Instituto Tecnológico de Buenos Aires

Trabajo Práctico Especial 2 -

72.42 - Programación de Objetos Distribuidos

Informe Técnico: Diseño e Implementación de modelo de programación MapReduce.

Integrantes

Grupo Número 5

Madero Torres, Eduardo Federico - 59494

Limachi, Desiree Melisa - 59463

Comisión S - Segundo Cuatrimestre del 2024

Introducción

El presente informe detalla el diseño e implementación de un sistema distribuido para el procesamiento de multas de estacionamiento, utilizando el framework Hazelcast y el modelo de programación MapReduce. El sistema procesa datos reales de multas de las ciudades de Nueva York y Chicago, Estados Unidos, abordando diversos análisis estadísticos y agregaciones sobre los conjuntos de datos.

La implementación se fundamenta en la arquitectura MapReduce, que permite procesar grandes volúmenes de datos de manera distribuida y eficiente. El sistema está diseñado para manejar conjuntos de datos significativos: 15 millones de registros para Nueva York y 5 millones para Chicago, junto con sus respectivos catálogos de infracciones y agencias. Cada query implementada representa un desafío específico de procesamiento de datos, desde simples agregaciones hasta análisis más complejos que requieren múltiples etapas de procesamiento.

En las siguientes secciones, se analizará en detalle el diseño de los componentes MapReduce para cada query implementada, las decisiones tomadas, las alternativas consideradas y descartadas, así como un análisis exhaustivo del rendimiento del sistema bajo diferentes configuraciones y optimizaciones. El objetivo es proporcionar una visión completa tanto de los aspectos técnicos de la implementación como de las consideraciones de rendimiento y escalabilidad del sistema.

Diseño de componentes de MapReduce

A. Query 1: Total de multas por infracción y agencia

El diseño implementado para la Query 1 se centró en la obtención del total de multas por cada par infracción-agencia. La arquitectura MapReduce se estructuró en cuatro componentes principales: un Mapper que genera pares clave-valor donde la clave es una cadena compuesta por "infracción:agencia" y el valor es siempre es 1, representando una ocurrencia; un Combiner que realiza una agregación local en cada nodo, sumando las ocurrencias para reducir significativamente el tráfico de red entre nodos; un Reducer que consolida los resultados parciales de todos los nodos para obtener los totales finales; y un Collator que implementa la lógica de ordenamiento requerida, priorizando el orden descendente por cantidad de multas y utilizando el orden alfabético como criterio de desempate.

Una decisión clave fue la utilización de una clave compuesta (infracción:agencia) en lugar de procesar las dimensiones por separado, descartando así la alternativa de implementar dos jobs MapReduce independientes. Esta última aproximación, aunque potencialmente más clara en términos de separación de responsabilidades, fue descartada debido al overhead que introduciría en términos de tiempo de procesamiento y recursos de red.

B. Query 2: Recaudación YTD

El diseño de la Query 2 se enfocó en calcular la recaudación Year-To-Date (YTD) por agencia a lo largo del tiempo. La arquitectura MapReduce se estructuró en cuatro componentes principales: un Mapper que genera pares clave-valor donde la clave sigue el formato "agencia-año-mes" y el valor representa el monto de la multa; un Combiner que realiza una agregación local en cada nodo, sumando los montos por clave; un Reducer que consolida los resultados parciales y mantiene un registro ordenado cronológicamente de los montos por año y mes, calculando el acumulado YTD para cada período; y un Collator que implementa la lógica de ordenamiento y presentación final, organizando los resultados primero alfabéticamente por agencia y luego cronológicamente por año y mes.

Una decisión crucial fue la utilización de estructuras jerárquicas (TreeMap) para mantener el orden natural de los datos temporales y facilitar el cálculo acumulativo del YTD. Se descartó la alternativa de realizar el cálculo YTD en el Reducer para evitar la complejidad de mantener el estado acumulativo en múltiples nodos, optando por centralizar esta lógica en el Collator donde se tiene una visión completa de todos los datos de cada agencia.

C. Query 3: Cálculo de patentes reincidentes por barrio.

Para la Query 3, enfocada en el cálculo de patentes reincidentes por barrio, se implementó una solución que procesa las multas dentro de un rango temporal específico. La arquitectura MapReduce se compone de un Mapper que emite pares ***(barrio, patente)*** para facilitar el conteo de patentes únicas por zona; un Combiner que optimiza el procesamiento mediante la agregación local de conjuntos de patentes únicas por barrio en cada nodo, reduciendo significativamente el tráfico de red; un Reducer que consolida los conjuntos de patentes a nivel global; y un Collator que implementa la lógica, calculando los porcentajes de reincidencia y aplicando el criterio de n infracciones del mismo tipo.

