Analizando un catálogo de un billón de objetos: Apache Spark para Físicos

Saul Andersson Rojas Coila
Departamento de Ciencia de la Computación
Universidad Católica San Pablo
Perú, Arequipa-Arequipa
Correo electrónico: saul.rojas@ucsp.edu.pe
18 de Octubre, 2022

***Resumen*—En este paper se mostrarán algunos ejemplos haciendo uso de Spark para el análisis de datos astronómicos con la librería de fuente de datos para spark: SparkFits. Con la aparición de spark y su uso con python (pyspark) la comunidad científica puede hacer uso de los conocimientos en programación para obtener mejores resultados a la hora del análisis de grandes volúmenes de datos, en este caso de uso los físicos no necesitan de mucho conocimiento en programación para obtener métricas relevantes debido a que python provee una curva de aprendizaje baja a la hora de otener datos, todos los ejemplos con el código fuente los encuentra en https://github.com/webtaken/SparkFitsTutorial.git.**

***Palabras Clave*—** apache spark, big data, spark fits.

1. Introducción

El análisis masivo de datos científicos se hace cada vez una tarea más imprescindible en el mundo académico, gracias a los avances en dispositivos que brindan métricas en gran volumen se genera la necesidad de procesar de una forma efectiva y óptima estos datos a fin de que los científicos puedan obtener conclusiones más rápidas acerca de los fenómenos que estudian.
La influencia de la programación en el campo científico también se hace muy notoria ya que brinda mayor flexibilidad a la hora de obtener mayor información sobre los datos obtenidos. En el mundo académico el lenguaje de programación `python` es muy utilizado por sus múltiples librerías para la visualización y su curva de aprendizaje no muy elevada, también con la aparición de Apache Spark [1] un *cluster* para el procesamiento en *big data*, se abre un nuevo camino al análisis científico sin la necesidad de escalar verticalmente, sino aprovechando las ventajas del escalamiento horizontal que se encuentra en la mayoría de universidades actualmente.
En este paper corto se replicará algunos experimentos hechos en el paper *Apache Spark for physicists* [2] el cual contiene ejemplos de cómo se pueden analizar grandes cantidades de datos sobre galaxias con un cluster dde computadoras relativamente pequeños para centros de estudio como universidades.

2. Configuración de los experimentos

Por el momento debido a que no se poseen los recursos físicos del paper original, 9 máquinas con 36GB de memoria RAM cada una con 18 cores los cuales procesarían $\approx 170GB$ de información, en lugar de ello simularemos los experimentos con un dataset menor, solo se tendrá una PC la cual tendrá un maestro y un esclavo.
También para la réplica de los experimentos a menor escala se usará ***SparkFits*** [3] un proyecto de fuente de datos astronómicos para Spark, en la versión 12 de escala. Cada ejemplo de código a ejecutarse se hará de la siguiente manera.

```
$ ./spark-submit --packages
"com.github.astrolabsoftware:
spark-fits_2.12:1.0.0"
--master <path/to/master>
<path/to/file.py> <...>
```

3. Experimentos

En esta sección se mostrará los diferentes experimentos que se pueden hacer con Spark para el trabajo con datos astronómicos.

3.1. Lectura de datos

Los archivos FITS son tablas que contienen datos astronómicos, estos también pueden ser transformados fácilmente a imágenes y viceversa, para leer un archivo `.fits` especificamos el formato en el método `.format()` al momento de hacer la lectura:

```
gal = spark.read.format("fits")\
        .option("hdu", 1)\
        .load(args.inputpath)\
        .select("RA", "Dec")
gal.printSchema()
gal.show()
```

Listing 1. Lectura de un archivo FITS

En el código 1 se especifica el formato de nuestro archivo, solo seleccionaremos las columnas `RA` y `Dec`, a partir de estos datos podemos trabajar ya con estas tablas en Spark. El código 1 nos devuelve el *schema* y nos muestra las primeras columnas del *dataset*.

```
# El Schema
root
 |-- RA: float (nullable = true)
 |-- Dec: double (nullable = true)

# Solo se muestra una parte del dataset
+----------+--------------------+
|        RA|                 Dec|
+----------+--------------------+
|  3.448297| -0.3387486324784641|
|  4.493667| -1.4414990980543227|
|  3.787274|  1.3298379564211742|
```

```
|  3.423602|-0.29457151504987844|
| 2.6619017|  1.3957536426732444|
| 4.0582724|  0.6997096205542732|
| 2.7494416|  1.3141908888028095|
|  5.603175|  1.0155749246503523|
| 6.0548716| 0.46148959939271705|
|  2.409234| -1.3573596005972968|
|  4.974555|  1.0644211404212554|
| 3.3231447|  0.6057694370339766|
| 3.5691292|  1.4388227773007016|
| 5.8156953| -1.1239912657406808|
|0.44633272| -1.2950476251356782|
| 0.5474495| 0.05931320874902668|
|0.12703593|-0.44003542475478663|
|  5.231505|   1.445992419083169|
|  4.889303|  0.6401693029716138|
| 5.4664474|-0.16966463268737808|
+----------+--------------------+
only showing top 20 rows
```

Listing 2. Resultados de la lectura

Y también podemos obtener un poco de información estadística.

```
gal.describe(["RA", "Dec"]).show()
```

