamente por año y mes.
    - **Filtrado:** Excluye meses/años con ingresos cero.
- **Objetivo:** Generar un informe de ingresos acumulados del año por agencia.

Diseños Alternativos Considerados y Descartados

- **Calcular YTD en los Reducers:** Descartado porque los Reducers operan de manera independiente y no pueden mantener el estado a través de diferentes meses.

Query 3: Porcentaje de Placas de Reincidentes por barrio

Diseño de Componentes MapReduce

Mapper (Query3Mapper)

- **Entrada:** Key (String), Value (Ticket)
- **Salida:** Key (county;plate), Value (String infraction code)

Decisiones de Diseño:

- **Filtrado por Rango de Fechas:** El mapper filtra las multas dentro del rango de fechas especificado.
- **Composición de la Clave:** Combina county y plate para agrupar las infracciones por placa en cada barrio.
- **Objetivo:** Recopilar infracciones por placa en cada barrio.

Reducer (Query3ReducerFactory)

- **Entrada:** Key (county;plate), Values (String infraction codes)
- **Salida:** Key (county;plate), Value (Boolean indicating repeat offender)
- **Decisiones de Diseño:**
    - **Conteo de Infracciones:** Cuenta las ocurrencias de cada código de infracción por placa.
    - **Determinación de Reincidentes:** Verifica si algún conteo de infracción alcanza o supera n, en el *finalizeReduce*.
    - **Objetivo:** Identificar placas de reincidentes en cada barrio.

Collator (Query3Collator)

- **Propósito:** Agrega los resultados para calcular el porcentaje de reincidentes por barrio.
- **Decisiones de Diseño:**
    - **Cálculo de Porcentaje:** Calcula los porcentajes utilizando los conteos de placas totales y reincidentes.
    - **Ordenación:** Los resultados se ordenan por porcentaje descendente, luego alfabéticamente por barrio.
    - **Truncamiento de Porcentajes:** Los porcentajes se truncan a dos decimales según los requisitos.
- **Objetivo:** Presentar el porcentaje de placas de reincidentes por barrio.

Diseños Alternativos Considerados y Descartados

- **Usar un Combiner:** Considerado pero descartado porque la naturaleza de la agregación de datos no se beneficiaba de un combiner.

Query 4: Top N Infracciones con Mayor Diferencia en Monto de Multa por Agencia

Diseño de Componentes MapReduce

Mapper (Query4Mapper)

- **Entrada:** Key (String), Value (Ticket)
- **Salida:** Key (infractionCode), Value (Double fine amount)
- **Decisiones de Diseño:**
    - **Filtrado de Agencia:** El mapper emite datos sólo si la agencia emisora de la multa coincide con la agencia especificada.
    - **Composición de la Clave:** Utiliza infractionCode como la clave para agrupar las multas por infracción.

Objetivo: Recopilar montos de multas para cada infracción emitida por la agencia especificada.

Combiner (Query4CombinerFactory)

- **Propósito:** Calcula montos mínimos y máximos parciales de multas por infracción.
- **Decisión de Diseño:** Implementado para reducir la transferencia de datos a los reducers mediante la actualización de valores mínimos/máximos.
- **Objetivo:** Optimizar el rendimiento minimizando el tráfico de red.

Reducer (Query4ReducerFactory)

- **Entrada:** Key (infractionCode), Values (MinMaxPair objects)
- **Salida:** Key (infractionCode), Value (MinMaxPair with overall min and max fine amounts)

Decisiones de Diseño:

- **Agregación:** Combina valores mínimos/máximos parciales para calcular el mínimo y máximo general por infracción.
- **Objetivo:** Determinar los montos mínimos y máximos de multas para cada infracción.

