POD - Trabajo Práctico II - Peatones

Integrantes:
Gaspar Budó Berra
Marcos Dedeu
Santiago Hadad
Bruno Squillari
Facundo Zimbimbakis

Diseño de la solución propuesta

Para la ejecución de las cinco queries desarrolladas se decidió guardar la misma información en el clúster. La misma consiste de una denormalizacion de los archivos csv leídos. Esta decisión se tomó teniendo en cuenta que el espacio de almacenamiento en el cluster es distribuido, y por tanto, es conveniente tener información redundante, para evitar el recupero de datos a través de la red, lo cual puede degradar la performance notablemente.

Por otra parte, se generó un debate entre guardar la información de sensors.csv en el cluster o en una colección en memoria del cliente. Al no tener una opción superadora, se proveen ambas funcionalidades para quien ejecute la query, conociendo las dimensiones del archivo y su capacidad de almacenamiento volátil, tome la decisión. Dicha decisión no es trivial, pues teniendo un archivo sensors.csv de gran tamaño, colecciones auxiliares para almacenamiento efímero, una memoria ram relativamente limitada puede llegar a impactar negativamente sobre el rendimiento de los mecanismos de swapping del sistema operativo. Sin embargo, en el caso de un archivo pequeño, se puede agilizar el acceso a la información si se guarda en memoria. También es conveniente analizar la ubicación física de los nodos del cluster para analizar el tiempo de latencia en la red.

Luego, como principio general, todas las queries fueron desarrolladas orientandonos a la eficiencia en la cantidad de información que viaja entre los nodos para computar el map-reduce. A continuación, se detallarán los valores emitidos por cada mapper y la función del reducer relacionado.

Query 1:

El mapper emite para cada entrada, si dicho sensor está activo, la descripción del sensor y la suma de peatones que tiene asociada. El reducer para cada clave suma los valores, y por tanto se obtiene la suma de los peatones para cada sensor.

Query 2:

El mapper emite para cada entrada el año como clave, y, el día y cantidad de peatones. Finalmente el reducer lleva la suma de dos enteros teniendo en cuenta si es un día de fin de semana o no.

Query 3:

El mapper emite, si la cantidad de peatones supera el umbral, para cada entrada la descripción como clave, la cantidad de peatones, y la fecha. Luego, el reducer toma el valor máximo asociado a una clave.

Query 4:

El mapper emite, si el valor es del año especificado, la descripción del sensor como clave, asociado al mes y cantidad de peatones registrados. El reducer suma las cantidades

por mes, y por último, calcula el promedio. A través de un collator se toman los primeros n valores especificados.

Query 5:

Se implementó a través de dos jobs map-reduce. El primero es equivalente al mapper de la query 1. Mientras que el segundo, emite el grupo al que pertenece el sensor como clave, y dicho sensor como valor. Finalmente un collator, agrega todos los valores en una lista. Mediante un collator se generan los pares de sensores pedidos.

Análisis de tiempos

Para acelerar el tiempo de pruebas, decidimos recortar el csv del readings.csv a 10.000 registros. Las pruebas fueron realizadas con 1 nodo por máquina en la red de ITBA-Libre.

Para cada query, realizamos un testeo con 1 nodo, con 2 nodos y con 3 nodos en el cluster. El tiempo de cada query incluye la carga de archivos. Estos tiempos y los gráficos correspondientes pueden verse en el anexo al final del documento.

Debido a los mecanismos de replicación, y partición que provee hazelcast, se esperaba que el tiempo de carga de información aumente a medida que los nodos miembros del cluster crezcan en número. Esto puede comprobarse con los resultados obtenidos, y la notable diferencia que se observa. Por otra parte, se esperaba que el tiempo de procesamiento de los map-reduce disminuya a medida que la cantidad de nodos sea mayor. Sin embargo, no se obtuvieron resultados concluyentes en este apartado. Consideramos que esto puede darse porque la cantidad de información no es tan grande.

Análisis con *Combiner*

Se decidió por implementar un combiner en la query 1. Debido a que se emite un valor por cada medición registrada, y el objetivo es computar la suma total, resulta más efectivo el procesamiento intermedio en el mapper, para colapsar la cantidad de información que viaja a través de la red. Los resultados pueden observarse en la tabla 2. La mejora es notoria.

Potenciales puntos de mejora y/o expansión

Debido a las limitaciones temporales, no se pudieron incluir tests de unidad a la solución propuesta. Sin dudas, incluir un testeo unitario, brindaría un sistema con mayor grado de confianza.

