Clústering de documentos a partir de métricas de similitud

Daniel Hoyos-Ospina
Escuela de ingeniería
Pregrado en ing. de sistemas
Universidad EAFIT
Medellín, Colombia
Email: dhoyoso@eafit.edu.co

Daniela Serna-Escobar
Escuela de ingeniería
Pregrado en ing. de sistemas
Universidad EAFIT
Medellín, Colombia
Email: dsernae@eafit.edu.co

***Abstract*—Este documento plantea una solución al problema de agrupamiento (clústering) de un conjunto de documentos de texto no ordenados, utilizando el algoritmo k-means y la función de similitud Jaccard para hallar la distancia entre ellos y crear clústers significativos y coherentes, desarrollada e implementada en Python de forma paralela y serial para la reducción de tiempos de ejecución.**

Palabras clave:

Clústering, K-means, similitud, programación paralela, Jaccard, MPI.

1. Introducción

La agrupación de documentos es una técnica útil usada para grandes volúmenes de documentos de texto que están en internet como bibliotecas digitales, repositorios, artículos de blogs, correos electrónicos, buscadores de web, sistemas de recomendación que hacen parte de los problemas asociados a computación de alto rendimiento y presentan un desafío para grandes compañías (Google, Facebook, Amazon, Netflix, Spotify, entre otros) pues la información debe ser organizada de forma correcta y automática en colecciones coherentes y eficaces.

Las métricas de comparación pueden variar según su propósito, por ejemplo en artículos de investigación dos documentos se consideran similares si comparten los mismos temas, pero para motores de búsqueda se compara según el tipo de información que presenta, sin embargo, para esta solución se concibe la similitud en términos de distancia y existen diferentes funciones de similitud, entre los mas usados están la distancia Euclidiana, Coseno, Jaccard y Pearson, gracias a esta diversidad la eficacia en la agrupación de documentos sigue siendo cuestionada. Hallar el algoritmo más optimo no es el propósito de este artículo, pero se enunciará más adelante porque se implementó con Jaccard.

2. Marco Teórico (Descripción del problema)

2.1. Representación de documentos

Se tiene un conjunto de documentos de texto grande o un dataset, el cual debe presentar algunas relaciones sintácticas y semánticas entre los documentos, es decir, que hablen del mismo tema o pertenezcan al mismo género. Para esto se usó la base de textos de Gutenberg. Las palabras aparecen de forma independiente y sin importar el orden, se determina el peso de cada termino, es decir su frecuencia de aparición en los documentos, lo que significa que las palabras que aparecen con mayor frecuencia son las que mejor describen la similitud para los documentos. Pero existen palabras que aparecen en repetidas ocasiones y que no representan los textos o no son importantes, por ejemplo, conectores, artículos, pronombres, preposiciones etc. Que son filtradas antes del procesamiento de datos en lenguaje natural para garantizar que los documentos si sean comparados correctamente.

2.2. Desarrollo

Es necesario contar con una línea base para poder analizar la similitud de cada documento con los demás mediante un primer acercamiento usando un algoritmo serial, este permitirá comparar las aceleraciones obtenidas posteriormente por las estrategias de computo paralelo en tiempos de ejecución y recursos consumidos. El resultado será no solo determinar la diferencia entre los documentos sino encontrar los distintos clústeres de los subgrupos dependiendo de la similitud entre ellos. Por último, se presenta una solución que evidencie la reducción de tiempos y muestre como las estrategias, herramientas, infraestructura y tecnologías permiten resolver adecuadamente problemas.

2.3. Procesamiento

Se utiliza el paradigma de paso de mensajes MPI 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." [1] Y es empleada en programación concurrente que permite la exclusión mutua y la sincronización entre procesos con la librería MPI4PY.

