HPC
Clústering de Documentos a partir de Métricas de Similitud

Hincapié Zapata. Mateo, Sierra Gallego. Marcos David

Clústering de Documentos a partir de Métricas de Similitud
Departamento de Ingeniería de Sistemas
Universidad EAFIT

Resumen

En este informe se presenta el análisis y los resultados de la implementación y ejecución de dos programas (uno serial y otro paralelo) que sirven para encontrar la similitud que existe entre documentos, utilizando como algoritmos principales, Jaccard y Kmeans (que serán explicados posteriormente), con los cuales agrupamos en diferentes clusters los documentos que más se parecen debido a una serie de palabras relevantes.

1. Palabras Clave

- Stopwords: Es una la lista que contiene una serie de palabras que deben ser eliminadas de cada documento leído ya que pueden ser redundantes y sin importancia por ser tan comunes en un texto.
- MPI: Es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.
- Dataset: Conjunto de datos que reside en memoria.
- Y otras que serán definidas posteriormente tales como:
  - Jaccard
  - KMeans
  - Text Mining
  - Data Mining

2. Introducción

Este es un proyecto desarrollado por estudiantes de la universidad EAFIT, el cual surge como práctica de la materia Tópicos Especiales en Telemática en su módulo HPC (high performance computing), el cual pretende exponer cómo la computación de alto rendimiento en la mayoría de los casos mejora los tiempos de ejecución de los algoritmos. Este documento contiene un resumen en el cual se da una breve síntesis de lo que se hizo, además se 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 práctica. Finalmente, se tuvo en cuenta ciertas recomendaciones hechas por parte de otros compañeros que también

desarrollaron la práctica y del docente que llevó de la mano este proyecto.

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

Para empezar con los programas se decidió gracias a la lectura de varios papers, escoger una cantidad 10 términos de cada documento para poder hacer la comparación entre ellos, entonces

lo que se hace es que después de eliminar las 'stopwords' se buscan las 10 palabras más usadas en los documentos, estas palabras se ingresan en una lista que posteriormente será utilizada por el algoritmo para encontrar la similitud de los documentos, cuando obtenemos esta similitud también obtenemos la distancia entre archivos y esto lo ingresamos en una matriz que será procesada por el algoritmo de agrupamiento donde finalmente asignamos a cada documento al grupo a que pertenecerá. Los algoritmos que se utilizaron fueron Jaccard para obtener las distancias entre los documentos y Kmeans para agrupar los documentos en sus respectivos centros.

4.1. Jaccard:

El índice de Jaccard (IJ) o coeficiente de Jaccard (CJ) mide el grado de similitud entre dos conjuntos, sea cual sea el tipo de elementos.La complejidad de este algoritmo es O(n^2) debido a que para cada documento se busca la distancia con todos los demás. La formulación es la siguiente:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

Su implementación en python es la siguiente:

```
def jaccard_similarity(x, y):
    intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
    # print(intersection_cardinality)
    union_cardinality = len(set.union(*[set(x), set(y)]))
    # print(list(set.union(*[set(x), set(y)])))
    return intersection_cardinality / float(union_cardinality)
```

Éste algoritmo se obtuvo de la siguiente página donde se explicaba de muy buena manera su funcionamiento, http://dataconomy.com/2015/04/implementing-the-five-most-popular-similarity-measures-in-python/

4.2. Kmeans:

Kmeans es un método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Es un método utilizado en minería de datos. La complejidad de este algoritmo es de O(n^{dk+1}) siendo d el número de dimensiones, k el número de clústers y n la cantidad de documentos a ser agrupados El código para el Kmeans se obtuvo de varios links y repositorios de dónde se trató de entender su funcionamiento y gracias a los cuáles se pudo realizar el KMeans que se encuentra en los programas realizados.

```
def kMeans(X, K, maxIters=10):
    centroidesI = X[np.random.choice(np.arange(len(X)), K), :]
    #Aquí es donde se asigna cada archivo que se está analizando a su cluster.
    C = []
    for i in range(maxIters):
        argminList = []
        for x_i in X:
            dotList = []
            for y_k in centroidesI:
                dotList.append(np.dot(x_i - y_k, x_i - y_k))
            argminList.append(np.argmin(dotList))
        C = np.array(argminList)
        centroidesTemp = []
        #Aquí es dónde se recalcula el centro.
        for k in range(K):
            tfArr = C == k
            nCentKArr = X[tfArr]
            promedioArr = nCentKArr.mean(axis=0)
            centroidesTemp.append(promedioArr)
        centroidesI = centroidesTemp
    return np.array(centroidesI), C
```

