Tópicos Especiales en Telemática: Proyecto 3 – Clústering de Documentos a partir de Métricas de Similitud basado en big data

Mateo Murillo Penagos murill5@eafit.edu.co Universidad EAFIT
Mariana Narvaez Berrio mnarvae3@eafit.eedu.co Universidad EAFIT

Resumen Este proyecto consiste en diseñar e implementar una aplicación con tecnología y modelo de programación distribuida en Big Data, específicamente con tecnología Hadoop y Spark que permita agrupar (clustering) un conjunto de documentos utilizando el algoritmo de k–means [1] y una métrica de similaridad entre documentos.

***Index Terms*—Clúster, big data, Clustering, k-means, spark , Algoritmo, Alto rendimiento, hadoop.**

I. Introducción

El rápido crecimiento de la tecnología y la ciencia en los últimos años, ha ocasionado un aumento inminente en la cantidad de información que se produce a diario. Dicha información muchas veces obtenida como datos estructurados, no estructurados o semiestructurados debe ser tratada en tecnología no convencional que posea una capacidad y velocidad de procesamiento suficiente, para efectuar eficientemente las operaciones requeridas y resolver ciertos problemas a los que nos enfrentamos frecuentemente. Este documento trata uno de los múltiples problemas con el que se topa la computación de alto rendimiento a diario y es el clustering, técnica que utilizan programas y compañías muy exitosas en la actualidad para agrupar documentos y a partir de esto realizar todo tipo de operaciones. A continuación, veremos una demostración a pequeña escala sobre cómo se realiza esto con tecnologías actuales tal y como lo es spark. Además se hará un análisis posterior donde se evaluara las diferencias entre el procesamiento con tecnología convencional (mp4py) y otra mas actual como spark.

II. Marco Teórico

El problema básicamente se sintetiza en 2 subproblemas candidatos a ser procesados en Big Data con procesamiento masivo paralelo de datos:

A. Subproblema 1

Diseño e implementación de una función de similaridad entre 2 documentos, esto es: dato di y dj , se define una función de de similaridad, fs, entre di y dj como: fs(di, dj) [min_val, max_val]. Hay varios algoritmos de similaridad [MRS08], de los cuales deberá elegir uno que se preste para ser paralelizado, entre los algoritmos más usados se encuentra: La distancia Euclidiana, Coseno, Jaccard o Pearson entre otros.

B. Subproblema 2

Una vez se tiene definida e implementada la función de similaridad, se puede ejecutar el algoritmo de agrupamiento (clustering), que permite dividir el conjunto de documentos en un número de subgrupos. Estos algoritmos de clustering pertenecen a la categoría de Aprendizaje de Máquina NO supervisado. Uno de los algoritmos más usados es k–means, el cual permite dividir el conjunto de documentos en k subgrupos. Este algoritmo requiere conocer de antemado k, por lo cual, está dentro del alcance de este proyecto3, realizar varias simulaciones con k diferentes, el valor de k es cálculado de forma heurística, teniendo en cuenta algunas de las siguientes consideraciones: 1) variando apriori k en algún intervalo dado: ej: 3 k 5. 2) conociendo en mayor detalle el dataset, lo que permita inferir un k, por ejemplo en un conjunto de noticias, se podría sugerir un valor central de k alrededor del número de géneros presentes de tipos de noticias, ej: 10 géneros de noticias, 8 k 12.

En resumen, se requiere diseñar una solución donde se reduzcan los tiempos para el análisis de cada uno de los documentos con respecto a los demás documentos y poder hacer un comparativo teniendo como línea base el tiempo de procesamiento de este problema en un acercamiento HPC con respecto a uno diseñado e implementado para Big Data, haciendo un análisis de las herramientas, la estrategia, hallando la aceleración del algoritmo HPC con respecto al distribuido en Big Data.

III. Implementación

Información:

El codigo se divide en 2 secciones, primero los transformadores y por ultimo los estimadores (Acciones). Dentro de esta estructura tambien se puede separa el algoritmo en 3 partes, una parte inicial donde se leen los documentos y se almacenan en una base de datos, el calculo del vector TF-IDF y por ultimo la ejecución del algoritmo de kmeans, siendo todas estas funciones dadas por las librerias de Spark.

Lectura de documentos:

La lectura de documentos se realiza con las funciones nativas de Spark y la variable de contexto del mismo, solicitandole que lea un path en el que encontrara muchos documentos y almacenando todos estos documentos en una

variable RDD, en un formato en el que me de la direccion del archivo y el texto que este contiene, de la siguiente manera:

```
    sc = SparkContext('local')
spark = SparkSession(sc)

    documents = sc.wholeTextFiles(path)
```

una vez se tiene esta estructura, es facilmente almacenada en una base de datos hive; para esto primero se debe crear un dataframe con la estructura necesaria para almacenarlo y guardar en este el contenido de la variable RDD con los documentos. De la siguiente manera:

```
    schema = StructType([StructField ("path" , StringType(), True) , StructField("text" , StringType(), True)])

    docDataFrame = spark.createDataFrame(documents,schema)
```

Calculo del vector TF-IDF

Para realizar el calculo del vector TF-IDF se utiliza una libreria que se encuentra incluida dentro de Apache Spark. Para el calculo de este vector es necesario realizar 2 acciones, que son: el calculo de TF (Term Frecuency) y el calculo del vector IDF (Inverse Document Frecuency) por separado, pero antes de poder calcular estos es necesario dividir la entrada que tenemos la cual es un String, en una variable iterable que este compuesta por las palabras que conforman el texto.

