Proyecto HPC

Lope Carvajal[1], Samuel Sarabia[2]

Abstract
En el presente artículo se explicará el funcionamiento y los procedimientos que realiza el algoritmo k-means. Además se abordará el tema de el análisis de eficiencia en la ejecución de este por medio de SPARK. El análisis abarca una diversa comparación de resultados de tiempos de ejecución en diferentes circunstancias donde se ha probado el código realizado. Además de esto, se documentará el código realizado para solucionar este análisis, dando a entender con mayor claridad la forma en que se está ejecutando y estamos llegando a dichos resultados. Para la implementación del algoritmo utilizamos Pyspark, haciendo énfasis en la librería MLlib para el uso de la implementación de K-Means. Por otro lado, se realizará una comparación de eficiencia entre una implementación previamente realizada de HPC y esta, enfocada a Big Data.

Keywords
Big Data, K-Means, Pyspark, Clusters, Spark, Pipeline, RDD, Dataframe, Python

[1] *Ingeniería de Sistemas, Universidad EAFIT, Medellin, Colombia, lcarva12@eafit.edu.co*
[2] *Ingeniería de Sistemas, Universidad EAFIT, Medellin, Colombia, ssarabia@eafit.edu.co*

Contents

Introducción

El proyecto presentó un gran reto para nosotros, y este fue el aprendizaje de la herramienta Spark, junto con implementaciones que se encuentran implicitamente implicadas con el uso de la misma, tal es el caso de HDFS por dar un ejemplo. En comparación al programa previamente realizado para el módulo de HPC, el desarrollo se hizo más corto en el proceso de codificación, pues gracias al uso de la librería MLlib no fue necesario realizar el algoritmo de K-Means explicitamente. Pero por otro lado la comprensión de la librería y como se debíá ejecutar el código se hizo similarmente dificultoso.

INCLUIR RESULTADOS DE LAS PRUEBAS

1. Marco Teorico

1.1 Definición del Problema

Para abarcar el análisis del presente documento, es importante dar una descripción detallada del problema que vamos a solucionar. Como se había mencionado anteriormente, se aborda el problema de clasificación de textos por medio de su similaridad utilizando el algoritmo de K-means, con encontrando el cálculo de las distancias entre documentos por medio de la aplicación de la medida numérica Tf-idf(Term frequency -Inverse Document frequency)

1.2 Análisis de Texto

Para poder comenzar a comparar para encontrar qué tan similares son dos documentos, debemos primero convertirlos en alguna estructura matemática de datos. Primero es necesario eliminar las palabras conectoras y artículos del lenguaje en el que se encuentren los textos. El algoritmo sólo eliminará las palabras conectoras de un lenguaje. La librería nos brinda una lista con estas palabras que después eliminamos. A partir de la lista con los elementos relevantes se aplica el algoritmo de Tf-idf, donde se obtiene el número de veces que un término aparece en un documento y la relevancia del mismo en la totalidad de documentos que se están evaluando.

$$IDF(t,D) = \log \frac{|x|+1}{DF(t,D)+1} \tag{1}$$

1.3 K-means

K-means es uno de los algoritmos no supervisados más simples para resolver el problema de agrupamiento. El proceso sigue una forma simple de clasificar un dataset en un número predeterminado de clusters. La idea en resumen es definir k

centros, uno por cada cluster. Estos centros, en algunos casos, son asignados en primera instancia de forma aleatoria, aunque existen diversas implementaciones de este paso para hacer el algoritmo más efectivo, ya que diferentes ubicaciones de los centros conducen a diferentes resultados. El siguiente paso es tomar cada punto del dataset otorgado y relacionarlo con el centro más cercano, cuando no quedan más puntos, el primer paso es completado y un agrupamiento simple es realizado. En este punto se deben recalcular k nuevos centros ubicados en el promedio de la agrupación realizada en la primera iteración. Después de tener estos k nuevos centros, una nueva agrupación debe ser realizada entre éstos y el dataset inicial. De este modo se crea un ciclo que termina cuando los centros nuevos son equivalentes a los previamente calculados y arroja el resultado.