Luego, la implementación actual siempre carga toda la información que se recibe en los archivos. Esto sería útil en el caso de que la información se suba una única vez, sin embargo, este no es el caso. Nos dimos cuenta que sería conveniente que cada query cargue únicamente la información necesaria. En cada consulta se carga información inutil lo cual impacta en el tiempo que se demora en leer los datos.

A su vez, en varias cargas, enviamos toda la información asociada al sensor y sus lecturas, haciendo que esa carga/descarga sea mucho más lenta que, capaz, manejarse con los IDs y luego utilizar collators para reemplazar los IDs por su correspondiente información.

Por último, el proceso de lectura de los archivos se realiza de a una línea. En otras palabras, se lee una línea y esta se envía a los clusters de hazelcast. El proceso de lectura se podría optimizar si la lectura y subida a los clusters se realizaran por chunks.

Anexo

Tabla de resultados

Query	Nodos	Lectura (s)	Map-reduce (s)	Total (s)
Query 1	1	102.777	0.454	103.231
	2	239.772	0.536	240.308
	3	275.757	1.081	277.567
Query 2	1	107.485	0.487	107.972
	2	249.034	0.850	249.884
	3	223.770	1.800	224.157
Query 3	1	116.495	0.508	117.003
	2	212.141	0.772	212.913
	3	240.777	1.038	241.815
Query 4	1	100.368	0.716	101.084
	2	229.167	1.050	230.217
	3	218.461	1.433	219.894
Query 5	1	159.863	0.734	160.597
	2	335.179	5.338	340.517
	3	317.517	3.982	321.499

Tabla 1: Tiempos de las 5 queries

	Nodos	Lectura	MapReduce	Total
Con Combiner	1	102.777	0.454	103.231
	2	228.774	0.542	229.316
	3	194.919	0.674	195.593
Sin Combiner	1	102.777	0.454	103.231
	2	239.772	0.536	240.308
	3	275.757	1.081	277.567

