Big Data
Clústering de Documentos a partir de Métricas de Similitud Basado en Big Data

Hincapié Zapata. Mateo, Restrepo Aristizabal. Daniel, Sierra Gallego. Marcos David

Clústering de Documentos a partir de Métricas de Similitud basado en Big Data
Departamento de Ingeniería de Sistemas
Universidad EAFIT

Resumen

En este informe se presenta el análisis y los resultados de la implementación y ejecución de un programa basado en Big Data con tecnología en Hadoop y Spark, el cual sirve para agrupar (Clustering) un conjunto de documentos utilizando los algoritmos TFIDF como técnica de similitud entre documentos y K-means para agrupar los documentos que sean mas parecidos debido a una serie de palabras relevantes en diferentes clusters; Posteriormente se comparan los resultados obtenidos con los resultados de una algoritmo basado en HPC utilizando programación en paralelo.

1. Palabras Clave

- Dataset: Conjunto de datos que reside en memoria.
- Y otras que serán definidas posteriormente tales como:
  - Hashing TF
  - KMeans
  - Text Mining
  - Data Mining
  - TFIDF

2. Introducción

Este es un proyecto desarrollado por estudiantes de la universidad EAFIT, el cual surge como practica de la materia Tópicos Especiales en Telemática en su módulo Big Data, el cual pretende aplicar las tecnologías y modelos de programación en Big Data, además de usar la plataforma Apache Spark y las bibliotecas que esta contiene. La idea también es analizar los resultados entre un acercamiento paralelo y las tecnologías y modelos en Big Data y entender los dos ambientes de supercomputación, las limitaciones, software y hardware asociadas a distintos problemas computacionales que podrían sortearse a partir de herramientas y estrategias como computación paralela y Big Data.
Este documento contiene un resumen de lo que se hizo, además muestra el diseño de la solución pensada, estructuras de datos y los resultados de la solución planteada por el equipo.
El propósito de este texto es mostrar todo lo relacionado con la actividad, teniendo en cuenta cada uno de los elementos que fueron necesarios para el desarrollo de la practica.

3. Marco Teórico

Hay una rama de la lingüística computacional que trata de obtener información y conocimiento basándose en conjuntos de datos que en principio no tienen un orden, para poder entender qué es Text Mining primero debemos comprender el concepto del Data Mining que surgió hace más de 5 años para ayudar a la comprensión de los archivos que hacen parte de las bases de datos. Para el Data Mining los datos son la materia prima bruta a los que los usuarios dan un significado convirtiéndolos en información que posteriormente será tratada y analizada por especialistas para que ésta se convierta en conocimiento. El data mining ha conseguido reunir las ventajas de áreas como la Estadística, la Inteligencia Artificial, la Computación Gráfica, las Bases de Datos y el Procesamiento Masivo, las bases de datos como materia prima. Ya luego de haber hecho una idea de lo que es Data Mining podemos entrar al tema del Text Mining que en vez de procesar datos que se encuentras en base de datos, busca procesar toda clase de textos. ésta técnica no debe confundir con la búsqueda o recuperación de datos ya que no es sólo eso ya que estas se hacen usando indexaciones de textos, clasificación, categorización, etc. La información que le interesa al Text Mining es la que está contenida de forma general en los textos y no en un sólo sino en todos los que se quieran analizar. El Text Mining comprende tres actividades fundamentales: *Recuperación de información, es decir, seleccionar los textos pertinentes. *Extracción de la información incluida en esos textos: hechos, acontecimientos, datos clave, relaciones entre ellos, etc. *Por último se realizaría lo que antes definimos como minería de datos para encontrar asociaciones entre esos datos claves previamente extraídos de entre los textos

3.1. ¿Para qué sirve el Text Mining?

