ITBA

Distributed Objects Programming

Map-Reduce on Hazelcast

(56.266) Joaquín Filipic *(56.343) Martín Biagini* *(53.396) Agustín Golmar*

November 1, 2018 | **ITBA**

❖ Map-Reduce on Hazelcast

Diseño de Componentes

En todas las consultas, se utilizan los componentes *'mapper'*, *'reducer'*, y *'collator'* como mínimo. En este último, se encapsula la lógica que haga falta luego del reducer, para poder devolver la estructura deseada. Evitamos rotundamente manipular estructuras del lado del cliente. Esto último, hace que tengamos que realizar mucha subida y bajada de estructuras al cluster, degradando la performance. Sólo se utilizó el componente *'combiner'* en aquellas situaciones en las que se esperaba un número muy alto de entradas clave-valor repetidas en el mapa *'output'* del componente mapper.

Decisiones Destacadas Adoptadas

Lambdas en flujo map-reduce: Se decidió no utilizar funciones lambda para la implementación de los métodos *'mapper'*, *'combiner'*, *'reducer'*, y *'submit'*, ya que muchas de las soluciones que adoptamos, contenían una lógica que consideramos no apta para ser implementadas con funciones lambdas por cuestiones de legibilidad y complejidad. Esto no ocurre en todos los casos, pero para mantener consistencia y unificar el código.

Reutilización de consultas: Se optó por utilizar consultas para facilitar la resolución de otras. Esto, que pareciera algo trivial, deja de serlo cuando la información devuelta por una consulta que se quiere utilizar, difiere en alguna medida con la información óptima que se busca para solucionar eficientemente el problema en cuestión. Además, se debe tener en cuenta que todas las consultas devuelven una respuesta ordenada. Por lo tanto, se parte de una base de pérdida de eficiencia. Esta decisión no fue tomada a la ligera, y finalmente se le dió más importancia a tener un código más limpio y fácil de seguir, que a ganar una leve mejoría de eficiencia.