5. Implementación:

Para ambas implementaciones, tanto serial como paralela se utilizó python como lenguaje de programación usando la versión 2.7 de su intérprete.
En ambas implementaciones se utilizaron como principales estructuras de datos:

- Diccionarios: Sus principales funciones dentro de ambas implementaciones fueron:
  - Contar la ocurrencia de las palabras de cada texto.
  - Contar cuantas veces estaban las palabras más relevantes en cada texto.
- Listas: Sus principales usos fueron:
  - Almacenar las 10 palabras más relevantes por archivo.
  - Almacenar las 10 palabras más relevantes en todos los archivos.
  - Almacenar los centroides.
  - Almacenar a que centroide pertenece cada documento
- Matrices: Su principal función fue almacenar las distancias entre documentos.

La diferencia más notable entre las dos implementaciones fue el uso de la biblioteca MPI para python, la cual permitió en la implementación paralela llevar a cabo el paso de información desde un nodo(procesador) master a el resto de procesadores para que así cada procesador tuviera la información y datos necesarios para trabajar, además nos permitió realizar la recolección de los trabajos realizados por cada procesador para que fueran analizados por el master y de esa forma tener siempre control sobre lo que está haciendo cada procesador durante la ejecución del programa.

6. Análisis de solución:

Para probar que tanto cambiaba y mejoraba el código de Serial a Paralelo se hicieron pruebas con datasets diferentes (20, 50, 100, 200, 300, 400, 600) de las cuáles se dieron los siguientes

resultados respectivamente:

6.1. 20 documentos:

- Serial: 18.8 segundos
- Paralelo: 7.43 segundos

6.2. 50 documentos:

- Serial: 55 segundos
- Paralelo: 20.55 segundos

6.3. 100 documentos:

- Serial: 104.28 segundos
- Paralelo: 65.19 segundos

6.4. 200 documentos:

- Serial: 415.52 segundos
- Paralelo: 212.22 segundos

6.5. 300 documentos:

- Serial: 649.42 segundos
- Paralelo: 365.06 segundos

6.6. 400 documentos:

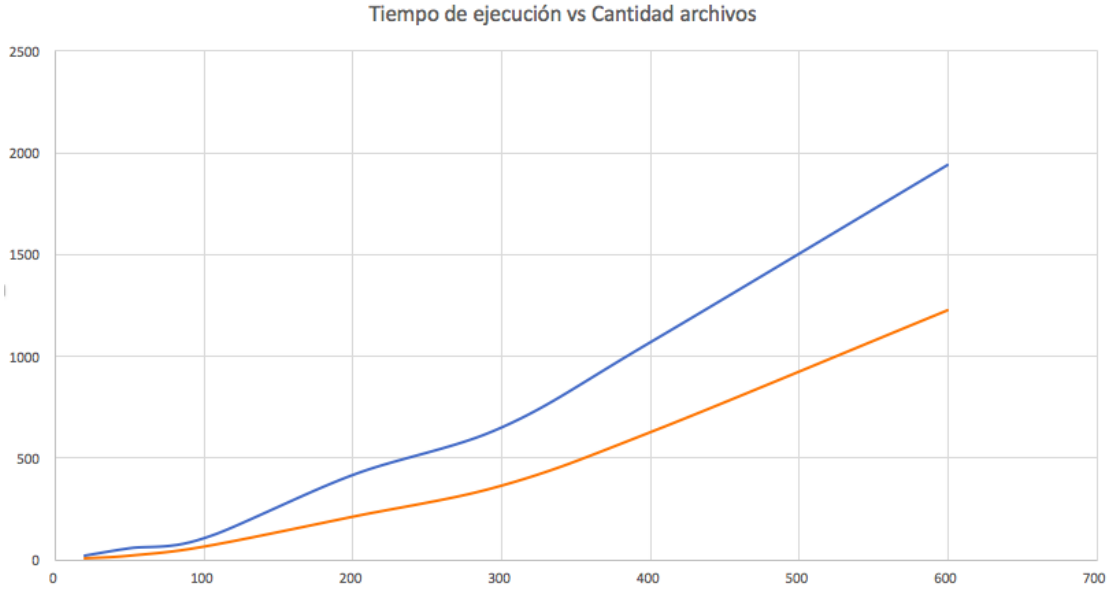
- Serial: 1068.76 segundos
- Paralelo: 627.98 segundos

6.7. 600 documentos:

- Serial: 1943.11 segundos
- Paralelo: 1229.1 segundos

6.8. Gráfica

6.8.1. Análisis de la gráfica

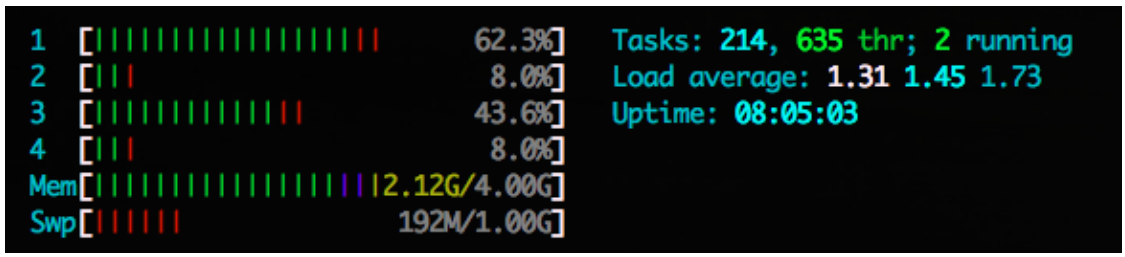


Según la pendiente de estas dos líneas en esta gráfica (3.2884907 y 2.06117) Serial (línea azul) y

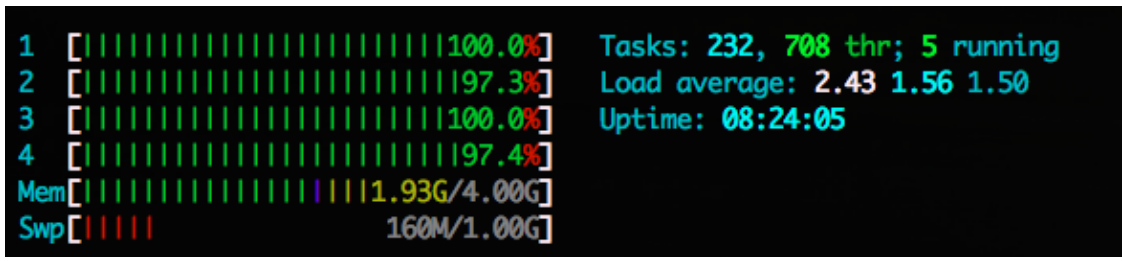
Paralelo (línea naranja) respectivamente, podemos observar que el programa en serial se ejecuta más lento que el paralelo pero además se ve que este se vuelve mucho más lento que el paralelo a medida que se aumenta la cantidad de documentos. Esta disminución de tiempo de ejecución en el programa en paralelo se debe a que se hizo una descomposición por dominio del problema, lo que quiere decir que a cada procesador le aplicó la misma operación a diferentes conjuntos de datos, esto se pudo hacer debido a que los algoritmos de Jaccard y Kmeans además de la obtención de los archivos tienen una estructura muy independiente, lo que facilitó la paralelización de los mismos.

7. Conclusiones:

- Se ha presentado la solución de el problema de la similitud entre dos archivos con la implementación de dos algoritmos (Serial y Paralelo), esto nos permite decir que con el uso de la programación en paralelo este problema tiene un buen comportamiento porque su tiempo de ejecución siempre es menor que con el paradigma de programación serial.
- El uso del algortimo para clústering KMeans funcionaría mejor si uno supiera qué cantidad de K's debe colocar pero así se volvería un problema np completo.
- El siguiente era el comportamiento habitual del programa corriendo en serial.



- El siguiente era el comportamiento habitual del programa corriendo en paralelo.



- La elección del Jaccard se vio basada en la lectura del documento 'Similarity measures for text document clustering', ya que ahí nos encontramos con que éste era uno de los más puros y uno de lo que tenía más entropía, además de su facilidad de uso.

8. Referencias:

1) Es.wikipedia.org. (2017). K-means. [online] Available at: https://es.wikipedia.org/wiki/K-means [Accessed 21 Oct. 2017].
2) Es.wikipedia.org. (2017). indice Jaccard. [online] Available at: https://es.wikipedia.org/wiki/
3) Google Docs. (2017). Gutenberg.zip. [online] Available at: https://goo.gl/LL4CgA [Accessed 21 Oct. 2017].
4) Mouwerik, J., Mouwerik, J., Mizerski, N. and Mouwerik, J. (2017). Implementing the Five Most Popular Similarity Measures in Python - Dataconomy. [online] Dataconomy. Available at: http://dataconomy.com/2015/04/implementing-the-five-most-popular-similarity-measures-

in-python/ [Accessed 21 Oct. 2017].
5) Textmining.galeon.com. (2017). Text mining Mineria de Textos Data Mining Mineria de datos recuperacion y organizacion de la informacion. [online] Available at: http://textmining.galeon.com/ [Accessed 21 Oct. 2017].
6) Anna Huang. Similarity measures for text document clustering. Proceedings of the Sixth New Zealand, (April):4956, 2008.
7) Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.