Clustering de Documentos Usando Spark, TF-IDF y K-Means

Daniel A. Martínez Montealegre
dmarti25@eafit.edu.co
Universidad EAFIT

David Medina Ospina
dmedina7@eafit.edu.co
Universidad EAFIT

ABSTRACT

Este documento desarrolla el reporte técnico de la implementación de un algoritmo para un cluster Spark que permite agrupar un conjunto de documentos utilizando el algoritmo K-Means y métricas de peso de importancia de palabras mediante TF-IDF como métrica de similaridad entre los documentos.

1. PALABRAS CLAVE

Big Data: "Big data is a term that describes the large volume of data – both structured and unstructured – that inundates a business on a day-to-day basis. But it's not the amount of data that's important. It's what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves." [1]

Cluster: "A group of similar things or people positioned or occurring closely together" [2]

K-Means: "K-means (MacQueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more." [3]

TF-IDF: "Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus" [6]

Dataset: "A collection of related sets of information that is composed of separate elements but can be manipulated as a unit by a computer". [5]

Spark: "Apache Spark is an open source big data processing framework built around speed, ease of use, and sophisticated analytics. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project."

2. INTRODUCCIÓN

En esta práctica sintetizamos todos los aspectos técnicos y teóricos de la implementación de un algoritmo de clustering de documentos. Se explican los tipos de datos que fueron usados, la estrategia de implementación para un cluster Spark y los métodos tanto de peso de documentos como clasificación.

Asimismo, este reporte revela una comparación entre tiempos de ejecución de las dos versiones del algoritmo (uno implementado usando paralelismo para un clúster de HPC y el segundo usando Spark) con diferentes cantidades de archivos, de igual manera, el algoritmo de Spark fue probado en modo local y en un cluster Yarn.

3. MARCO TEÓRICO

La minería de texto (text mining), es una de las técnicas de análisis de textos que ha permitido implementar una serie de aplicaciones muy novedosas hoy en día. Buscadores en la web (Google, Facebook, Amazon, Spotify, Netflix, entre otros), sistemas de recomendación, procesamiento natural del lenguaje, son algunas de las aplicaciones. Las técnicas de agrupamiento de documentos (clustering) permiten relacionar un documento con otros parecidos de acuerdo a alguna métrica de similaridad. Esto es muy usado en diferentes aplicaciones como: Clasificación de nuevos documentos entrantes al dataset, búsqueda y recuperación de documentos, ya que cuando se encuentra un documento seleccionado de acuerdo al criterio de búsqueda, el contar con un grupo de documentos relacionados, permite ofrecerle al

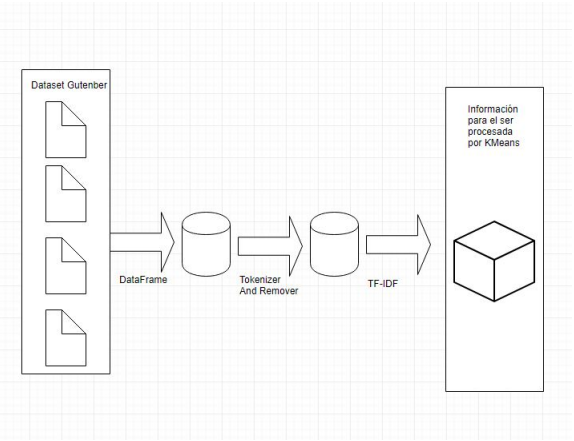
usuario otros documentos que potencialmente son de interés para él.

La idea es que usted diseñe 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.

4. ANÁLISIS Y DISEÑO

4.1 ETL

Inicialmente, para la parte de limpiar los archivos de stopwords y para hacer el stemm de los documentos , los datos son divididos equitativamente en la cantidad de cores a evaluar, por ejemplo, si se tuvieran 10 documentos y 5 cores para procesar, la división de los datos estaría representada por la imagen a continuación.