Es muy útil para todas la compañías, administraciones y organizaciones en general que por las características propias de su funcionamiento, composición y actividades generan gran cantidad de documentos y que están interesadas en obtener información a partir de todo ese volumen de datos. Les puede servir para conocer mejor a sus clientes, cuáles son sus hábitos, preferencias, etc.

3.2. ¿Cómo hacer Text Mining?

Aunque es una técnica relativamente nueva y no hay una serie de pasos específicos para seguir, sí existen 4 etapas bases que se pueden usar.

- Etapa 1: Determinación de los objetivos. Aclarar que es lo que se está buscando con esta investigación, acotando hasta qué punto se quiere profundizar en la misma y definiendo claramente los límites.
- Etapa 2: Preprocesamiento de los datos, que sería la selección, análisis y reducción de los textos o documentos de los que se extraerá la información. Esta etapa consume la mayor parte del tiempo.
- Etapa 3: Determinación del modelo. Según los objetivos planteados y la tarea que debe llevarse a cabo, pueden utilizarse unas técnicas u otras.
- Etapa 4: Análisis de los resultados. A partir de los datos extraídos se tratara de ver su coherencia y se buscarán evidencias, similitudes, excepciones, etc, que puedan servir al especialista o al usuario que haya encargado el estudio para extraer conclusiones que pueda utilizar para mejorar algún aspecto de su empresa, compañía, administración u organización en general.

4. Análisis y Diseño de algoritmos

Ya que teníamos conocimiento previo sobre como empezar a desarrollar el programa para darle solución al problema, sabíamos que primero necesitábamos usar una técnica para encontrar la similitud entre los documentos, para esto hay que vectorizar las palabras, usamos la función Hashing TF, después para hallar la similitud entre los documentos usamos TFIDF, posteriormente

el resultado de este se lo pasamos a la función de k-means para así organizar todos los documentos en sus respectivos clusters dependiendo de su similaridad.

4.1. Hashing TF:

En machine learning, la característica hashing, también conocida como el truco hash (por analogía con el truco del kernel), es una forma rápida y eficiente de vectorizar características, es decir, convertir características arbitrarias en índices en un vector o matriz. Funciona aplicando una función hash a las características y utilizando sus valores hash como índices directamente, en lugar de buscar los índices en una matriz asociativa.

4.2. TFIDF:

En la recuperación de información, tf-idf o TFIDF, abreviatura de term frequency–inverse document frequency, es una estadística numérica que pretende reflejar qué tan importante es una palabra para un documento en una colección o corpus. A menudo se utiliza como un factor de ponderación en las búsquedas de recuperación de información, extracción de texto y modelado de usuarios. El valor de tf-idf aumenta proporcionalmente al número de veces que aparece una palabra en el documento, pero a menudo se compensa con la frecuencia de la palabra en el corpus, lo que ayuda a ajustar el hecho de que algunas palabras aparecen con más frecuencia en general. Hoy en día, tf-idf es uno de los esquemas de ponderación de términos más populares; El 83 % de los sistemas de recomendación basados en texto en el dominio de las bibliotecas digitales usan tf-idf.

4.3. K-means:

Kmeans es un metodo de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observacion pertenece al grupo cuyo valor medio es mas cercano. Es un método utilizado en minería de datos. La complejidad de este algoritmo es de O(n dk+1) siendo d el numero de dimensiones, k el número de clústers y n la cantidad de documentos a ser agrupados.

Las estructuras de datos utilizadas fueron las siguientes:

- RDD: Es una estructura de datos inmutable y distribuida que esta compuesta de objetos, pertenece a spark; todas las funciones que utilizamos de hashing tf retornaban objetos RDD, para mostrar la salida se hizo por medio de un RDD enviado a HDFS.
- Listas: Se usaron para Almacenar el nombre de los documentos a analizar, y los cluster a los que pertenecían.
- Diccionarios: Se usaron para guardar los archivos según su cluster.