La decisión de utilizar Sets para el manejo de patentes únicas fue crucial para mantener la eficiencia en memoria, mientras que la determinación de realizar el cálculo de reincidencia en el Collator, aunque más costosa computacionalmente, fue necesaria para garantizar el acceso a los datos completos. Se descartó la alternativa de realizar estos cálculos en el Reducer debido a la complejidad que introducía en el manejo del estado y la dificultad para mantener la consistencia de los datos.

Disclaimer: Para esta query la lectura de las fechas, es con comillas simples, si bien no es requerida de esta forma en el tp, teníamos un error que nos llevó a utilizar este separador.

Por ejemplo, un ejemplo para correr esta query es:

```
sh client/src/main/assembly/overlay/query3.sh -Daddresses="127.0.0.1:5701" -Dcity=NYC
-DinPath=./resources -DoutPath=./resources/out -Dn=2 -Dfrom='01/01/2020' -Dto='31/12/2023'
```

D. Query 4: TOP N infracciones con mayor diferencia.

La implementación de la Query 4, diseñada para identificar las N infracciones con mayor diferencia entre montos máximos y mínimos para una agencia específica, adoptó un enfoque de optimización temprana. El flujo MapReduce se estructura con un Mapper que emite pares ***(infracción, monto)*** únicamente para las multas de la agencia especificada, implementando así un filtrado temprano de datos; un Combiner que mantiene eficientemente los valores mínimos y máximos por infracción en cada nodo mediante la clase MinMaxAmount; un Reducer que consolida estos valores extremos a nivel global; y un Collator que calcula las diferencias finales y selecciona las top N infracciones según los criterios establecidos.

Una decisión fundamental fue la implementación del filtrado por agencia en la etapa más temprana posible del proceso, reduciendo significativamente el volumen de datos a procesar en las etapas subsiguientes. Se evaluó la alternativa de almacenar todos los montos para realizar los cálculos al final del proceso, pero fue descartada debido al excesivo consumo de memoria y la innecesaria transferencia de datos a través de la red.

E. Query 5: Infracciones con multas 24/7 en el rango [from, to]

La implementación de la Query 5, diseñada para identificar las multas realizadas en el rango de las fechas [from, to] para una infracción específica. En este caso también se adoptó una optimización temprana dejando el mapa de tickets únicamente con las fechas que se piden. El flujo comienza con un Mapper que recibe los 2 parametros de fechas, así poder verificar la información correcta; el Combiner en este punto no realiza mucha tarea, ya que las llaves “infraction-year-month-day” serán diferentes y no podrán combinarse, este trabajo se hará cargo el Reducer donde verificará que se cumpla que en cada hora del rango especificado exista una multa de alguna infracción. Finalmente, el Collator muestra por orden el nombre de infracciones en un set.

Disclaimer: Para esta query la lectura de las fechas, es con comillas simples, si bien no es requerida de esta forma en el tp, teníamos un error que nos llevó a utilizar este separador. Además solo se puede ser utilizada para el dataset Chicago.

Por ejemplo, un ejemplo para correr esta query es:

```
sh client/src/main/assembly/overlay/query5.sh -Daddresses='127.0.0.1:5701' -Dcity=CHI
-DinPath=./resources -DoutPath=./resources/out -Dfrom='02/12/2013' -Dto='03/12/2013'
```

Análisis de Tiempos de ejecución

Los datos de prueba utilizados para fueron los provistos por la cátedra, se realizaron las corridas con una cantidad menor a las mismas.

- Tickets NYC: 105.583 registros
- Tickets CHI: 100.302 registros
- Infracciones NYC: 97 registros
- Infracciones CHI: 128 registros
- Agencias NYC: 32 registros
- Agencias CHI: 6 registros

Dado que no fue posible ejecutar el sistema con múltiples nodos, se realizó una estimación teórica del comportamiento al escalar en una red local. Se espera que, al aumentar la cantidad de nodos, el tiempo de ejecución disminuye inicialmente de forma significativa debido a la distribución del procesamiento. Sin embargo, factores como el tráfico de red y la necesidad de coordinar los nodos introduce un overhead que reduce los beneficios a medida que se agregan más nodos.

Además, el tipo de query influye en el escalado. Como la Query 1, que procesan datos con una distribución uniforme de claves, aprovechan mejor la paralelización. Por otro lado, la Query 5 o Query3, que dependen de filtros específicos, pueden no escalar eficientemente debido a la concentración de datos relevantes en algunos nodos. Estas estimaciones permiten prever que el sistema es capaz de manejar mayores volúmenes de datos al distribuir la carga, aunque con limitaciones prácticas derivadas de los costos de comunicación y coordinación.

