Lab 5 – Apache Spark

CC5212-1 – April 9, 2025

Hoy vamos a analizar las series de TV para obtener su rating promedio por episodio y una lista de sus episodios mejor evaluados. Usaremos datos de IMDb (los mismos que en el Laboratorio 4), que tienen el siguiente formato:

```
375095      9.3       The Wire       2002       tvSeries       null
  4952      7.7       The Wire       2002       tvSeries       Ebb Tide (#2.1)
  6208      8.2       The Wire       2002       tvSeries       The Detail (#1.2)
  3997      8.8       The Wire       2002       tvSeries       Misgivings (#4.10)
  3923      8.6       The Wire       2002       tvSeries       Home Rooms (#4.3)
  ...
```

Las columnas son: (1) Número de votos, (2) Calificación promedio, (3) Nombre de la serie, (4) Año de la serie, (5) Tipo, (6) Nombre del episodio. Cuando la columna (6) es nula para una serie de TV, la información de votación corresponde a la calificación general de la serie (en IMDb, se puede votar tanto en la página de la serie como en episodios individuales).

Los datos están almacenados en HDFS en la siguiente ruta: hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings.tsv. Queremos calcular tuplas de salida con el siguiente formato:

```
The Wire#2002       -30- (#5.10)|Middle Ground (#3.11)       9.6       8.583
```

Donde las columnas representan: (1) Clave de la serie (name#year), (2) Lista de episodios con la mejor calificación, separados por `'|'`, (3) Mejor calificación de un episodio (que corresponde a los episodios listados en (2)), (4) Calificación promedio de todos los episodios, excluyendo la calificación general de la serie.

Usaremos Spark para el procesamiento. Para probar, `hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings-two.tsv` es un archivo de ejemplo pequeño con dos series y una película irrelevante que debes filtrar; una de las series es The Wire (2002), cuyos resultados deberían coincidir con los indicados anteriormente (salvo pequeñas diferencias de redondeo). Ten en cuenta que la salida real generada por Spark usará comas y corchetes en lugar de tabuladores, lo cual es aceptable: la sintaxis de las líneas no es importante, sino el contenido de cada una. **Además, Spark suele generar múltiples archivos de salida del tipo `part-NNNNN` en el directorio de resultados**; todos ellos forman parte del output. (Puedes usar "`part-*`" en combinación con `cat` para concatenar las diferentes partes, donde `*` actúa como wildcard.)

Existen múltiples opciones para ejecutar Spark, incluyendo Java, Scala, Python y R. Entre estas opciones, tanto Java como Scala pueden considerarse "nativas" ya que – al igual que Spark – se ejecutan en la Java Virtual Machine (JVM). El servidor que usaremos puede ejecutar Java, Scala y Python. Por tanto, debes entregar el trabajo en Java, Scala o Python.[1]

- Descarga el proyecto de código desde u-cursos. El proyecto contiene ejemplos en Java, Scala y Python como se describe a continuación. Los más relevantes son las implementaciones de `AverageSeriesRating` que te ayudarán a comenzar usando Spark para calcular el rating promedio de los episodios de cada serie. Luego podrás extender esto.

  - Java 8+: Si abres el proyecto en Eclipse, en la carpeta `src` encontrarás algunas clases de utilidad y ejemplos para comenzar. Revisa `AverageSeriesRating.java`, que calcula el rating promedio de los episodios de la serie. Estos ejemplos usan lambdas de Java (version 8+). Para ejecutar este ejemplo, necesitas construir un archivo JAR. En el proyecto en Eclipse, haz clic derecho en `build.xml`, luego selecciona `Run As ...`, asegúrate de que `dist` esté seleccionado y haz clic en `Run`. y un archivo JAR aparecerá en la carpeta `dist`. Cópialo al servidor (como en el laboratorio de Hadoop) y ejecuta:

    ```
    spark-submit --master spark://cluster-01:7077 mdp-spark.jar AverageSeriesRating
    hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings-two.tsv
    hdfs://cm:9000/uhadoop2025/USERNAME/series-avg-two-java/
    ```

    Los resultados se almacenarán en HDFS en la carpeta indicada por el último argumento.

[1] Debes usar Spark Core, no, por ejemplo, Spark SQL.