Tabla 2: Comparación de ejecuciones con y sin combiner

Query 1

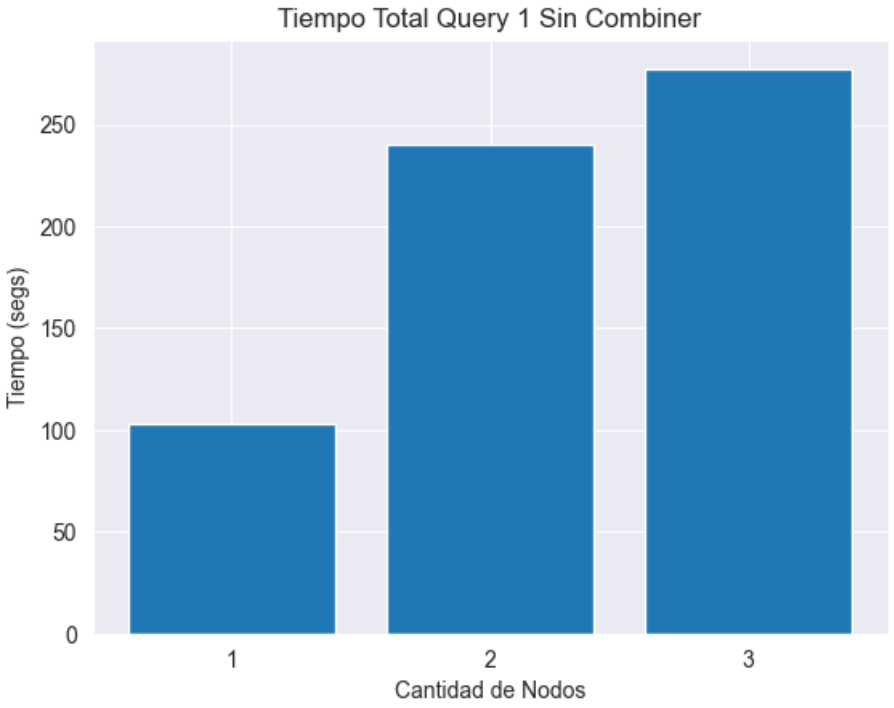

Gráfico 1: Tiempo total de la query 1 sin combiner

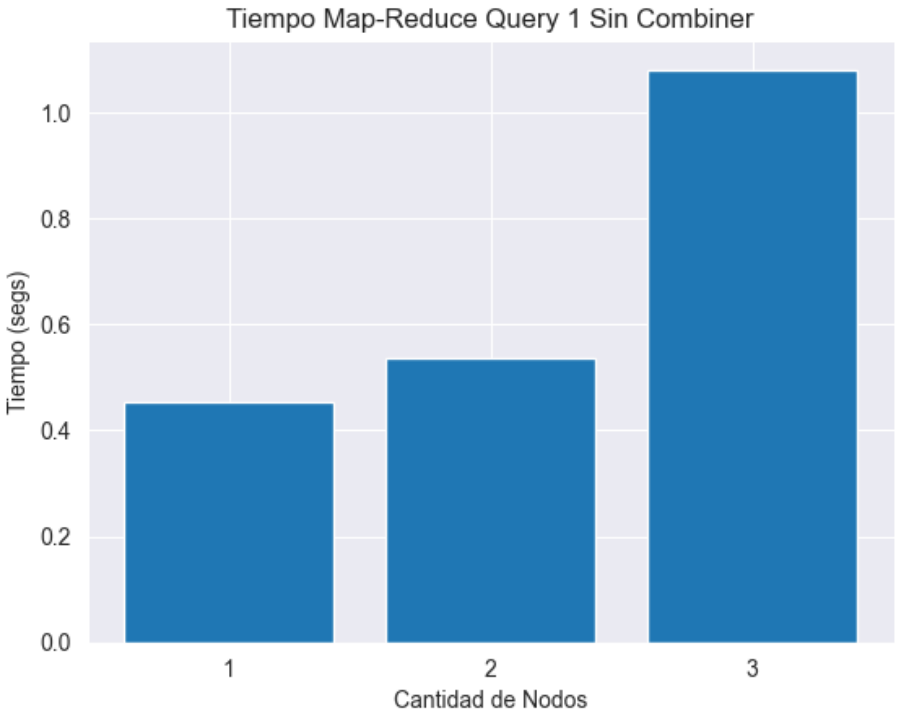

