Infraestructuras Computacionales para Procesamiento de Datos Masivos: Tarea 3

Sergio Soler Rocha
Universidad Nacional de Educación a Distancia

1. Introducción

El objetivo de este ejercicio es implementar un benchmark en Google Cloud Platform (GCP) para evaluar el desempeño de un despliegue basado en Spark, centrándose en velocidad, rendimiento y uso de recursos. Se busca analizar de un cluster en GCP, utilizando servicios como Dataproc (para ejecución de clústeres de Apache Spark y Apache Hadoop) y Google Cloud Storage (para almacenamiento en la nube). El benchmark permitirá extraer datos y realizar análisis para comprender las implicaciones del rendimiento en diferentes escenarios, facilitando la toma de decisiones sobre la configuración óptima y la mejora continua de la infraestructura en la nube.

El benchmark utilizado es se encuentra en la siguiente url: https://github.com/DIYBigData/pyspark-benchmark/

2. Configuración inicial en GCP

Creamos un clúster implementado en Google Cloud Platform con Dataproc, denominado "nombre-del-cluster". A continuación, se detallan las principales características del clúster:

- Nombre del Clúster: nombre-del-cluster
- Estado: En ejecución
- Región: europe-west6
- Zona: europe-west6-c
- Total de Nodos Trabajadores: 2
- VMs Flexibles: No
- Eliminación Programada: Desactivado
- Bucket de Etapa de Pruebas de Cloud Storage: dataproc-staging-europe-west6-592252024140-2eafoyog

Lo configuramos con dos nodos trabajadores para distribuir eficientemente las tareas de procesamiento. La desactivación de la eliminación programada garantiza la persistencia del clúster, mientras que el almacenamiento en el bucket de etapa de pruebas de Cloud Storage facilita el manejo de datos durante el proceso de ejecución.

Ahora creamos un Bucket en Google Cloud Storage. Este Bucket será destinado para almacenar los datos y los scripts esenciales que serán utilizados en las evaluaciones de rendimiento, es decir, los script generate-data.py, benchmark-shuffle.py y benchmark-cpu.py.

El siguiente paso es Instalar la biblioteca de Google Cloud Storage para Python, desde el cmd escribimos pip install google-cloud-storage. La instalación de la biblioteca de Google Cloud Storage para Python es fundamental para habilitar la interacción y manipulación de objetos almacenados en Google Cloud Storage (GCS) a través de scripts y aplicaciones escritas en Python.

3. Preparación de Datos

Para llevar a cabo las pruebas de rendimiento, se utiliza el script generate-data.py para generar conjuntos de datos de prueba, el cual es un trabajo PySpark. El archivo resultante será un archivo CSV particionado. Estos son los siguientes parámetros:

- –master spark://spark-master:7077: Opción de spark-submit que identifica la ubicación del maestro de Spark.
- –name 'generate-benchmark-test-data': Opción de spark-submit para nombrar el trabajo presentado (opcional-).
- /path/to/test/data/file: Ruta completa donde se generará el archivo de datos de prueba.
- -r num_rows: Número total de filas a generar. Cada fila tiene aproximadamente 75 bytes.
- -p num_partitions: Número de particiones que debería tener el archivo de datos de p

El esquema de los datos producidos por el script es el siguiente:

- value: string (nullable = true)
- prefix2: string (nullable = true)
- prefix4: string (nullable = true)
- prefix8: string (nullable = true)
- float_val: string (nullable = true)
- integer_val: string (nullable = true)

La siguiente línea de código envía un trabajo PySpark al clúster de Dataproc con el script 'generate-data.py', generando 1000000 filas de datos particionadas en 100 particiones y guardando el resultado en un archivo CSV en el bucket creado anteriormente de Google Cloud Storage.

```
!gcloud dataproc jobs submit pyspark \
  --cluster=nombre-del-cluster \
  --region=europe-west6 \
  gs://bucket_1990/generate-data.py \
  -- gs://bucket_1990/Data/datos_generados.csv -r 1000000 -p 100
```

Figura 1: Generación de los datos

4. Ejecución del Benchmark en Dataproc: benchmark-shuffle.py

El script benchmark-shuffle.py se centra en evaluar el rendimiento de operaciones comunes de PySpark en marcos de datos que desencadenan una operación de 'shuffle'. Las operaciones de prueba incluyen:

- Group By and Aggregate: Agrupación y agregación de datos.
- Repartition: Reorganización de datos entre particiones.
- Inner Join: Operación de unión interna entre dos conjuntos de datos.
- Broadcast Inner Join: Unión interna con transmisión de datos pequeños.