5. Implementación:

Para la implementación se utilizó python como lenguaje de programación usando la versión 3.6.2 de su intérprete.
Utilizamos Apache Spark 2.1.1.2.6.1.0-129 para el procesamiento de los datos.
Hay que tener el cuenta que el programa HPC también tiene python como lenguaje.

Además para la implementación se utilizaron las bibliotecas de HashingTF e IDF que proporciona pyspark para poder con esto hallar la similitud entre los documentos y posteriormente con el uso de la biblioteca k-means se pudieron agrupar los documentos, después de esto se creó un diccionario en el cuál las claves son los cluster y el valor es una lista con los documentos que le pertenecen a cada uno y por último este diccionario se convirtió en RDD para poder guardarlo en hdfs y poder ver su salida después de ser ejecutado en el cluster de Big Data proporcionado por el profesor de la materia.

6. Análisis de solución:

Para probar que tanto cambiaba el progama entre HPC y basado en Big data se hicieron pruebas con el dataset cambiando el numero de archivos (6, 12, 24, 80, 89, 126, 150, 461) de las cuáles se dieron los siguientes resultados respectivamente:

6.1. 6 documentos:

- BigData: 42 segundos
- Paralelo: 1.12 segundos

6.2. 12 documentos:

- BigData: 48 segundos
- Paralelo: 1.19 segundos

6.3. 24 documentos:

- BigData: 50 segundos
- Paralelo: 2.55 segundos

6.4. 80 documentos:

- BigData: 57 segundos
- Paralelo: 3.46 segundos

6.5. 89 documentos:

- BigData: 61 segundos
- Paralelo: 4.11 segundos

6.6. 126 documentos:

- BigData: 60 segundos
- Paralelo: 6.84 segundos

6.7. 150 documentos:

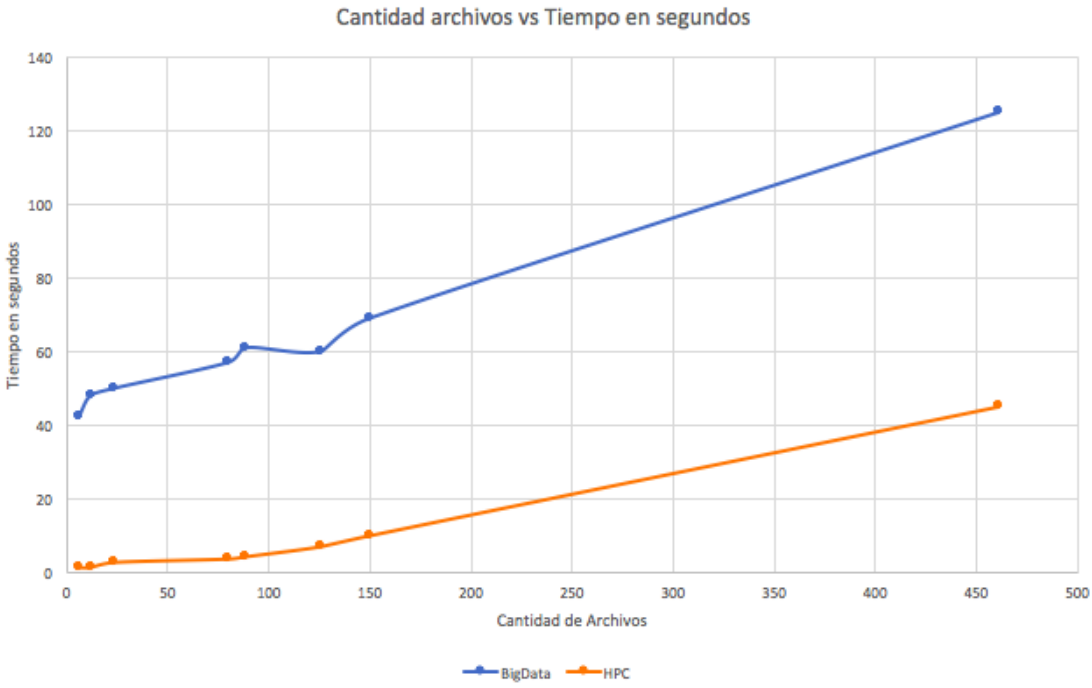
- BigData: 69 segundos
- Paralelo: 9.81 segundos