4.2 Procesamiento

Para el procesamiento de la aplicación de usó la librería de Spark para Machine learning(ML). Primero, se transforman los datos del dataset de gutenberg en data frames.

```
val files = sc.wholeTextFiles("hdfs:///user/
    ↪ dmedina7/pruebas/*.txt").map(data =>
    ↪ Row(data._1, data._2))
val dataFrame = spark.createDataFrame(files,
    ↪ struct)
dataFrame.show()
```

Posteriormente, se tokenizan las palabras.

```
val tokenizer = new Tokenizer()
    .setInputCol("text")
    .setOutputCol("words")
```

Después se remueven las stopwords.

```
val remover = new StopWordsRemover()
    .setInputCol(tokenizer.getOutputCol)
    .setOutputCol("cleanedWords")
```

Como último paso de preparación, se calcula el peso de cada palabra con relación al resto usando TF-IDF.

```
val hashingTF = new HashingTF()
    .setNumFeatures(1000)
    .setInputCol(remover.getOutputCol)
    .setOutputCol("features")
val idf = new IDF()
    .setInputCol(hashingTF.getOutputCol)
    .setOutputCol("newFeatures")
    .setMinDocFreq(3)
```

Finalmente, se ejecuta el K-Means en base a esos datos.

```
val kmeans = new KMeans()
    .setK(4)
```

Todos esto usando Pipelines para seguir una secuencia de procesamiento y darle un orden específico a la ejecución de cada transformación a hacer. Posterior a este proceso se realiza el entrenamiento y se obtiene el resultado.

```
val pipeline = new Pipeline()
    .setStages(Array(tokenizer, remover,
        ↪ hashingTF, idf, kmeans)
```

5. IMPLEMENTACIÓN EN UN CLUSTER HADOOP/SPARK EN PRODUCCIÓN.

Para la implementación se inicia configurando el proyecto, en nuestro caso al utilizar Scala realizamos este proceso mediante SBT, un manejador de proyectos Open Source para este lenguaje.

Una vez configurado el proyecto, procedimos con la implementación. Inicialmente, tuvimos un enfoque en el cual se usaban vectores de RDD como estructura para almacenar los resultados de procesamiento de cada documento. En este estado, no realizamos ninguna limpieza de los documentos, como de stopwords ni procesos de stemming, debido a que el algoritmo TFIDF se encargada de darle menor peso a las palabras que más aparecen en todos los documentos. Para esta implementación nos pareció que el código era un poco inentendible de por sí, además, realizando pruebas para analizar el tiempo de ejecución, en pro de realizar comparaciones con la implementación en HPC, notamos que los tiempos de ejecución era notablemente mucho más lentos. En este punto, decidimos re escribir el algoritmo, usando Pipelines y la implementación completa del algoritmo, con limpieza de stopwords incluída.

Para esta implementación, utilizamos una técnica que vimos en ejemplos en internet, la cual realiza un pipeline a través de diferentes etapas del procesamiento, las cuales son crear la estructura inicial de filepath y texto, tokenizar el texto, limpiar de stopwords el texto, realizar los pesos con TFIDF y posteriormente realizar el KMeans. Con esta implementación, notamos un incremento en los tiempos de ejecución.

6. ANÁLISIS DE RESULTADOS (HPC VS BIG DATA)

# Archivos	Tiempo Paralelo 16 core(s)	Tiempo Spark
50	15.638	100
100	23.047	161
200	64.755	222
500	123.243	695
1000	365.903	1296
2000	1265.146	2698

Para la primera prueba, evaluamos las dos versiones del algoritmo: el serial y el paralelo (con 16 cores), con diferentes cantidades de archivos. Como se evidencia en la tabla 1, los tiempos de ejecución del programa paralelo son menores que los del programa serial, evidenciando una ganancia en rendimiento. Se puede observar que en la últimas pruebas (1000, 2000 y 3037 documentos) el tiempo de ejecución paralela es casi 10 (aproximadamente 9.833) veces menor con respecto al serial

7. CONCLUSIONES

- Las tecnologías de Big Data de la actualidad facilitan la aplicación de algoritmos para problemas como el manejo de grandes cantidades de datos y la analítica de estos.
- Nuestra implementación de HPC superó en rendimiento a la de Big Data, aún así, esta última lo hizo mejor que la versión serial. El posible que con una mayor cantidad de documentos se observe un comportamiento diferente.
- Las librerías que provee Spark, como lo es ML, hacen el proceso de escribir la solución mucho más sencillo y simple, lo que influyó en la longitud del código.

8. REFERENCES

[1] What is Big Data?
`https://www.sas.com/en_us/insights/big-data/what-is-big-data.html`

[2] Cluster in Oxford Dictionary
`https://en.oxforddictionaries.com/definition/data_set`

[3] K-Means Algorithm
`https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html`

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

[5] Dataset in Oxford Dictionary
`https://en.oxforddictionaries.com/definition/data_set`

[6] TF-IDF
`http://www.tfidf.com/`

[7] Big Data Processing with Apache Spark
`https://www.infoq.com/articles/apache-spark-introduction`