Query1:

Con Combiner	Sin Combiner
111.696 segundos	112.267 segundos.

Query2:

Con Combiner	Sin Combiner
33.094 segundos.	39.203 segundos

Query3:

Con Combiner	Sin Combiner
24.243 segundos	26.002 segundos

Query4:

Con Combiner	Sin Combiner
14.354 segundos	14.946 segundos

Query5:

Con Combiner	Sin Combiner
1.450 segundos	1.600 segundos

Tiempos con lectura paralela

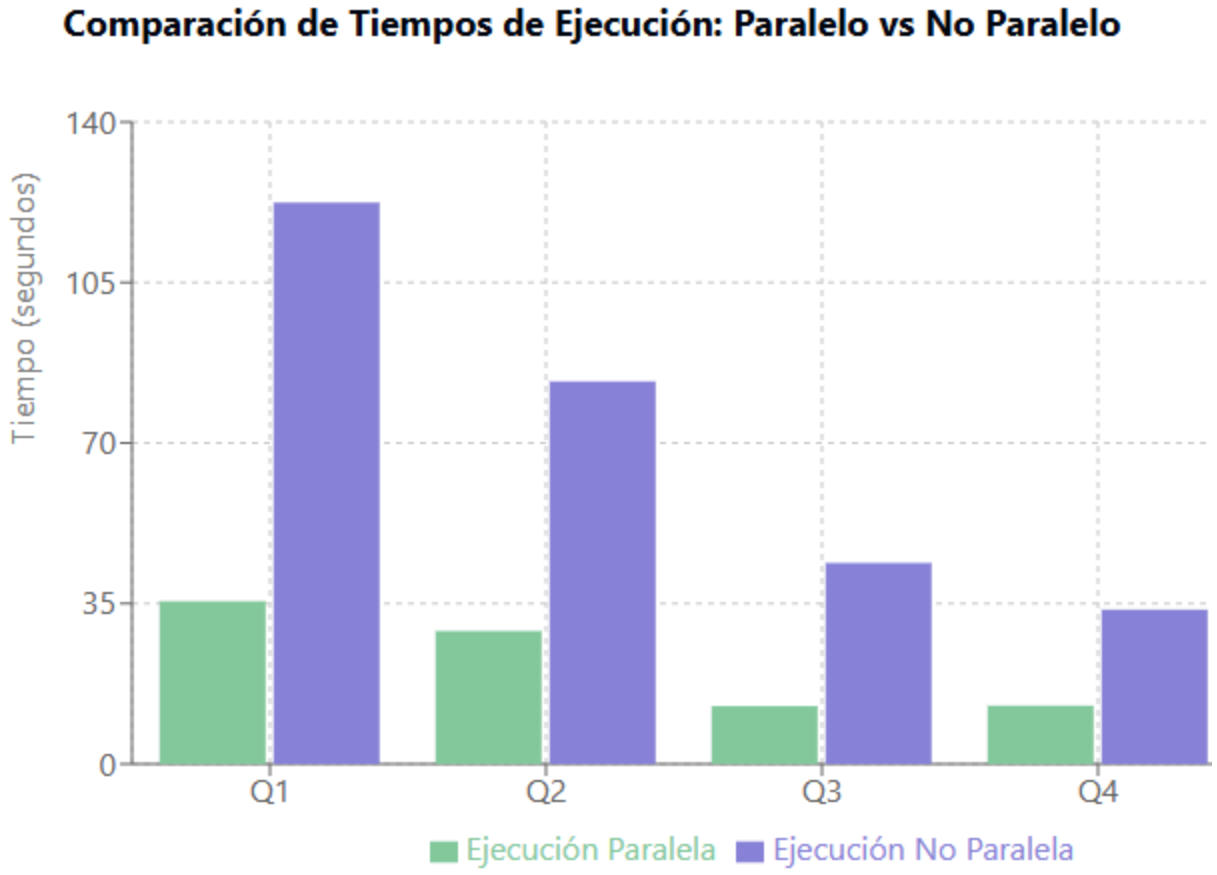
Query	Tiempo Lectura	Tiempo procesamiento	Tiempo Total	%Lectura	%Procesamiento
1	34.236s	1.250s	35.486s	96.5%	3.5%
2	25.613s	3.366s	28.979s	88.4%	11.6%
3	12.413s	0.206s	12.619s	98.4%	1.6%
4	12.468s	0.247s	12.715s	98.1%	1.9%
5	0.367s	0.128s	0.495s	74.1%	25.9%

El análisis de tiempos de ejecución de las cinco queries implementadas revela patrones interesantes en cuanto a rendimiento y distribución de carga. La Query 1, que totaliza multas por infracción y agencia, muestra el tiempo total más elevado (35.486s), con un marcado desbalance hacia el tiempo de lectura (96.5%). La Query 2, enfocada en la recaudación YTD por agencia, presenta una distribución más equilibrada aunque mantiene tiempos elevados (28.979s totales). Las Queries 3 y 4 muestran comportamientos notablemente similares, con tiempos totales cercanos a los 12.6 segundos y una fuerte predominancia del tiempo de lectura (superior al 98%). La Query 5 se destaca significativamente del resto, completando su ejecución en apenas 0.495 segundos y mostrando la distribución más balanceada entre lectura (74.1%) y procesamiento (25.9%).