A continuación ejecutamos el script benchmark-shuffle.py sobre los datos generados en el apartado anterior. El número de particiones que se utilizarán durante la prueba de repartición será 200.

```
!gcloud dataproc jobs submit pyspark \
  --cluster=nombre-del-cluster \
  --region=europe-west6 \
  gs://bucket_1990/benchmark-shuffle.py \
  -- gs://bucket_1990/Data/datos_generados.csv -n 'shuffle-benchmark' -r 200 \
       -o gs://bucket_1990/benchmark_results/shuffle_results
```

Figura 2: Ejecución del script benchmark-shuffle.py

4.1. Resultados de la ejecución

- **Group By test time** = 15.213665729999775 seconds: Indica el tiempo que toma ejecutar la operación de agrupación (Group By). En este caso, la operación de agrupación ha tomado aproximadamente 15.21 segundos.
- **Repartition test time** = 8.370773898001062 seconds (200 partitions): Muestra el tiempo de ejecución de la operación de repartición (Repartition). Además, indica que durante esta prueba se utilizó un total de 200 particiones. En este caso, a la operación de repartición le ha llevado aproximadamente 8.37 segundos.
- **Inner join test time** = 14.873300867009675 seconds: Indica el tiempo que tomó ejecutar la operación de unión interna (Inner Join). En este caso, la operación de unión interna ha tardado aproximadamente 14.87 segundos.
- **Broadcast inner join time** = 6.686923230998218 seconds: Representa el tiempo de ejecución de la operación de unión interna mediante difusión (Broadcast Inner Join). Aquí, la operación de unión interna mediante difusión ha tomado aproximadamente 6.69 segundos.

5. Ejecución del Benchmark en Dataproc: benchmark-cpu.py

El benchmark de CPU tiene como objetivo evaluar operaciones de PySpark que están principalmente vinculadas a la CPU. A diferencia de otras pruebas, este benchmark se centra en evaluar la velocidad de la CPU y la eficiencia de las tareas, sin considerar E/S de disco o red significativas. Las operaciones de prueba incluyen:

- SHA-512 hashing de una cadena.
- Estimación de Pi con muestras aleatorias y una función Python definida por el usuario.
- Estimación de Pi con muestras aleatorias y solo funciones nativas de Spark.

Cada operación se cronometra de forma independiente. Los resultados proporcionan información valiosa sobre el rendimiento de las operaciones de CPU en PySpark, y los tiempos se pueden comparar entre ejecuciones para evaluar la eficiencia del entorno PySpark.

```
!gcloud dataproc jobs submit pyspark \
    --cluster=nombre-del-cluster \
    --region=europe-west6 \
    gs://bucket_1990/benchmark-cpu.py \
    -- gs://bucket_1990/Data/datos_generados.csv -s 5000000000 -p 10 -n 'cpu-benchmark' \
        -o gs://bucket_1990/benchmark_results/cpu_results
```

Figura 3: Ejecución del script benchmark-cpu.py

5.1. Resultados de la ejecución

Los resultados del benchmark de CPU muestran el tiempo que tardó cada operación en ejecutarse:

- **SHA-512 benchmark time**: Este resultado indica el tiempo que le ha llevado calcular 100,000 hashes SHA-512. En este caso, fue de aproximadamente 14.61 segundos. Esta prueba mide la velocidad de procesamiento para operaciones intensivas en CPU, como la generación de hashes.

- **Calculate Pi benchmark**: Para esta prueba, se han tomado 5 mil millones de muestras para estimar el valor de Pi. El resultado muestra que ha tardado alrededor de 852.47 segundos (algo más de 14 minutos) realizar esta operación.
- **Calculate Pi benchmark using dataframe**: Similar a la prueba anterior, esta vez, la estimación de Pi se ha realizado utilizando manipulaciones de DataFrame. Ha tomado aproximadamente 74.67 segundos. Esto proporciona una comparación entre el rendimiento de las operaciones puras de PySpark y aquellas que involucran manipulaciones de DataFrame.

6. Conclusiones

A continuación mostramos la tabla en la cual si se superan esos tiempos está considerado de bajo rendimiento para los parámetros que hemos utilizado, es decir, 1000000 rows y 100 partitios [1]:

Test	**Tiempo** (segundos)
Group By Test Time	> 5.34
Repartition Test Time	> 3.200
Inner Join Test Time	> 2.500
Broadcast Inner Join Test Time	> 1.973
SHA-512 Benchmark Time	> 4.620
Calculate Pi Benchmark	> 228
Calculate Pi Benchmark Using Dataframe	> 12.9

Cuadro 1: A partir de estos tiempos se considera un bajo rendimiento

Vemos que el tiempo de todas las pruebas es mayor que los valores límites del Cuadro 1 lo que indica un bajo rendimiento del cluster creado. Sería interesante ajustar la configuración del clúster, como la cantidad de nodos o tipo de máquinas virtuales, para mejorar el rendimiento en las pruebas mencionadas, pero debido a las limitaciones de la cuenta esto no ha sido posible.

Como conclusión final, la implementación de benchmarks en el clúster de GCS proporciona una base valiosa para la optimización continua y la toma de decisiones informadas para garantizar un rendimiento eficiente y rentable.

Referencias

[1] https://openbenchmarking.org/test/pts/spark&eval=8dbf91eee81b8c1295c3d871d534bd6055fa787b#metrics

[2] https://en.wikipedia.org/wiki/Benchmark_(computing)

[3] https://github.com/DIYBigData/pyspark-benchmark/