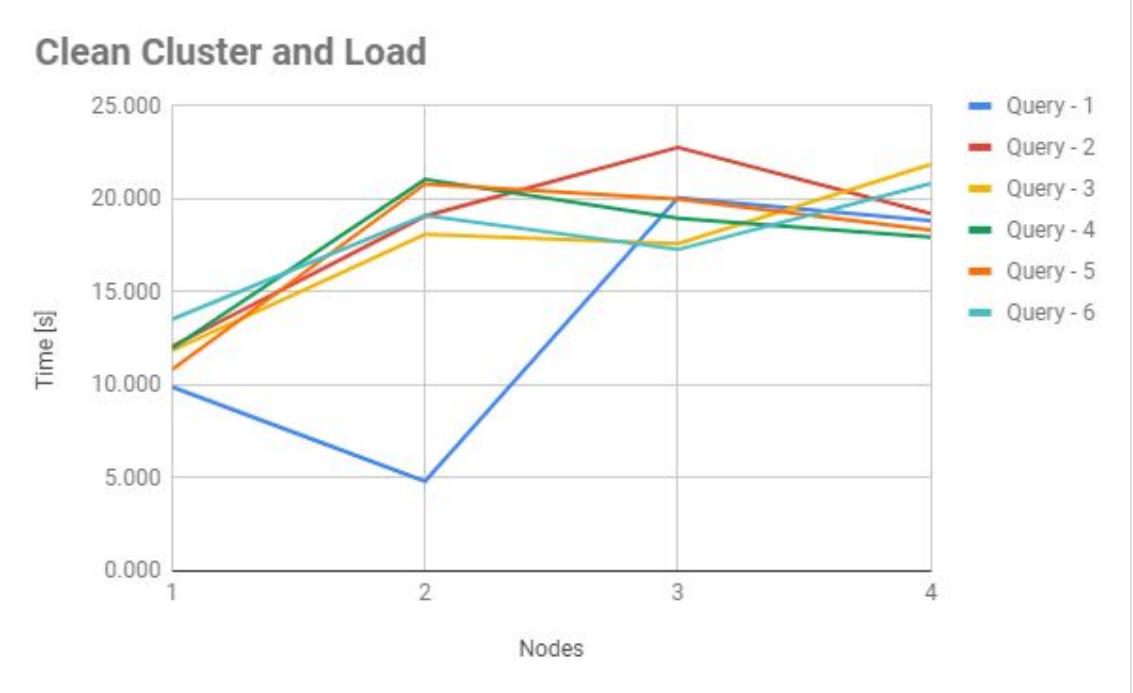

Figura 1: La limpieza del cluster implica vaciar las estructuras remotas por completo (**clear**). La carga implica subir los aeropuertos y movimientos a las listas distribuidas.

Carga completa de datos en memoria: utilizado para leer los aeropuertos y movimientos en formato *CSV*. Esto, con un volúmen de datos muy grande, se debería realizar de a poco, leyendo y subiendo la información de a bloques. Decidimos no realizar esto último, ya que el propósito del trabajo no lo requería (y la cátedra indicó que no era necesario).

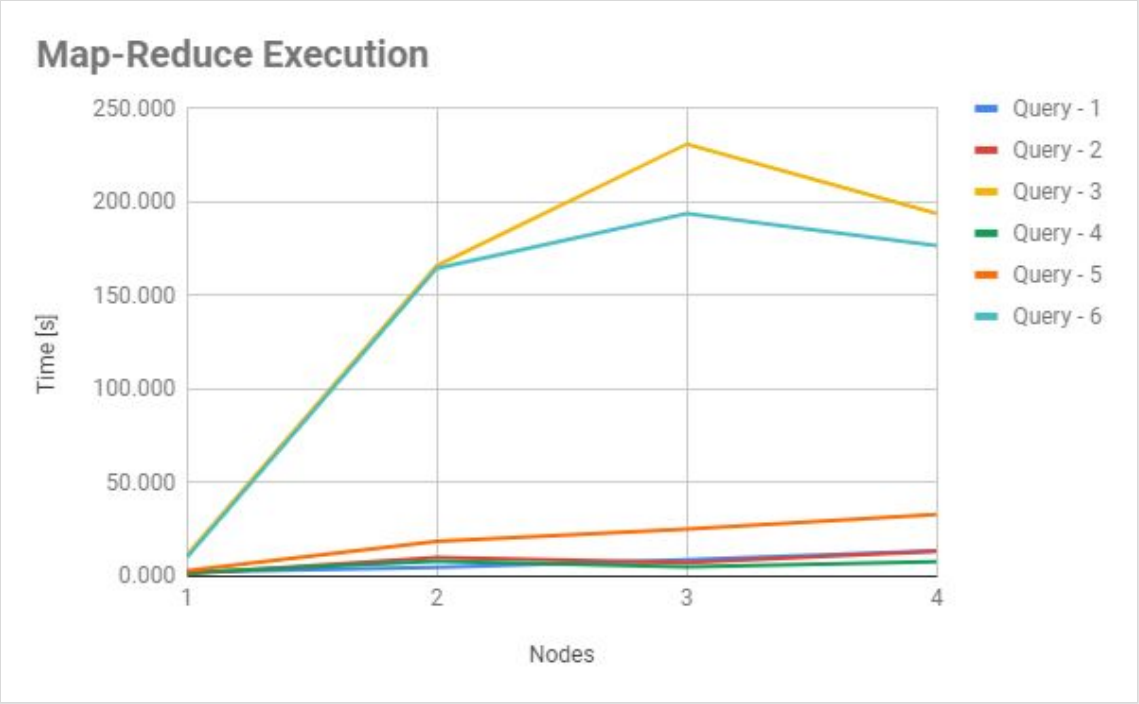

Figura 2: La ejecución completa de cada query, según la cantidad de nodos solicitada (máximo 4 nodos). Las queries menos performantes son la N° 3 y la N° 6 (cabe destacar que la 6 depende de la 3).

Estadísticas desactivadas: el sistema *Hazelcast* permite computar estadísticas de acceso en las estructuras de datos utilizadas. Se desactivaron por completo para incrementar la performance.