Tiempos sin lectura paralela

Query	Total NO paralelo	Total paralelo	Mejora (index)
1	122.425s	35.486s	3.45x
2	83.412s	28.979s	2.88x
3	43.841s	12.619s	3.47x
4	33.653s	12.715s	2.65x
5	0.502s	0.495s	1.01x

La implementación de procesamiento paralelo demostró una mejora significativa en los tiempos de ejecución de las queries. Las Queries 1 y 3 exhibieron las mejoras más notables, con una reducción del tiempo total de aproximadamente 3.45x, pasando de 122.425s a 35.486s y de 43.841s a 12.619s respectivamente. La Query 2 mostró una mejora de 2.88x, mientras que la Query 4 logró una aceleración de 2.65x. Notablemente, la Query 5, que ya era extremadamente eficiente en su versión no paralela, mostró una mejora marginal de sólo 1.01x, sugiriendo que para operaciones muy rápidas, el overhead de la paralelización puede neutralizar sus beneficios. La mayor parte de la mejora se observó en los tiempos de lectura, donde la paralelización permitió reducir significativamente los tiempos de carga de datos. Este análisis demuestra que la paralelización es especialmente beneficiosa para queries que manejan grandes volúmenes de datos y requieren operaciones de lectura extensivas, mientras que puede no ser necesaria para operaciones más simples y rápidas.


Comparación de Tiempos de Ejecución: Paralelo vs No Paralelo
Tiempo (segundos)
140
105
70
35
0
Q1
Q2
Q3
Q4
Ejecución Paralela
Ejecución No Paralela

Optimizaciones Adicionales Implementadas

Entre las principales optimizaciones implementadas, se destaca el filtrado temprano de datos como estrategia fundamental para reducir la carga de procesamiento. En la Query 4, se implementó un filtrado por agencia durante la fase inicial de carga, lo que permitió procesar únicamente las multas relevantes para la agencia especificada, reduciendo significativamente el volumen de datos en las etapas posteriores del procesamiento. De manera similar, en la Query 3, el filtrado por rango de fechas en la fase de carga minimizó la cantidad de registros a procesar en las etapas subsiguientes. Por otra parte, la elección de estructuras de datos optimizadas jugó un papel crucial en el rendimiento global del sistema. La utilización de Sets para el manejo de elementos únicos en la Query 3 permitió una gestión eficiente de las patentes sin duplicados, mientras que la implementación de la clase MinMaxAmount en la Query 4 optimizó la transferencia de datos a través de la red al mantener únicamente los valores extremos necesarios. Estas decisiones de diseño no solo mejoraron el rendimiento en términos de tiempo de procesamiento, sino que también resultaron en un uso más eficiente de los recursos de memoria y red en el sistema distribuido.

Potenciales mejoras y/o expansiones

Lectura paralela de datos

Actualmente, los datos se procesan de forma directa antes de ser distribuidos a los nodos para su análisis. Implementar un mecanismo de lectura paralela para los registros de multas podría reducir significativamente los tiempos de preparación inicial. Esto se podría lograr dividiendo los archivos de entrada en particiones manejadas por hilos o procesos independientes en cada nodo.

Se acelera el tiempo de carga inicial de datos, lo que es crucial al trabajar con grandes conjuntos como los de 15 millones de registros para Nueva York, o 5 millones de registros de Chicago. Si bien agregamos una lectura paralela para analizar tiempos de optimización, pero no con un gran volumen de datos.

Optimización del procesamiento de consultas

En particular:

Query 3: Usar estructuras probabilísticas como Bloom Filters para filtrar patentes no relevantes antes de procesarlas completamente.

Query 5: Paralelizar la verificación de rangos horarios por nodo en lugar de centralizar esta validación.

Testing

Una mejora fundamental para garantizar la robustez y confiabilidad del sistema MapReduce sería la integración de una suite de pruebas. Esto permitiría validar el comportamiento de la aplicación en distintos escenarios y detectar errores antes de ejecutar el sistema con los grandes volúmenes de datos reales.