Collator (Query4Collator)

- **Propósito:** Calcula la diferencia entre las multas máximas y mínimas, ordena los resultados y selecciona las N principales infracciones.
- **Decisiones de Diseño:**
    - **Cálculo de Diferencia:** Calcula Diff como Max menos Min.
    - **Ordenación:** Los resultados se ordenan por Diff descendente, luego alfabéticamente por infracción.
    - **Limitación de Resultados:** Emite solo las N principales infracciones según lo especificado.
- **Objetivo:** Generar una lista clasificada de infracciones basada en las diferencias en los montos de multas.

Objetos de Soporte (InfractionDiff, MinMaxPair)

- **InfractionDiff**: Almacena la información sobre la infracción, incluyendo la diferencia calculada entre los montos de multa. Esto permite una representación clara y ordenada para la salida final.
- **MinMaxPair**: Estructura de datos utilizada para mantener y combinar valores de multas mínimas y máximas en el Combiner y Reducer.

Diseños Alternativos Considerados y Descartados

- **No Usar un Combiner:** Descartado porque resultaría en un aumento del tráfico de red y una ejecución más lenta debido al gran volumen de montos de multas.
- **Realizar Todos los Cálculos en el Reducer:** Considerado pero descartado a favor del uso de un combiner para mayor eficiencia.

Análisis de Tiempos de Ejecución

Aclaramos que todos estos test fueron corridos sin la optimización de batching ni cacheo de agencies e infractions.

En general, el rendimiento de MapReduce debería mejorar con más nodos porque la carga de trabajo se distribuye entre más recursos. Al agregar más nodos, Hazelcast puede dividir y procesar las tareas en paralelo en estos nodos, lo que lleva a una reducción del tiempo de ejecución para cada operación, asumiendo que los datos se encuentran bien distribuidos y que no hay cuellos de botella significativos en la red o en la coordinación.

Sin embargo, hay algunos puntos importantes que tener en cuenta:

- Distribución de datos: Si los datos no están distribuidos uniformemente, algunos nodos pueden quedar con más carga que otros, lo que reduce la eficiencia del procesamiento paralelo.
- Sobrecarga de coordinación: Al agregar más nodos, puede aumentar la sobrecarga de comunicación entre nodos y la coordinación de las tareas, lo que podría impactar el rendimiento si no se maneja adecuadamente.
- Tamaño de los datos: Si el conjunto de datos es pequeño, agregar más nodos podría no ofrecer una mejora significativa en el rendimiento debido al tiempo necesario para coordinar la tarea entre nodos.
- Configuración y recursos de hardware: La cantidad de memoria, CPU y ancho de banda de red disponible en cada nodo afectará cómo se benefician del aumento en el número de nodos.

Debido a las limitaciones ambientales, solo se pudieron medir los tiempos de ejecución con diferentes números de nodos a través de vms conectadas mediante una red local. Por lo que tenemos limitaciones de configuración y recursos de hardware a la hora de analizar los datos.

Comportamiento Predicho con el Incremento de Nodos (Hasta 5 Nodos)

Consulta 1

Esta query fue testeada con el dataset de Tickets de Chicago entero, es decir las 5 millones de entradas. Esta query fue testeada con 1 nodo,2 nodos y 5 nodos. Cabe aclarar que la corrida de 1 nodo fue con una computadora con mucho mejores specs que en los otros casos. A continuación se mostrará cuánto tardó en cada caso:

- 1 nodo : 32 minutos.
- 2 nodos: 48 minutos.
- 5 nodos:49 minutos.

La gran diferencia de tiempo observada se puede dar debido a varios motivos, entre ellos, y el que fue mencionado con anterioridad, es que la de 1 nodo se corrió con una computadora con mejores specs, luego otro motivos que puede ser, es que se demoraba por el viaje en la red que tenían que hacer los datos.

Puntos Potenciales de Mejora y Expansión

- Buscar formas de mejorar la lectura de los csv de entrada.
- Crear un entorno de pruebas distribuido más realista, más allá de un solo nodo o red local, para analizar mejor cómo Hazelcast maneja múltiples nodos, comunicación, latencia y escalabilidad en condiciones cercanas a las de un caso de uso real.
- Realizar unit testing para un control más granular del funcionamiento.

