# Programación de Objetos Distribuidos

## Informe del Trabajo Práctico Especial Nº2


Integrantes del Grupo nº8:

- Emilio Basualdo Cibils 58172
- Pedro Remigio Pingarilho 58451
- Miguel Di Luca 58460
- Fermín Gómez 58111

# Diseño e implementación de los MapReduces

Aunque todas las queries realizadas tenían sus particularidades, muchos componentes del mapreduce se pudieron reutilizar.El Combiner, por ejemplo usado en el query1 es el mismo usado después en todas las otras queries con excepción de la quinta. Eso se da porque con la excepción de la misma, todas las otras no necesitaban ningún tipo de filtro más específico para realizar la combinación. Eso mismo también está presente en algunos reduces, que dado que son bastante simples(la simple suma de las repeticiones) se pudieron reutilizar. Para todas las queries filtramos los aeropuertos y previo a la carga a HazelCast

Aclaración: no sabíamos si se supone que en teoría los datos estarían ya previamente cargados y distribuidos en los nodos(cosa que no existiría la opción de un pre-filtrado) o si serían cargados manualmente *ad hoc*, por ende la Query 1 hace referencia a la primera opción y la Query 6 a la segunda ( la que nos permite filtrar información previo a ser enviada por la red) como forma de implementar las diferentes situaciones.

Query 1: Usamos un IMap para guardar pares OACI => Denominación para simplificar la generación de la respuesta y para filtrar desde la función mapper. Luego filtramos/emitimos en el Mapper según condición de la consulta usando el IMap mencionado arriba(habiendo implementado HazelcastInstanceAware). El combiner y reducer son similares al de un word count. El ordenado y creación del resultado final se hace en el collator donde utilizamos el IMap para buscar las denominaciones. Contras encontradas: utilizamos una IList y como esta no se particiona, todo el MapReduce se corre en el mismo nodo. Tal vez una estructura de tipo MultiMap hubiera sido mejor.

Query 6: A diferencia de la Query 1, filtramos y reducimos el tamaño de los movimientos de interés previo a la función MapReduce. Usamos un mapa local para guardar pares OACI => Provincia para simplificar la generación de la respuesta. Map, Combine y Reduce son muy simples. El ordenado y creación del resultado final se hace en el collator donde utilizamos el IMap para buscar las denominaciones y además filtramos según -Dmin. Usamos una IList por ende la información no se encuentra particionada. No encontramos otra manera de filtrar según -Dmin previo al reduce.

Query 2: filtramos los cabotajes desde el mapper. Así como la implementación del primero, el combiner y reducer son bastante simples, y así se reutilizan. Sin embargo, en este trabajo necesito procesar la data final, lo que hizo con que se use un collator. Ahí, se calcula el porcentaje dado que es necesario saber el número de vuelos de cabotaje total. Además es necesario retirar a los vuelos de cabotaje que no poseen una aerolínea especificada. Como se necesitaba el número total de vuelos, se filtra a la información apenas después de la reducción.

Query 3: realizamos dos mapreduces. Primeramente se reduce de la misma manera que la primer query, para obtener a los OACIs y el número de movimientos en cada uno. Una vez que obtenemos a este mapa realizamos el segundo. El mapeo se responsabiliza

en hacer el redondeo a la casa de los millares y la inversión de las claves y valores obtenidos en el primer mapeo. Al momento del reduce, juntamos a todos los OACIs, permutando a las combinaciones de a pares en orden alfabético sin repeticiones,que es más rápido que hacerlo una vez que está concluido al mapreduce. Se usa un collator para ordenar al número de movimientos.

Query 4: se envía como parámetro el OACI del aeropuerto de origen al mapper, éste filtra por el OACI y por los movimientos de despegue. Luego de pasar por el mapper se utiliza el mismo combiner y reducer que en la query número dos. Al finalizar, se utiliza un collator el cual recibe como parámetro la cantidad de aeropuertos destino a mostrar, el collator ordena el mapa de manera descendente y luego retorna un submapa con la cantidad deseada.

Query 5: usamos el mapeo de una manera distinta a las queries anteriores para llegar al resultado esperado. Como necesitamos saber tanto al número total de vuelos como cuantos de los mismos son privados, se usa como estrategia demarcar a los vuelos no privados con un 1 y los no privados con un 0. De esta manera, en el combiner y reducer pudimos guardar de cada oaci el número de vuelos privados y el número de vuelos totales, para así en el collator poder sacar el porcentaje correspondiente. Para filtrar a los aeropuertos a los que están en el csv de aeropuertos, optamos por hacerlo en el collator sin la necesidad de un mapreduce aparte solo para eso, dado que puede ser trivial el filtro desde java 8(reescribir esto bien).

## Análisis de los tiempos

| --------- --------- | Un nodo | Dos nodos | Tres nodos | Cuatro nodos |
|---|---|---|---|---|
| Query 1 | **31.107s** | 32.152s | 34.655s | 59.083s |
| Query 2 | 4.456s | **3.910s** | 3.943s | 5.079s |
| Query 3 | **4.885s** | 5.178s | 5.627s | 8.003s |
| Query 4 | **5.111s** | 5.242s | 5.224s | 5.359s |
| Query 5 | **6.621s** | 7.624s | 7.974s | 12.445s |
| Query 6 | 5.636s | **4.756s** | 5.180s | 5.777s |

El mejor tiempo para cada query se encuentra resaltado en la tabla. Como podemos observar, en la mayoría al incrementar la cantidad de nodos se incrementa el tiempo de

ejecución, solo son dos los casos en que obtenemos un menor tiempo con dos nodos, esto sucede en la query 2 y 6.

Cabe destacar que las ejecuciones se realizaron en distintos usuarios de pampero y el resultado podría verse alterado según como pampero administra los recursos para cada usuario.