Listing 3. Información estadística

```
+-------+------------------+--------------------+
|summary|                RA|                 Dec|
+-------+------------------+--------------------+
|  count|             20000|               20000|
|   mean| 3.115512763023628|-0.00168123882615...|
| stddev|1.8249500061863815|   0.905928569761147|
|    min|       4.552145E-4| -1.5697126529721062|
|    max|         6.2830467|   1.570541054945147|
+-------+------------------+--------------------+
```

Listing 4. Datos estadísticos

3.2. Imágenes

También se pueden transformar estos datos a imágenes de índice *healpix*, una manera de visualizar estos datos astronómicos haciendo uso de un mapeo, en el siguiente código se muestran los primeros 10 píxeles de la imagen.

```
import healpy as hp
import numpy as np
import matplotlib.pyplot as pl

def dec2theta(dec):
    """
    Convert Declination into Theta
    Parameters
    ----------
    dec : Double
        Declination angle (in radian)
    Returns
    ----------
    theta : Double
        Theta angle (in radian)
    """
    return np.pi / 2. - dec

def ra2phi(ra):
    """
    Convert RA into Phi
    Parameters
    ----------
    ra : Double
        Right Ascension angle (in radian)
    Returns
    ----------
    phi : Double
        Phi angle (in radian)
    """
    return ra
gal = spark.read.format("fits")\
        .option("hdu", 1)\
        .load(args.inputpath)\
        .select("RA", "Dec")
nside = 16
## Pyspark DataFrames do not have "map" method - so you need
## to convert it to RDD first.
rdd = gal.select(gal["RA"], gal["Dec"])\
    .rdd\
    .map(lambda x: hp.ang2pix(nside, dec2theta(x[1]), ra2phi(x[0])))
## Show the first 10 pixels
print(rdd.take(10))
```

Listing 5. Conversión a imágenes

Ya que los *dataframes* no poseen la opción de mapeo de sus elementos lo que se hace es transformar estos a `RDDs` los cuales pueden ser procesados por varios nodos y se pueden mapear. Mostramos la imagen con la librería `matplotlib`.

```

hitcount = rdd.map(lambda x: (x, 1))
myPartialMap = hitcount.countByKey()

myMap = np.zeros(12 * nside**2)
myMap[list(myPartialMap.keys())] = list(
    myPartialMap.values())

## Fake rotation of the coordinate system of the map for visualization
hp.mollview(myMap, coord="CG")
pl.show()
```

Listing 6. Gráficamos la imagen

El resultado se ve en la imagen 1.

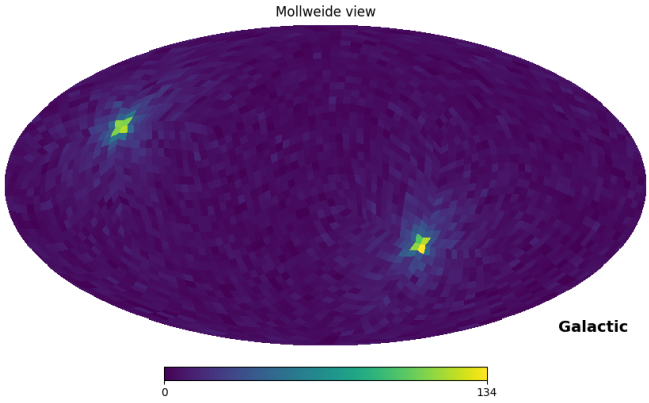

Figura 1. Mostrando la imagen de la transformación.

3.3. Histogramas

También se pueden crear histogramas que ayuden a mejorar la visualización de los datos en los archivos FITS.

```
    minmax = gal.select(F.min("RA"), F.max("RA"))
    minRA = minmax[0]
    maxRA = minmax[1]
    Nbins = 100
    dRA = (maxRA - minRA) / Nbins

    binRA = gal.select(gal.RA,\
            (( gal.RA-minRA-dRA/2)/dRA).astype("
    int")\
            .alias("bin"))

    h = binRA.groupBy("bin")\
        .count()\
        .orderBy(F.asc("bin"))

    pd= h.select("bin",\
        (minRA+dRA/2+h.bin*dRA).alias("RAbin") ,\
        "count")\
        .drop("bin")\
        .toPandas()

    binNum = F.udf(lambda z: int ((z-minRA-dRA/2)/
    dRA))
    RAbin = gal.select(gal.RA,\
    binNum(gal.RA)\
        .alias("bin"))

    @pandas_udf ("float", PandasUDFType.SCALAR)
    def binNumber(z):
        return pd.Series((z-minRA)/dRA)
    zbin=gal.select(gal.RA,\
    binNumber("z").astype("int")\
        .alias("bin"))
```

Listing 7. Conversión a imágenes

4. Conclusión

Se han mostrado ejemplos de cómo se puede realizar diferentes ejemplos con Spark y Spark Fits para el trabajo científico, resaltamos que el escalamiento horizontal hoy en día es muy importante si se desea analizar grandes volúmnees de datos también es alcanzable en la mayoría de centros de estudio universitarios, y muy fácil de desplegar teniendo las herramientas como Apache Spark, también con el uso de Pyspark la aplicación de estos métodos no requiere de un conocimiento profundo en programación.

Referencias

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," Jun. 2010. [Online]. Available: https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets

[2] S. Plaszczynski, J. Peloton, C. Arnault, and J. E. Campagne, Analyzing billion-objects catalog interactively: Apache Spark for physicists. arXiv, 2018. doi: 10.48550/ARXIV.1807.03078.

[3] J. Peloton, C. Arnault, and S. Plaszczynski, FITS Data Source for Apache Spark. arXiv, 2018. doi: 10.48550/ARXIV.1804.07501.