Gráfico 2: Tiempo del Map-Reduce de la query 1 sin combiner

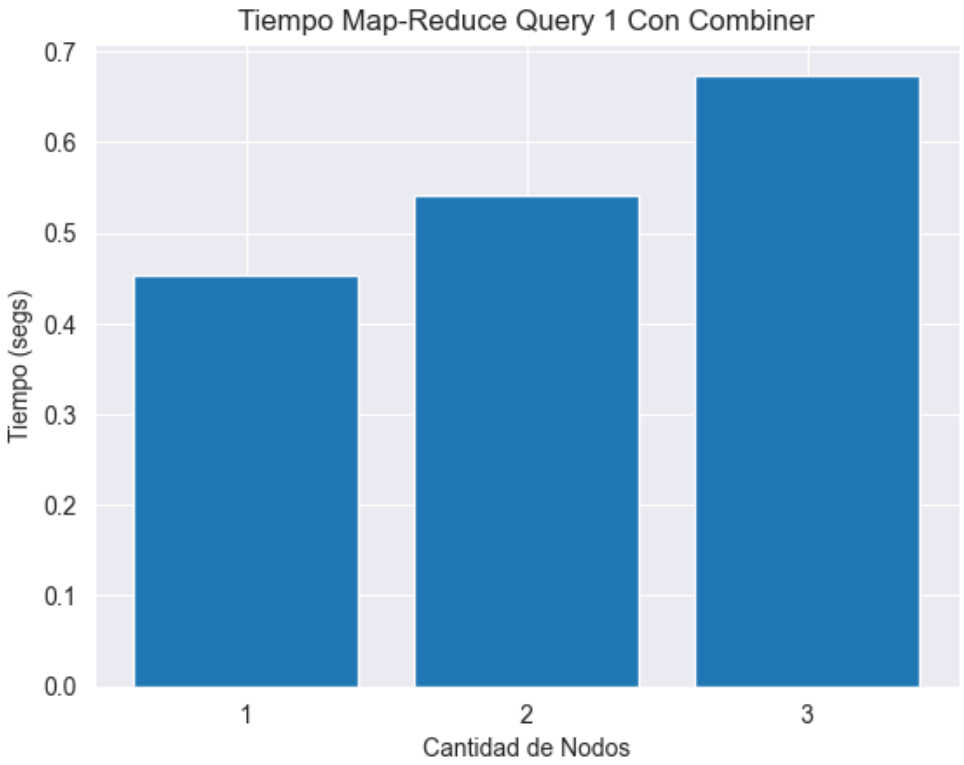

Gráfico 3: Tiempo del Map-reduce de la query 1 con combiner