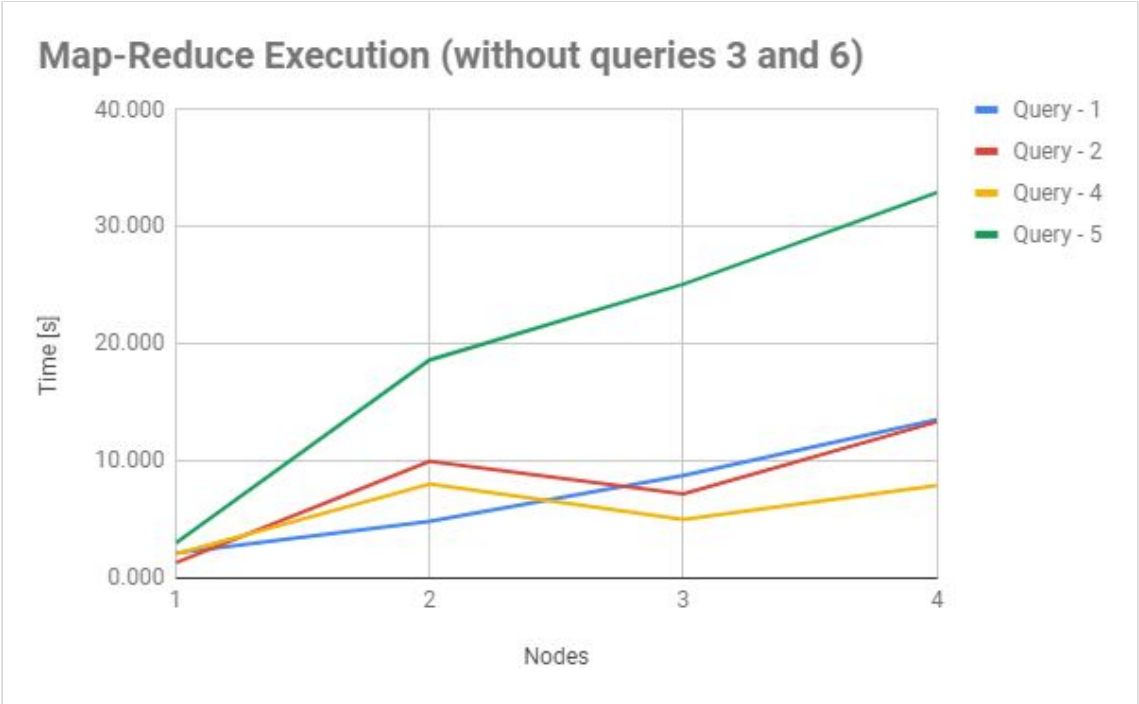

Figura 3: Se descartan la 3 y la 6, para apreciar con mayor exactitud las variaciones de las demás consultas.

Comentarios Finales

En los gráficos se puede observar una caída de performance al pasar, por ejemplo de un cluster de 3 nodos, a uno de 4. Creemos que esto está relacionado con varias cuestiones. En primer lugar, como ya fue mencionado al comienzo de este documento, se realizan muchas subidas y bajadas de datos, que, teniendo en cuenta que muchas operaciones manejan volúmenes de datos muy chicos (en operaciones intermedias, aparecen listas o mapas del orden de decenas de elementos), terminan degradando la performance. Además, el sistema fue configurado con réplicas. Esto ocasiona que se deba sincronizar cada vez que se sube información al cluster (esto empeora al incrementar la cantidad de nodos).

Además, la documentación de Hazelcast indica que las estructuras de tipo lista, y, a diferencia de las estructuras mapa, no son particionables, es decir, que deben residir por completo dentro de un mismo nodo durante su procesamiento, lo cual implica que no se puede aprovechar la capacidad de escalamiento horizontal del cluster durante las acciones que se llevan a cabo sobre estos datos. Esto degrada significativamente la performance.

Lo ideal es utilizar mapas, siempre que sea posible, y en todas las consultas.