$$MIN \sum_{j=1}^{k} \sum_{i=1}^{n} ||x - \mu_j||^2 \tag{2}$$

2. Análisis y diseño mediante analítica de datos

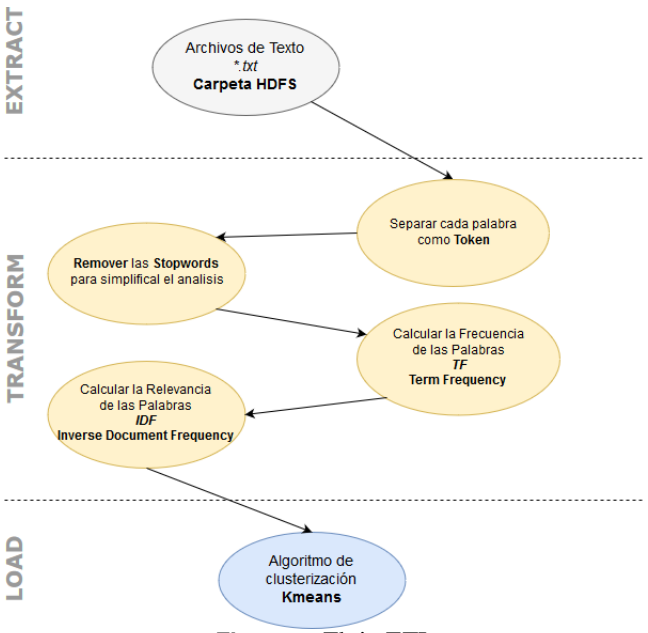

Figure 1. Flujo ETL

2.1 Extract

- Para la extracción de datos se accede a un directorio ubicado en el HDFS de la máqina, de allí se extraen todos los archivos de texto por medio del método wholeTextFiles() contenido en Pyspark, que devuelve un RDD que contiene estos datos.

2.2 Transform

- Se separan las palabras como Tokens, y se guardan como una lista con uno en cada posición.
- Se realiza un proceso de remoción de stopwords para simplificar el análisis y se ignoren las palabras que no son útiles para la comparación.
- Se calcula la frecuencia de las palabras en cada documento y la relevancia de las mismas dentro de todos los documentos.

2.3 Load

- Se pasan los datos previamente procesados al algoritmo de clusterización para que termine la ejecución.

3. Implementación

Hay tres lenguajes compatibles con Spark, estos son:

- Scala
- Java
- Python

Para nuestra implementación decidimos utilizar Python, más específicamente Pyspark.

En primer lugar, se crea un RDD con todos los archivos de texto de una carpeta específica.

```
files = sc.wholeTextFiles("hdfs:///user/
    ↪ lcarva12/pruebaKmeans")
```

Posteriormente se crea un esquema para convertirlo en un dataframe que contiene el path y el texto del archivo.

```
schema = StructType([StructField ("path"
    ↪  , StringType(), True) ,
StructField("text" , StringType(), True)
    ↪ ])
df = spark.createDataFrame(files,schema)
```

En base al texto de cada uno de los archivos se crea una lista con todas las palabras separadas.

```
tokenizer = Tokenizer(inputCol="text",
    ↪ outputCol="tokens")
```

A esta lista se le extraen las palabras que no son necesarias, como artículos y conectores.

```
remover = StopWordsRemover(inputCol="
    ↪ tokens", outputCol="
    ↪ stopWordsRemovedTokens")
```

Después de esto se realiza el proceso previamente mencionado para calcular el Tf-idf

```
hashingTF = HashingTF(inputCol="
    ↪ stopWordsRemovedTokens", outputCol
    ↪ ="rawFeatures", numFeatures=2000)
idf = IDF(inputCol="rawFeatures",
    ↪ outputCol="features", minDocFreq
    ↪ =5)
```

En base a esto se ejecuta el algoritmo de K-Means.

```
kmeans = KMeans(k=5)
pipeline = Pipeline(stages=[tokenizer,
    ↪ remover, hashingTF, idf, kmeans])
model = pipeline.fit(df)
results = model.transform(df)
```

Todo este proceso se realiza mediante la definición y posterior ejecución de un pipeline donde se encuentran todas las operaciones previamente mencionadas.