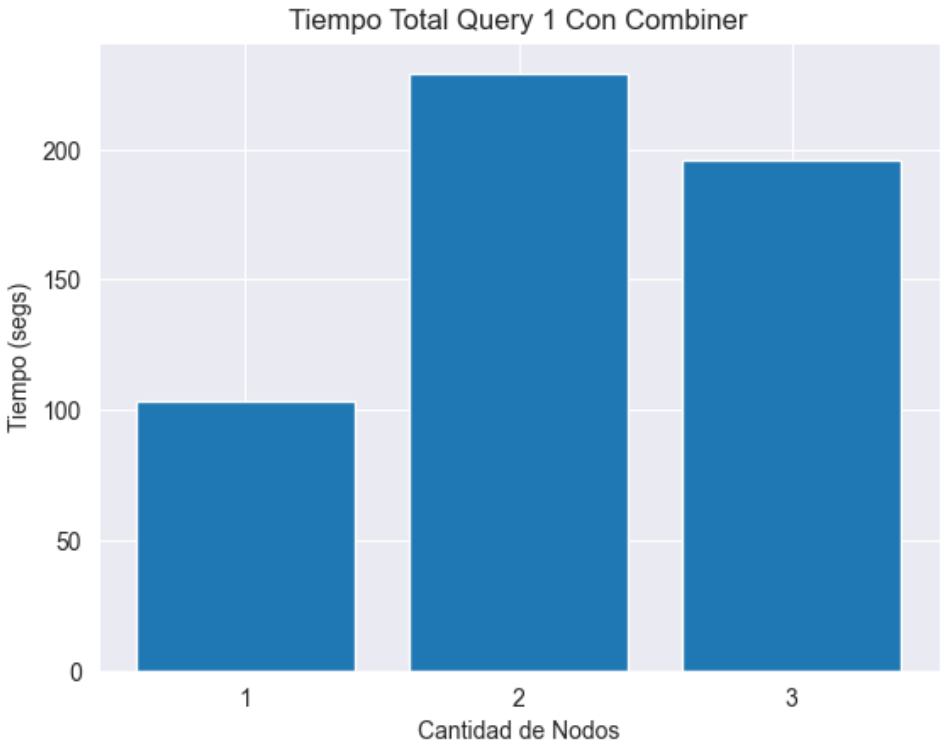

Gráfico 4: Tiempo total de la query 1 con combiner

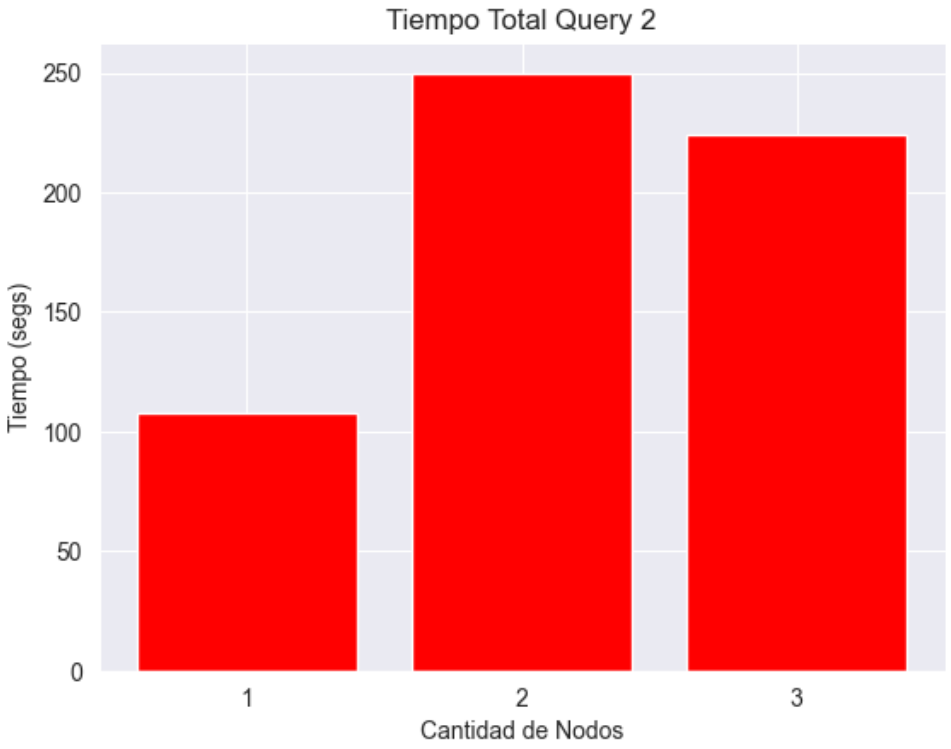

Gráfico 5: Tiempo total de la query 2

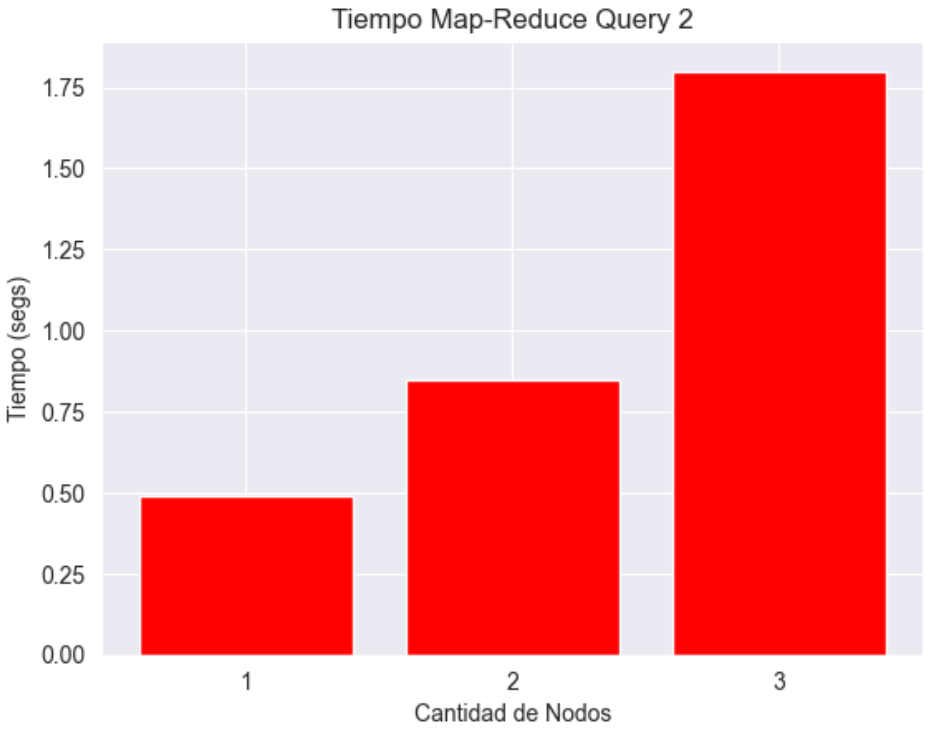