Nota: El vector TF-IDF realiza un calculo de peso para cada palabra, y a aquellas palabras que son repetidas una gran cantidad de veces y se encuentran en varios documentos les resta peso, con el objetivo de quitarle importancia a las palabas auxiliares que se usan en la lengua (StopWords), como lo es la palabra "the", "to" "for" y varias otras.

De la siguiente manera:

```
    tokenizer = Tokenizer(inputCol="text", outputCol="terms")
hashingTF = HashingTF(inputCol="filtered terms", outputCol="rawFeatures", numFeatures=20)

    wordsData = tokenizer.transform(docDataFrame)
featurizedData = hashingTF.transform(filteredData)
```

IV. Análisis de resultados

La comparación, se realizo con el desarrollo realizado para HPC y Spark; dado que nuestro desarrollo en HPC tiene un mal manejo de los hilos de ejecución creados, este, toma un tiempo bastante considerable para terminar la ejecución:

En este gráfico se puede observar como el desarrollo en HPC toma 0 minutos cuando tiene un nodo, esto es porque con un solo nodo la ejecución del programa falla y nunca terminara, y con el mínimo de 2 nodos podemos ver que toma un tiempo cercano a 37 minutos para analizar los 362 libros, esto, gracias al overheat generado al manejar manualmente el paso de mensajes entre procesadores mientras que en el desarrollo

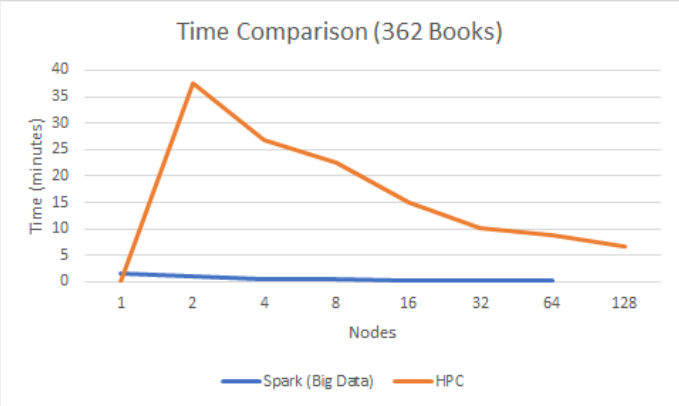

Fig. 1. Tiempo vs No. de Cores

de Spark, Apache Spark presenta librerías ya optimizadas para clustering y se auto gestiona para la comunicación entre los hilos, lo cual no hace increíblemente eficiente en comparación con nuestro desarrollo en HPC.

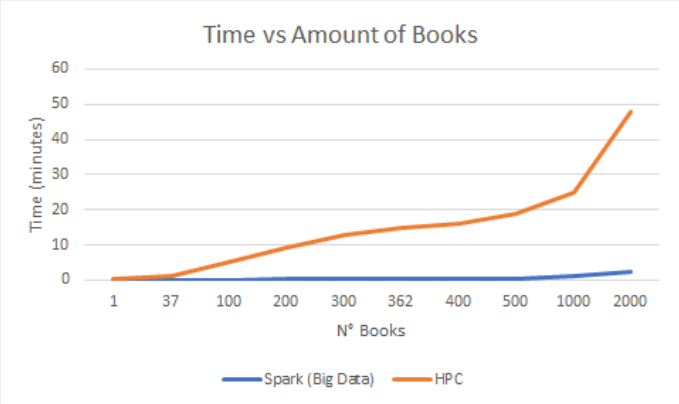

Fig. 2. Tiempo vs No. de Books

Esta gráfica es un supuesto realizado para los resultados que arrojarían los 2 desarrollos con 32 nodos disponibles. En esta gráfica podemos observar igualmente la gran diferencia en tiempos entre el desarrollo realizado en HPC y el desarrollo en spark, cada vez que la cantidad de libros se duplica, el tiempo tomado para spark no aumenta en la mayoría de veces a mas del doble anterior, mientras que en el desarrollo de HPC gracias al la baja optimización en el trabajo con los hilos, el tiempo tomado al duplicar la cantidad de libros puede llegar a hacer hasta el triple del tiempo anterior.

V. Conclusiones

- Gracias a la los algoritmos nativos de spark para el procesamiento de texto y la optimización para cluster el análisis de los datos es muchísimo mas eficiente que el desarrollo realizado previamente para HPC, dado que el manejo de los procesos fue mas manual, por lo que el tiempo de procesamiento se vio muy afectado.

- Apache spark presenta una extensa colección de librerias e implementaciones de algoritmos de análisis de datos

y data minning, lo que permite realizar el código de un programa en muy poco tiempo en comparación con el tiempo tomado de desarrollo para el mismo proyecto en HPC.

- Hemos notado que gracias a la eficiencia de las librerías que brinda spark para el análisis de datos, la lectura de textos con un solo nodo ha sido mucho mas eficiente que la lectura de la misma cantidad de documentos con mas nodos en el desarrollo de HPC

REFERENCES

[1] Bistaumanga. (01 de 09 de 2017). kMeans.py. Recuperado el 18 de 10 de 2017 de: https://gist.github.com/bistaumanga/6023692.

[2] J. Pineda. (28 de 09 de 2017). Tópicos Especiales en Telemática: Proyecto 3 – Clústering de Documentos a partir de Métricas de Similitud. Recuperado el 30 de septiembre de 2017, pp. 1-4.