Figure 2. Resultados ejecución con 500 archivos

4. Pruebas

Se realizaron diversas pruebas para comparar el rendimiento y la diferencia entre la implementación realizada para el módulo de HPC y el de Big Data. Para estas pruebas se utilizaron diferentes cantidades de documentos del dataset proveído por Gutenberg(Incluyendo una prueba donde se realiza clustering de todos los documentos) para analizar como se desenvolvía el algoritmo en diferentes escenarios. La realización del primer test se justificó bajo la búsqueda de una comparación con la ejecución paralela de K-Means relaizada en el módulo pasado. Para estas pruebas se utilizaron tres escenarios:

- 500 documentos del dataset Gutenberg clasificados en 4 clusters.
- 1000 documentos del dataset Gutenberg clasificados en 6 clusters.
- Dataset Gutenberg completo clasificados en 8 clusters.

Gutenberg completo (8 clusters)	
Big Data	HPC
2609	62005

Figure 3. Resultados ejecución Gutenberg completo

1000 documentos (6clusters)	
Big Data	HPC
922	2553

Figure 4. Resultados ejecución 1000 documentos

Con estos resultados se puede apreciar que la implementación de Big Data es más eficiente para realizar el clustering de documentos por medio del algoritmo de KMeans. La diferencia de tiempo de ejecución es por poco abismal cuando se trata de una cantidad relativamente grande de archivos, como son los aproximadamente tres mil archivos de texto contenidos en la biblioteca de Gutenberg.

500 documentos(4 clusters)	
Big Data	HPC
259	915

Figure 5. Resultados ejecución 500 documentos

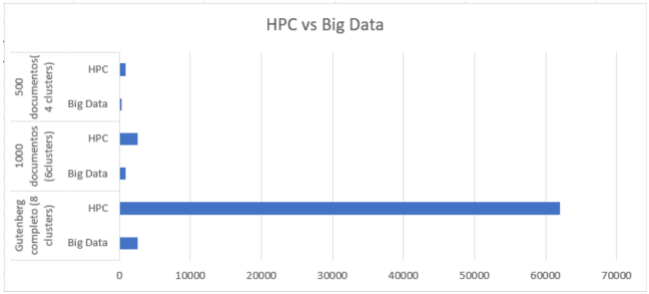

Figure 6. Comparación de tiempos de ejecución en segundos

5. Conclusión

Spark provee grandes ventajas para la resolución de problemas de Big Data como el tratado en este documento, siendo este la clasificación de documentos por medio del algoritmo de K-Means.

Esta implementación fue sustancialmente más sencilla que la del módulo pasado, pues las herramientas disponibles para atacar el problema por medio de Pyspark facilitan el proceso y hacen que este sea mucho más corto.

Además de esto, se obtiene un código más legible y entendible, ya que muchas partes de la secuencia son casi legibles en lenguaje natural.

La implementación actual superó con creces la realizada para el módulo de HPC en términos de rendimiento, pues gracias a la ejecución de pruebas se comprobó que la diferencia es muy singificativa, alcanzando que en una gran cantidad de archivos a procesar se tardara hasta 27 veces menos la clasificación de los mismos.

Una causa de esta diferencia en tiempo de ejecución se debe a nuestra incapacidad de paralelizar la sección del algoritmo de K-Means en el proyecto pasado, donde se distribuyó únicamente la sección donde se procesaban los documentos para entregarselos a dicho algoritmo.

Esta dificultad se tuvo por la forma en que planteamos el problema, ya que no hicimos una división correcta del mismo para que fuera paralelizado más facilmente.

Acknowledgments

Utilizamos el git[3] como base para la implementación de nuestra solución.

References

[1] Anna Huang *Similarity measures for text document clustering.* Proceedings of the Sixth New Zealand, April):49–56, 2008.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008

[3] MLlib, *https://spark.apache.org/mllib/*

[4] Pyspark, *https://pypi.python.org/pypi/pyspark*