ARBOLADO PÚBLICO

Trabajo Practico especial 2

72.42 Programación de Objetos Distribuidos
Grupo 9

Alumnos

Lucas Sanz Gorostiaga	56312
Tomas Maximiliano Dallas	56436
Agustina Osimani	57526
Micaela Banfi	57293

Instituto Tecnológico de Buenos Aires
ITBA

29-10-2020

1 Introducción

En el siguiente informe se presentará la implementación de una aplicación de consola que utilice el modelo de programación MapReduce junto con el framework Hazelcast para el procesamiento de datos de arbolado público, basado en datos reales. Se hará una breve descripción de cómo está organizado el sistema, de cómo funciona y se darán explicaciones sobre decisiones de diseño tomadas.

2 Organización

El proyecto está basado en el arquetipo brindado por la cátedra, y consta de las siguientes tres partes

- API: presenta las interfaces para la comunicación entre los clientes y los servicios y todo lo necesario para las queries.
- Client: aquí se encuentran las queries, los validadores, los parsers
- Server: simplemente levanta una instancia de Hazelcast

3 Decisiones de diseño e implementación de los MapReduce

3.1 Mappers

Utilizamos tres mappers:

- Counter: para las queries 1 y 2. Cuenta la cantidad de entradas, dejando al objeto pasado como parámetro como clave y un 1 como valor.
- DiameterPerSpecies: para la query 3. Devuelve un par <especie, diámetro>.
- NeighbourhoodCount: para las queries 4 y 5. Ya que ambas contabilizaban los barrios pero una filtraba por especie decidimos hacer una versión más específica del Counter que puede recibir, o no, una especie en su constructor.

3.2 Reducers

- Counter: para las queries 1 y 2. Suma las entradas e indica cuántas hay.
- Average: para la query 3. Calcula el promedio de los diámetros.
- NeighbourhoodCount: para las queries 4 y 5. Ya que ambas contabilizaban los barrios pero una filtraba por especie decidimos hacer una versión más específica del Counter que puede recibir, o no, una especie en su constructor.

3.3 Combiners y collators

- CounterCombiner: para todas las queries excepto la 3. Funciona como un paso intermedio antes del CounterReducer. Para la query 3 nos encontramos con el problema de que al estar emitiendo pares de <especie, diámetro> no podíamos contabilizar el promedio de a "chunks", porque necesitaríamos saber el tamaño total de antemano para poder hacer el promedio de cada "chunk".
- BiggerThanCollator: lo utilizamos para las queries 2 y 4, que requerían filtrar las entradas que tuvieran valores mayor a lo indicado por el usuario.
- TopNCollator: lo utilizamos para la query 3 que requería filtrar los primeros N resultados.

  En ambos collators hacemos un ordenamiento de los resultados también. En el caso del BiggerThan se le puede pasar opcionalmetne un Comparator, como lo hacemos para la query 5 que pide, además del ordenamiento por las claves, un ordenamiento del par valor.

3.4 Seguridad

Decidimos omitir la utilización de password para la seguridad del cluster, debido a que la misma no está creada para proveer ningún tipo de seguridad. Entendemos que la seguridad de Hazelcast viene dada por la utilización de encripción SSL, la cual probamos, pero para la utilización del trabajo práctico la desestimamos, dado que queríamos visualizar los tiempos de ejecución con el menor ruido posible.

3.5 Extensibilidad

Como lo pide el enunciado, el trabajo está preparado para poder extender su uso a datos de otras ciudades. Si bien fue aclarado que podíamos tomar como únicos valores de columnas los ejemplificados en el enunciado, nos pareció apropiado guardar los nombres de las columnas en un archivo de configuración cities.json en la carpeta de resources (proyecto) o config (deploy) del cliente. De esta forma si se quisieran procesar datos de los árboles de Francia, bastaría agregar una entrada "FRA" con los nombres "custom" de los csv (árboles y barrios) de Francia.

4 Análisis de tiempos

Para probar Hazelcast y las operaciones de concurrencia con varios nodos, utilizamos Docker.
Instanciamos para ello, de 1 a 5 instancias de Docker con el servidor corriendo, y realizamos pruebas con las distintas queries para ver la evolución del tiempo a medida que se agregaban nodos en nuestra arquitectura. Lo que notamos fue un leve aumento del tiempo de ejecución de la operación de MapReduce a medida se agregaban más nodos. Creemos que esto fue influenciado por dos motivos:

1. Las instancias de Docker corren sobre una misma PC, en paralelo. Lo que le quita recursos a las otras instancias.
2. A esa quita de recursos entre sí, se le suma el tiempo de respuesta entre las instancias de Docker y el host, lo que agrega una latencia, dada porque la información viaja por la red virtual, que crece a medida hay más nodos, porque hay más paquetes viajando por la red.

Hicimos pruebas con archivos más grandes, es decir, duplicamos y triplicamos el tamaño del csv provisto por la cátedra, y observamos este patrón con mayor fuerza.
Para las pruebas utilizamos la query 3, que es la que creemos consume más recursos. Con el archivo triplicado y 4 instancias de Docker corriendo, notamos como los recursos del host sobre el cual se realizaban las pruebas eran utilizados en totalidad, lo que llevó a que una query que normalmente tarda 1 segundo, tarde entre 3 y 4 segundos.

Creemos que el patrón observado se debe también a que el archivo es muy chico y los tiempos de respuesta son proporcionalmente más bajos y ante

cualquier perturbación en el ambiente (latencia, algún proceso ocupando recursos, etc), se modifica mucho el tiempo de ejecución, por lo que no lo consideramos una muestra muy real de la utilidad del procesamiento distribuido de Hazelcast. Las pruebas reales las deberíamos hacer con archivos del orden de los GB, y muchas instancias distribuidas por la red, en distintos hosts.

Este procesamiento distribuido, además de agilizar los tiempos, también provee tolerancia a fallos, por lo que la optimización de lo distribuido no solo viene de la mano del aprovechamiento de recursos, sino de proveer una robustez a los fallos, con las opciones de backup y redundancia que tiene configurables Hazelcast.

5 Potenciales puntos de mejora y/o expansión

- Agregado de combiner para la query 3. Habría que considerar qué emitiría el mapper para poder procesar con un combiner los promedios.
- Mejor manejo de excepciones. Actualmente las excepciones están siendo "catcheadas" pero no son tratadas con demasiado detalle. Por ejemplo si el archivo csv tuviera algún problema de formato caería bajo la misma IOException que si hubiera un problema al leer el archivo de configuración cities.json.