Programación de Objetos Distribuidos (72.42)

2do Cuatrimestre 2021

Trabajo Práctico Especial 2

Árboles

Grupo 6 - Integrantes:

- Santiago Burgos - legajo 55193
- Ximena Zuberbuhler - legajo 57287
- Cristóbal Matías Rojas - legajo 58564
- Enrique Tawara - legajo 58717

Repositorio

https://github.com/etawara/ITBA_POD_2C2021_G06_TPE2

Decisiones de diseño de los componentes MapReduce

Para la primera query, la cual se pide la cantidad total de árboles por barrio, se decidió implementar Trees*CounterMapper*, que por cada árbol cuyo barrio esté presente en el archivo de barrios, emite como clave el barrio y un 1 como valor. Para el Reducer se implementó el *SumReducer*, que retorna la suma de los valores dado una clave. En este caso, se mantuvo la clase genérica, para poder ser reutilizada. Para finalizar la query, se creó *DesVAscKCollator*, el cual ordena los resultados del mapReduce.

En el caso de la segunda query, la cual se pide para cada barrio la especie con mayor cantidad de árboles por habitante, se decidió realizar dos jobs de mapReduce. El primero, se implementó el mapper *countOverValueMapper*, que por cada árbol cuyo barrio esté presente en el archivo de barrios, emite como clave una clave compuesta *PairCompoundKeyValue* y como valor 1/(cantidad de habitantes). La clase *PairCompoundKeyValue* posee dos String como clave y un valor de tipo Double. En este caso, las claves son el barrio y la especie y el valor es la cantidad de habitantes. Para el Reducer, se utiliza el mismo reducer que la query 1 obteniendo una lista con PairCompoundKeyValue como clave y el índice de árboles por habitante como valor.

En el segundo job, se creó el mapper *Key1PairMapper*, el cual emite la Key1 del PairCompoundKeyValue como clave y un PairCompoundKeyValue como valor, en el cual se guarda el barrio, especie e índice de arboles. Para el reducer, se usó MaxValueReducer el cual conserva el máximo valor del índice, en caso de tener el mismo valor se queda con el árbol con menor nombre alfabeticamente. Finalmente se usa KascCollator para ordenar por barrio ascendente.

Para la tercer query, la cual pide cantidad de especies únicas por barrio se decidio armar el NeighbourhoodSpeciesCounterMapper, el cual emite un par clave valor (barrio, nombre del árbol) por cada entrada del mapper, asegurándonos que el barrio del árbol pertenezca a un array de barrios válidos inicializado en la creación del Mapper (Se utiliza este método en vez de un KeyPredicate dado que la clave en este caso es el nombre de la colección y no tiene info sobre barrio, además se utiliza esta forma en vez de filtrar por código antes del MapReduce para tratar de mantener toda la lógica en el MapReduce). Luego para la reducción implementamos un UniqueReducer el cual guarda para un barrio determinado un set con todos los nombres de árboles y al finalizar la reducción se retorna el tamaño de ese set, que es efectivamente la cantidad de especies de árboles únicos en ese barrio. Finalmente se implementó el TopNCollator, el cual ordena los elementos por el valor, en este caso, la cantidad de especies únicas de cada barrio. En el caso de que haya dos barrios con el mismo valor, se ordena por la clave. Una vez ordenado se retornan los n primeros elementos.

En el caso de la cuarta query reutilizamos el Mapper y Reducer de la tercera, con la unica modificacion siendo al reducer agregando un parámetro floorValue que se aplica al finalizar el reducer y redondea el número de especies únicas (Por ejemplo si floorValue es 100 redondea 325 a 300). Finalmente como se solicitan pares de barrios implementamos el PairedValuesCollator el cual crea una lista ordenada de PairedValues. PairedValues es un modelo que implementamos que contiene el valor que ambos miembros tienen en común (commonValue) y ambos miembros (miembro 1 y miembro 2) asegurándonos que miembro1 es siempre el menor de los dos, de esta forma evitamos que existan inversos y tiene la lógica de comparación para poder ordenarlos correctamente según enunciado.

Finalmente la quinta query utiliza el StreetSpecificTreeMapper que recibe en construcción el nombre de un barrio y el nombre de un árbol y emite un par (nombre de

calle, 1L) únicamente si esa entrada cumple con que el barrio y el nombre del árbol sean iguales a los recibidos en construcción. Luego se llama al SumCombiner como paso intermedio para sumar parcialmente estos pares clave valor emitidos y el reducer es el SumInTensReducer que similar al combiner suma las entradas pero con la diferencia de que al finalizar retorna en decenas (es decir si el valor es 58 devuelve 50). Para el armado de pares se utilizó el Collator del cuarto query.

Análisis de tiempos de resolución

Nodos	Tiempo BUE	Tiempo VAN
1	0.29	0.43
2	1.31	0.27
3	0.40	0.30
4	0.85	0.36
5	0.92	0.35

Duración en segundos de operaciones MapReduce para Query 1

Nodos	Tiempo BUE	Tiempo VAN
1	15.71	3.02
2	19.94	5.78
3	19.94	6.60
4	21.07	7.33
5	21.46	6.62

Duración en segundos de operaciones MapReduce para Query 2

Nodos	Tiempo BUE	Tiempo VAN
1	0.91	0.29
2	0.73	0.22
3	0.75	0.42
4	0.97	0.41
5	1.21	0.44

Duración en segundos de operaciones MapReduce para Query 3

Nodos	Tiempo BUE	Tiempo VAN
1	0.50	0.17
2	0.70	0.40
3	0.92	0.62
4	0.68	0.36
5	1.03	0.51

Duración en segundos de operaciones MapReduce para Query 4

No pudimos probar casos con servidores corriendo en múltiples máquinas, pero para el caso de servidores locales observamos que aumentar la cantidad de nodos genera un aumento del tiempo de ejecución de la operación MapReduce para mayoría de las queries. Consideramos esto sucede por el relativamente pequeño tamaño de datos y el hecho de que todos los nodos compiten por la misma cantidad de recursos. En el caso de 1 solo nodo, ese nodo tiene acceso a todos los datos y todos los recursos del dispositivo y no pierde tiempo enviando resultados a otros nodos, por lo que logra terminar más rápido. Cuando aumentamos la cantidad de nodos introducimos la latencia entre las comunicaciones de nodo a nodo que afectan el performance. Si estuviéramos en un ambiente propiamente distribuido donde cada nodo puede usar el total de recursos de su dispositivo entonces esperaríamos ver una mejora de performance a pesar de esta latencia de comunicación entre nodos, pero en el caso local como tener 2 nodos implica que cada nodo trabaja con la mitad de recursos y además se agrega latencia de comunicación entre nodos, la performance decrece. Para Vancouver las diferencias de duración son chicas pero se ve más notoriamente en Buenos Aires donde el volumen de datos es mayor y genera operaciones más complejas.
Se omitieron las tablas de la Query 5 por tener muchas ramificaciones en sus parámetros y no era posible realizar una comparación similar entre Buenos Aires y Vancouver.

Potenciales puntos de mejora y/o expansión

Una potencial mejora sería tratar de hacer uso de de los KeyPredicates con algún IMap con claves formadas por barrio + algún valor que evite pisar entradas anteriores, de esta forma conservamos cada Tree de la lista de árboles pero además lo tenemos inicialmente mapeado a una clave que agrega valor y no al nombre de la colección, y de esta forma podemos pre-filtrar en base a esas claves.