Gráfico 6: Tiempo del Map-reduce de la query 2

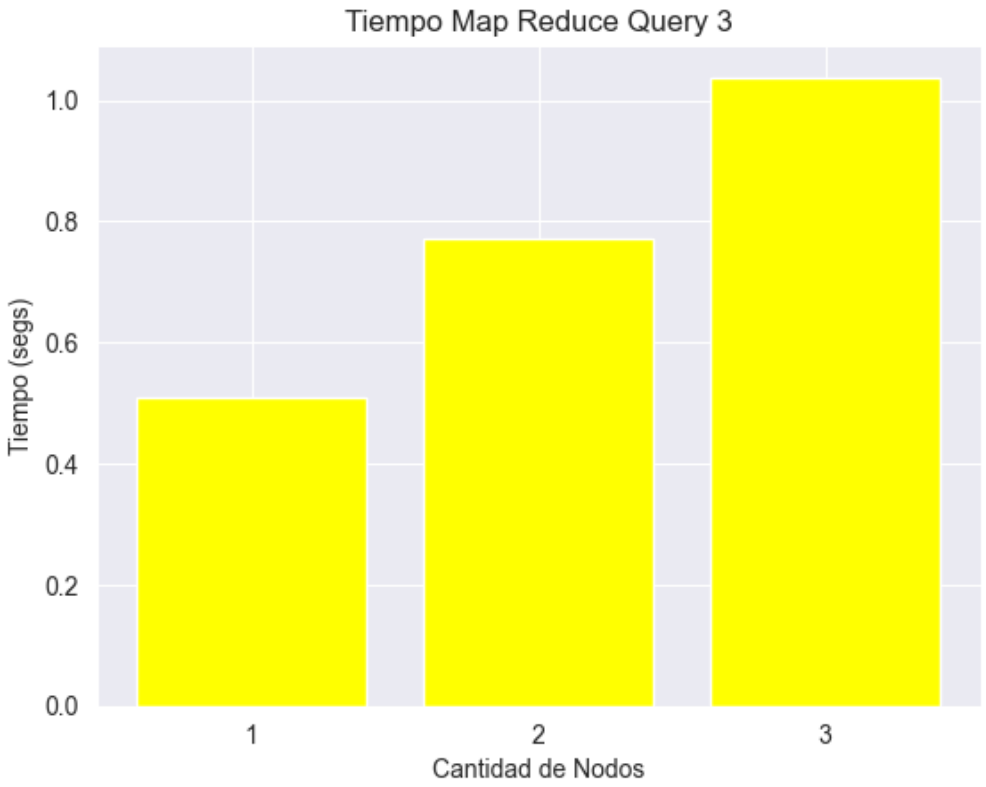

Gráfico 7: Tiempo total de la query 3

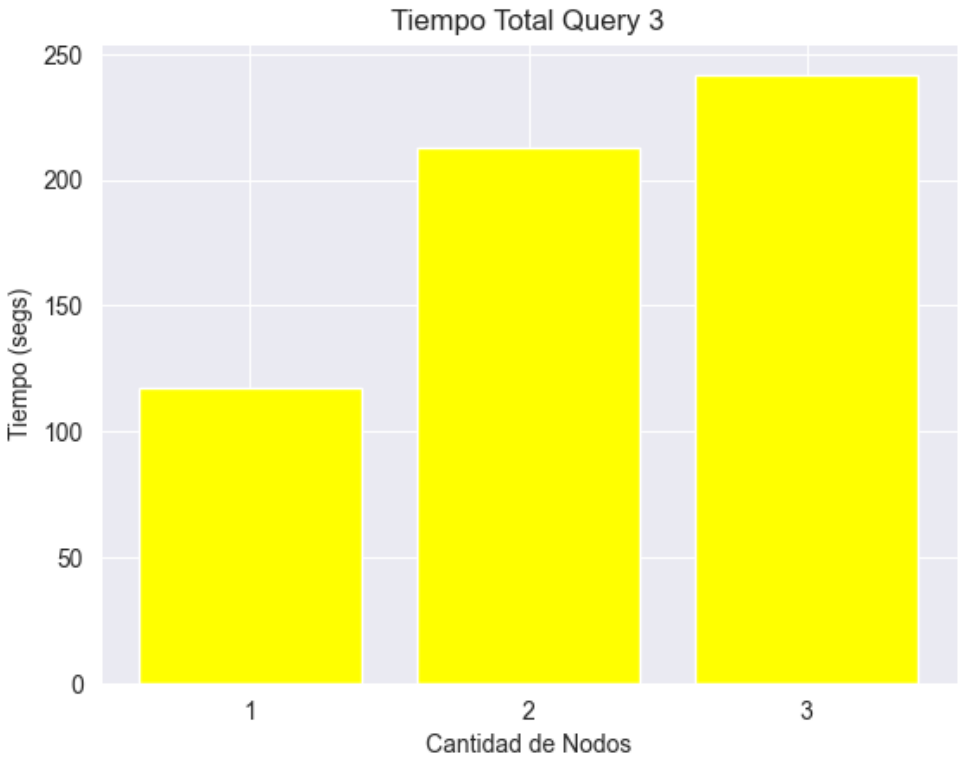