6.8. 461 documentos:

- BigData: 125 segundos
- Paralelo: 45 segundos

6.9. Gráfica

6.9.1. Análisis de la gráfica



Según la pendiente de estas dos líneas en esta gráfica (3.2884907 y 2.06117) BigData (línea azul) y HPC (línea naranja) respectivamente, podemos observar que el programa basado en Big Data se ejecuta más lento que el de HPC pero además se ve que este se vuelve mucho más lento que el de HPC a medida que se aumenta la cantidad de documentos.

7. Conclusiones:

- Se ha presentado la solución de el problema de la similitud entre archivos con la implementación de un programa basado en big data, esto nos permitió hacer una comparación entre este programa y otro en HPC; teniendo esta comparación observamos que el de HPC es mas rápido, esto puede ser debido a que en HPC tiene más máquina, tiene 50 cores y más Ram también influye también que en el de HPC nos quedamos solo con las 10 palabras mas repetidas por documento quitando las ”stopwords” (Palabras que no le dan sentido al documento y por la tanto no sirven para comparar) mientras en el de Big Data trabajamos con todas las palabras de cada documento aumentando así el numero de comparaciones.
- El uso del algoritmo para clústering KMeans funcionaría mejor si uno supiera qué cantidad de K’s debe colocar pero así se volvería un problema np completo.

8. Referencias:

1) CLUSTERING - RDD-BASED API - SPARK 2.2.0 DOCUMENTATION En el texto: (Spark.apache.org, 2017) Bibliografía: Spark.apache.org. (2017). Clustering - RDD-based API - Spark 2.2.0 Documentation. [online] Available at: https://spark.apache.org/docs/2.2.0/mllib-clustering.html [Accessed 18 Nov. 2017].
2) K-MEANS En el texto: (Es.wikipedia.org, 2017) Bibliografía: Es.wikipedia.org. (2017). K-means. [online] Available at: https://es.wikipedia.org/wiki/K-means [Accessed 18 Nov. 2017].
3) KMEANS (SPARK 1.0.1 JAVADOC) En el texto: (Spark.apache.org, 2017) Bibliografía: Spark.apache.org. (2017). KMeans (Spark 1.0.1 JavaDoc). [online]

Available at: https://spark.apache.org/docs/1.0.1/api/java/org/apache/spark/mllib/clustering/KMeans.html [Accessed 18 Nov. 2017].
4) FEATURE EXTRACTION AND TRANSFORMATION - RDD-BASED API - SPARK 2.2.0 DOCUMENTATION En el texto: (Spark.apache.org, 2017) Bibliografía: Spark.apache.org. (2017). Feature Extraction and Transformation - RDD-based API - Spark 2.2.0 Documentation. [online] Available at: https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html [Accessed 18 Nov. 2017].
5) RDD (SPARK 1.1.1 JAVADOC) En el texto: (Spark.apache.org, 2017) Bibliografía: Spark.apache.org. (2017). RDD (Spark 1.1.1 JavaDoc). [online]
Available at: https://spark.apache.org/docs/1.1.1/api/java/org/apache/spark/rdd/RDD.html [Accessed 18 Nov. 2017].
6) RDD (SPARK 2.0.1 JAVADOC) En el texto: (Spark.apache.org, 2017) Bibliografía: Spark.apache.org. (2017). RDD (Spark 2.0.1 JavaDoc). [online]
Available at: https://spark.apache.org/docs/2.0.1/api/java/org/apache/spark/rdd/RDD.html [Accessed 18 Nov. 2017].