Comparación de Tiempos de Ejecución con y sin Combiner

Para esta comparación de tiempos se usó el dataset de ticketsCHI1M.csv, que contiene el primer millón de rows del archivo ticketsCHI.csv

Con Combiner

	1 Nodo	**2 Nodos**
query 1	6m,30s	9m,26s

Sin Combiner

	1 Nodo	**2 Nodos**
query 1	6m,27s	9m,29s

Con Combiner

query 2 : 14m,3s 1 nodo

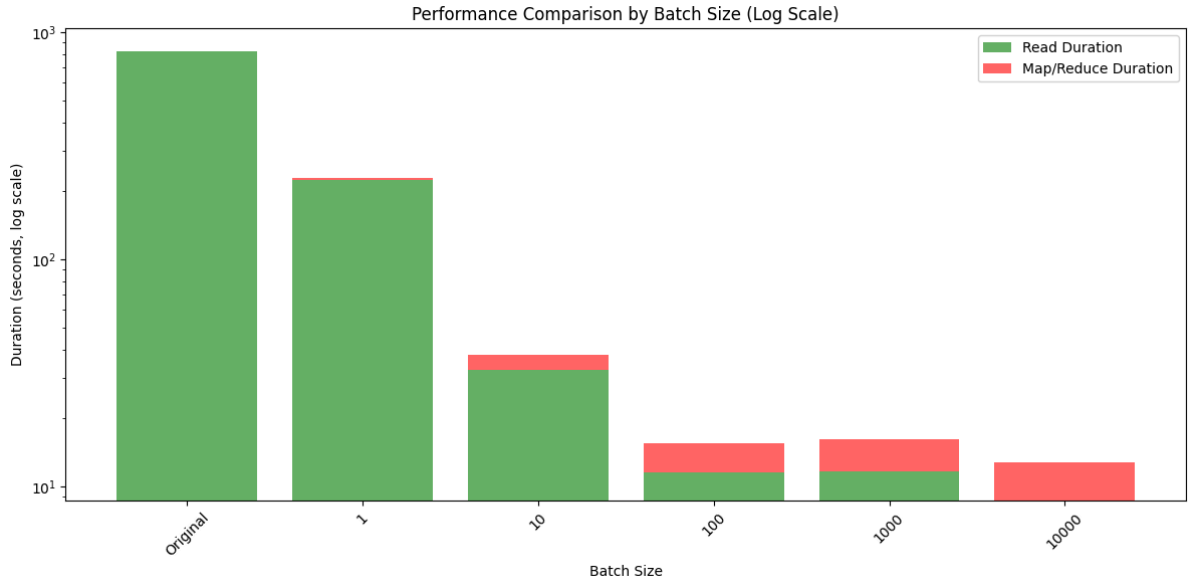
Sin Combiner

query 2 : 14m,16s 1 nodo

El análisis de los resultados muestra que el uso de un Combiner en el proceso MapReduce puede mejorar el rendimiento, especialmente cuando se incrementa el número de nodos. Para la **query 1**, se observa una ligera mejora en el tiempo al usar el Combiner, tanto en el caso de 1 nodo como de 2 nodos. En contraste, para la **query 2**, el Combiner proporciona una mejora marginal cuando se usa 1 nodo, lo que sugiere que el efecto de la combinación de datos para reducir la transferencia de información entre los nodos tiene un mayor impacto en ciertas cargas de trabajo. En general, el uso de Combiners parece ser más efectivo para consultas que involucran grandes volúmenes de datos intermedios.