Gráfico 8: Tiempo del Map-reduce de la query 3

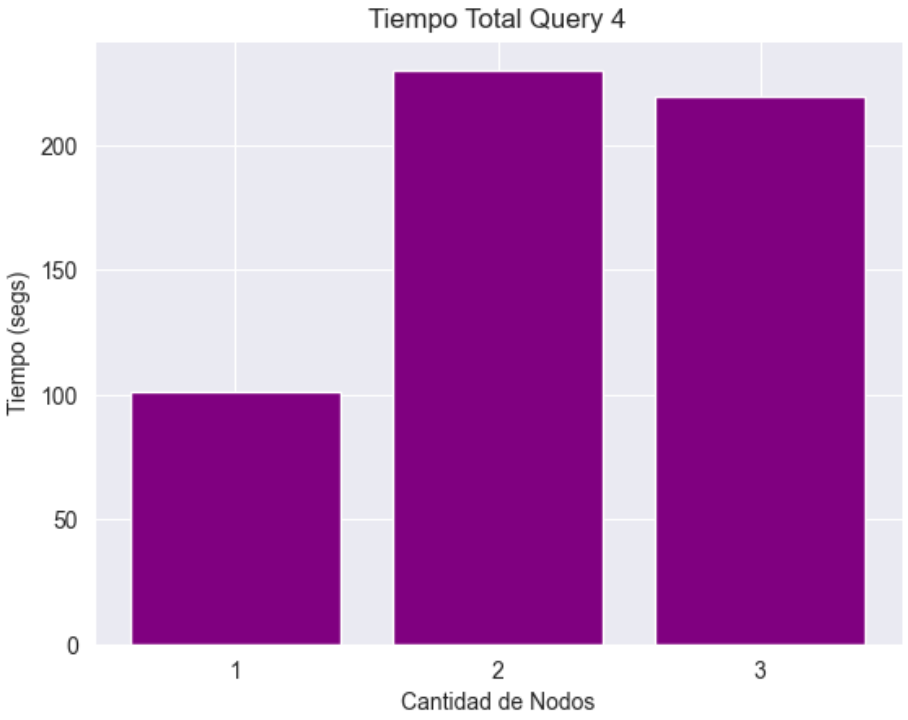

Gráfico 9: Tiempo total de la query 4

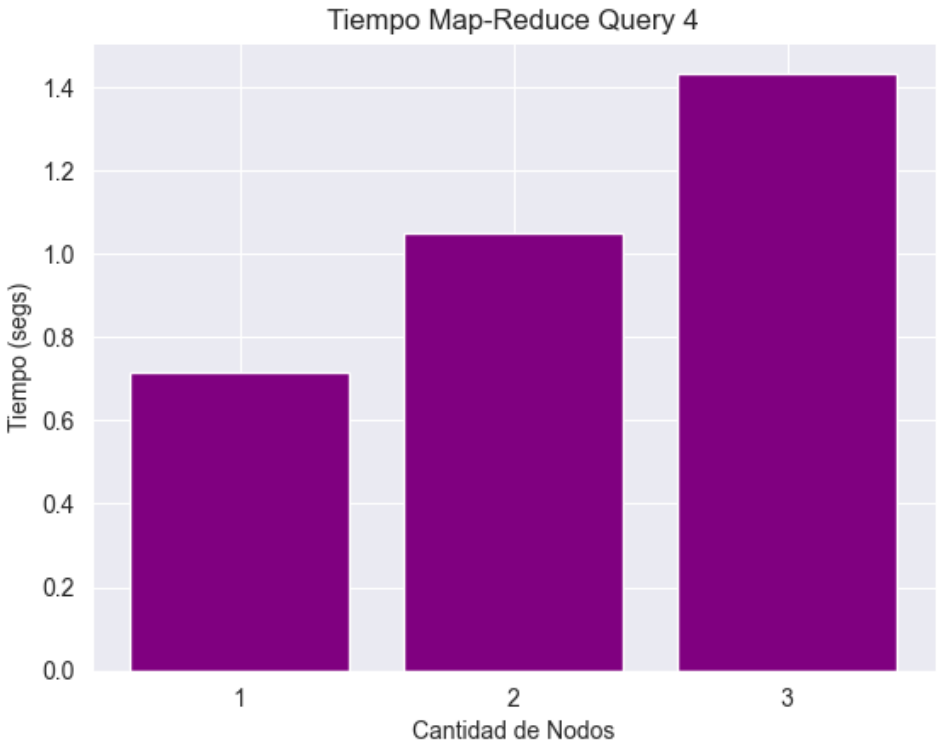

Gráfico 10: Tiempo del Map-reduce de la query 4