- JAVA 7: En la carpeta `j7` encontrarás ejemplos sin lambdas que utilizan clases anónimas de Java 7 como referencia. ¡Las lambdas realmente hacen el código mucho más conciso!
- PYTHON: Si deseas probar Python, en la carpeta `py` encontrarás ejemplos para PySpark, que interpreta scripts de Python para trabajos de Spark. Por ejemplo, `AverageSeriesRating.py` puede ejecutarse con (un solo comando):

  ```
  spark-submit --master spark://cluster-01:7077 AverageSeriesRating.py
  hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings-two.tsv
  hdfs://cm:9000/uhadoop2025/ USERNAME /series-avg-two-py/
  ```

  Para desarrollar un script, ten en cuenta que, similar a Pig, puedes iniciar un shell local con `pyspark` y ejecutar cada comando uno por uno. Nuevamente, usa esto solo para desarrollo con un ejemplo pequeño. En el shell, puedes llamar a `rdd.collect()` para ver el contenido de un RDD llamado `rdd`.
- SCALA Si deseas probar Scala, en la carpeta `scala`, encontrarás ejemplos relevantes. Por ejemplo, puedes ejecutar `AverageSeriesRating.scala` siguiendo estos pasos: Primero cambia el valor de `fileout` en el archivo Scala (reemplazando `<user>`con tu carpeta HDFS) Luego ejecuta el siguiente comando: [2]

  ```
  spark-shell --master spark://cluster-01:7077 -i AverageSeriesRating.scala
  ```

  Los resultados se almacenarán en HDFS en la carpeta indicada por el valor de `fileout` en el script. Al ejecutarlo, el proceso terminará en la shell de Spark. Puedes usar el comando `:quit` para salir. Si deseas abrir una consola local para probar comandos individualmente, simplemente ejecuta `spark-shell`. Nuevamente, usa esto solo para desarrollo con un ejemplo pequeño, ya que ejecuta Spark en modo local, no en el cluster. Puedes llamar a `rdd.collect()` para ver los contenidos de un RDD llamado `rdd`.

- Ahora, la tarea principal: tu objetivo es usar Spark Core para generar, para cada serie: (1) el rating promedio de episodios, (2) el mejor rating de episodio, y (3) los nombres de los episodios con el mejor rating, según el ejemplo en la introducción. Debes copiar el archivo `AverageSeriesRating` en el lenguaje de tu elección a un nuevo archivo llamado `InfoSeriesRating` y comenzar a extenderlo para buscar información sobre los mejores episodios. Al hacer esto, debes seguir el ejemplo de `AverageSeriesRating`, leyendo los datos de entrada desde HDFS y escribiendo los resultados en HDFS. Debes probar esto primero con el pequeño ejemplo proporcionado antes de ejecutarlo con el archivo completo. Para los casos de Python y Scala, recomendamos encarecidamente desarrollar el nuevo script línea por línea en su respectivo shell, revisando los resultados a medida que avanzas con el ejemplo. Además, asegúrate de cachear cualquier RDD que utilices más de una vez. Por favor, consulta el ejemplo de salida en la introducción para `The Wire#2002` como referencia de la tarea que debes implementar; recuerda nuevamente que no es necesario replicar exactamente la sintaxis mostrada para cada línea, sino únicamente incluir la información correcta en cada una.
- Una vez que tu código esté listo y funcione correctamente con el archivo pequeño, deberás ejecutarlo con el conjunto completo de datos de ratings ubicado en: `hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings.tsv` y verificar los resultados obtenidos.
- OPCIONAL: En lugar de extender `AverageSeriesRating`, puedes calcular la suma de ratings, conteo de episodios, rating máximo de episodios y nombres de episodios con rating máximo todo en una sola transformación `reduceByKey` o `aggregateByKey` (usando `map` después para calcular el promedio dividiendo la suma entre el conteo). ¡Inténtalo! Debería ser la opción más eficiente. (Si ya resolviste el laboratorio de esta forma, está perfectamente bien).
- OPCIONAL: Intenta programar los ejercicios del Laboratorio 3 y Laboratorio 4 en Spark (o piensa cómo lo harías)!
- ENTREGAR: (1) Tu solución `InfoSeriesRating` en Java, Scala o Python, (2) La tupla de salida generada para la serie "`The Simpsons#1989`" y otra tupla que no aparezca en el archivo de ejemplo. Puedes incluir (2) como comentario en la entrega o al inicio del código.

[2] En la práctica, deberíamos compilar un archivo JAR para el script Scala, pero esto es complicado de configurar para todos, por lo que usamos esta alternativa. Para compilar un JAR a partir de código Scala, puedes usar SBT: Scala Build Tool. El formato del código sería ligeramente diferente, requiriendo la definición de un objeto y un método main, etc.