Optimización con batching y caching de infractions y agencies

Realizamos una optimización de lectura del archivo mediante la técnica de batching y agregamos caching de infractions y agencies para cargar los tickets de manera local en la función, estos fueron los resultados para la query 1, con 1 nodo:



- **Batch Size 'Original' (sin batching ni cacheo):**
  - **Duración de lectura:** 820.87 segundos.
  - **Duración de Map/Reduce:** 4.23 segundos.
  - En este escenario, cada operación de carga de tickets y validación de `agencies` e infractions se realiza de manera independiente, lo que conlleva múltiples operaciones remotas y un uso ineficiente del procesamiento. La duración de lectura es considerablemente más larga debido al alto número de operaciones remotas.
- **Batch Size '1':**
  - **Duración de lectura:** 224.02 segundos.
  - **Duración de Map/Reduce:** 3.71 segundos.

    - La mejora significativa en la duración de la lectura (reducción de aproximadamente 70%) se debe al uso del cacheo de agencies e infractions. Aunque el batching de tamaño 1 no reduce la cantidad de operaciones de inserción de tickets, evita múltiples consultas remotas para validar cada ticket.
- **Batch Size '10':**
    - **Duración de lectura:** 32.57 segundos.
    - **Duración de Map/Reduce:** 5.28 segundos.
    - Con un tamaño de lote de 10, el tiempo de lectura se reduce drásticamente a solo 32.57 segundos. La reducción se debe a que se agrupan varias inserciones de tickets en lotes más grandes, reduciendo las operaciones remotas al servidor distribuido.
- **Batch Sizes '100', '1000', y '10000':**
    - **Batch Size '100':**
        - **Duración de lectura:** 11.51 segundos.
        - **Duración de Map/Reduce:** 3.93 segundos.
    - **Batch Size '1000':**
        - **Duración de lectura:** 11.64 segundos.
        - **Duración de Map/Reduce:** 4.49 segundos.
    - **Batch Size '10000':**
        - **Duración de lectura:** 8.66 segundos.
        - **Duración de Map/Reduce:** 4.15 segundos.
    - A medida que el tamaño del batch aumenta, la duración de la lectura sigue disminuyendo. Con tamaños de lote de 100, 1000 y 10000, el tiempo de lectura se estabiliza en torno a los 8-12 segundos, lo que representa una mejora considerable respecto al escenario original.

2. Impacto del Cacheo de agencies e infractions:

- El cacheo de agencies e infractions reduce significativamente la cantidad de operaciones remotas necesarias para validar los tickets. Esto disminuye la latencia y mejora el tiempo total de procesamiento.
- La diferencia entre el tiempo de ejecución original y el tiempo de ejecución con batching de tamaño 1 destaca el impacto positivo del cacheo: el tiempo de lectura se reduce en más de un 70%, lo que muestra la importancia de evitar múltiples consultas remotas.

3. Análisis General:

- **Disminución del Tiempo de Lectura:** El tiempo de lectura disminuye exponencialmente a medida que aumenta el tamaño del batch. Esto se debe a la reducción del número de inserciones individuales y a la agrupación de las operaciones.
- **Duración del Map/Reduce:** La duración del trabajo de Map/Reduce no muestra variaciones significativas con el cambio de tamaño de lote. Esto indica que la fase de Map/Reduce no se ve afectada por el uso de batching, lo cual es esperado ya que esta operación no depende directamente de cómo se cargan los datos.

- **Mejora Global del Rendimiento:** La combinación de batching y cacheo reduce el tiempo total de procesamiento de manera significativa. El mejor tiempo se obtiene con tamaños de lote grandes, como 10000, donde el tiempo de lectura es mínimo.

Conclusión:

- El uso de batching mejora considerablemente el rendimiento al reducir las operaciones remotas de inserción, lo que resulta en una menor latencia y un menor tiempo de ejecución.
- El cacheo de elementos agencies e infractions complementa esta mejora al reducir las operaciones de búsqueda remota necesarias para validar los tickets.
- A medida que el tamaño del batch aumenta, el tiempo de lectura disminuye de forma drástica hasta estabilizarse, mientras que la duración del trabajo de Map/Reduce permanece constante, lo que sugiere que el cuello de botella inicial estaba en las operaciones remotas y no en el procesamiento del Map/Reduce.