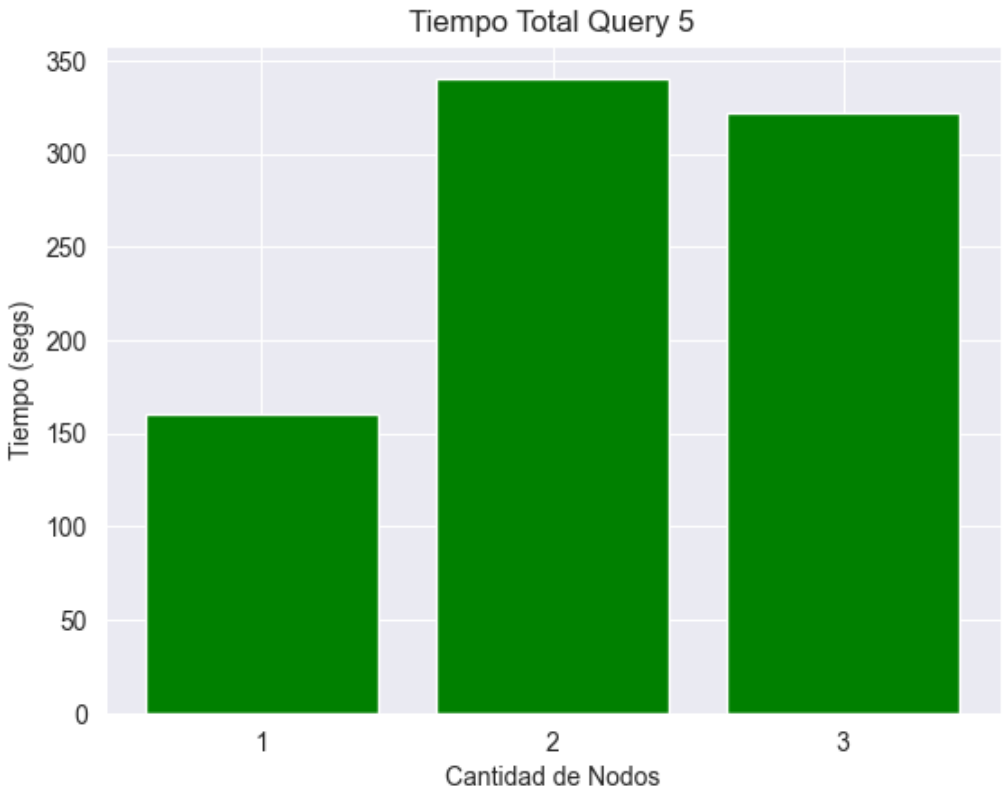

Gráfico 11: Tiempo total de la query 5

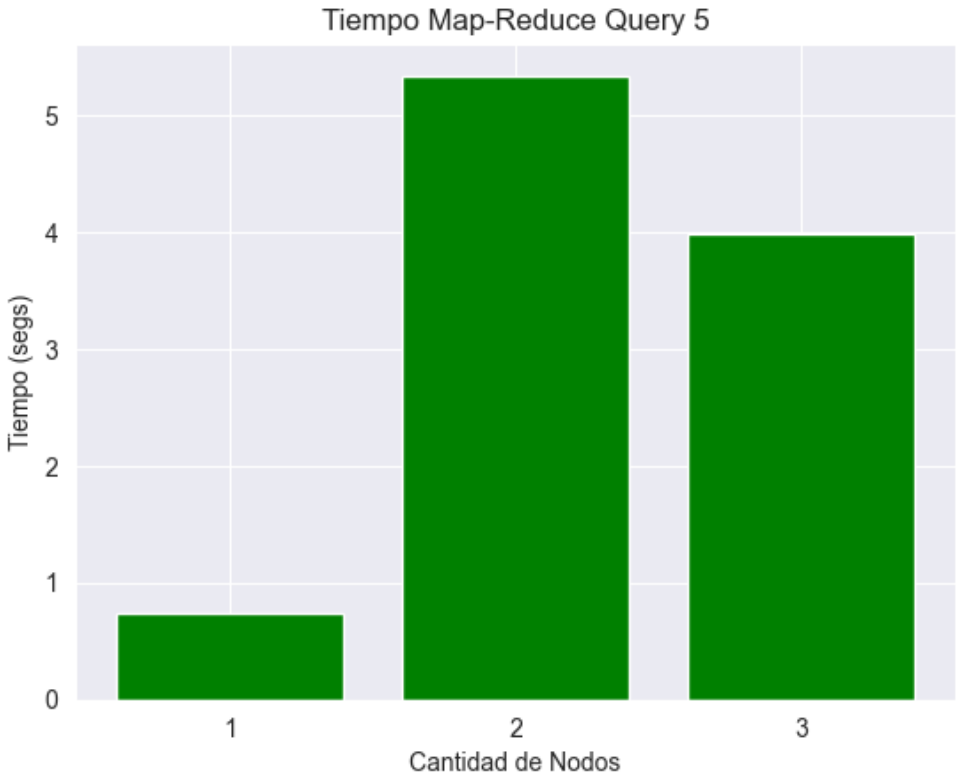

Gráfico 12: Tiempo del Map-reduce de la query 5