[1]Definición tomada de: $https://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes$

2.4. Elección del algoritmo de similitud

Se debe determinar la medida de similitud que refleja el grado de cercanía o separación de los documentos y debe corresponder a características que se crea distingan al clúster, mediante los distintos algoritmos que no solo dan como resultado diferentes particiones finales sino impone también diferentes requisitos para el mismo agrupamiento. Se estudiaron las distintas métricas que existen para comparar los algoritmos y determinar el mejor o el más adecuado para este caso, estas son: la distancia entre dos puntos cualquiera debe ser no negativa, la distancia entre dos puntos debe ser cero si los dos objetos son idénticos, la distancia debe ser simétrica entre dos puntos, entre otras.

Las diferentes funciones de similitud tienen ventajas y desventajas que dependen de múltiples factores, así que es casi imposible determinar cuál es el mejor y que este aplique para todos los casos, por ejemplo, los algoritmos que se basan en distancias métricas (como distancia euclidiana) no son apropiados para altas dimensiones o dominios dispersos, es decir, palabras muy diferentes. Por lo tanto, como no sabemos qué tan dispersos estarán los documentos, se debe escoger un algoritmo que responda bien ante esta situación.

Por último, después de realizar una investigación sobre cual algoritmo usar, se encontró que los dos mejores (enfocados a nuestro problema) son Coseno y Jaccard. Se realizaron pruebas con ambos y en algunos casos Coseno provocaba errores por lo cual fue descartado.

El coeficiente de Jaccard mide la similitud de la intersección dividida por la unión de los objetos, 1 significa que los objetos son los mismos y 0 que son completamente diferentes.

$$\mathrm{J}(A,B) = \frac{|A \cap B|}{|A \cap B|}$$

Figure 1. $https://www.packtpub.com/graphics/9781783287017/graphics/7017OS_09_03.jpg$

2.5. Algoritmo de clústering

El algoritmo K-means trabaja con medidas de distancia que básicamente apuntan a minimizar las distancias dentro del clúster. Por lo tanto, las medidas de similitud no encajan directamente en el algoritmo, porque los valores más pequeños indican desemejanza. La distancia euclidiana y la divergencia KL promediada son medidas de distancia, mientras que la similitud del coseno, el coeficiente de Jaccard y el coeficiente de Pearson son medidas de similitud. Se tuvo que aplicar una transformación simple para convertir la medida de similitud en valores de distancia.

Dado un conjunto de objetos D y un número preestablecido de clústeres K elegidos al azar. Cada k es un centroide (centro geométrico) de un clúster y los objetos restantes se asignan al clúster representado por el centroide más cercano o más similar. A continuación, se vuelven a calcular los centroides nuevos para cada grupo y a su vez todos los nuevos documentos en base a los nuevos centroides. Además, se saca un grupo de documentos sobrantes, es decir, aquellos que no tienen relación con ningún centroide.

Las soluciones de agrupamiento generadas son óptimas para el conjunto de datos dato y las semillas iniciales, las diferentes elecciones de conjuntos y clústeres iniciales pueden dar lugar a particiones finales muy diferentes y por ende a tiempos de ejecución variantes. Sin embargo, se utilizó el algoritmo básico de K-means porque optimizar el agrupamiento no es el enfoque de este proyecto.

Los resultados son óptimos cuando se minimiza la distancia intra-clústern (cohesión) y se maximiza la distancia inter-clúster (separación), pero este algoritmo no garantiza que los centroides finales obtenidos sean los mejores, ya que en este caso solo se itera dos veces porque hacerlo más veces implicaba un costo mayor en términos de tiempos de ejecución.

3. Análisis y diseño mediante PCAM incluyendo algoritmos y estructuras de datos y rendimiento análitico de la solución.

La necesidad que surge para resolver problemas que requieren tiempo elevado de computo origina lo que hoy se conoce como computación paralela, existe una metodología de diseño que busca abarcar programas paralelos. Inicia con la especificación de un problema, desarrollando una partición, determinando los requerimientos de comunicación, las tareas aglomeradas y finalmente el mapeo de procesadores.

3.1. Particionamiento

El particionamiento se trata de decomponer los calculos en tareas más pequeñas. Este punto es independiente a la arquitectura o el modelo de programación. El

parcicionamiento puede basarse en división funcional o por dominio. En este caso, el algoritmo de K means se puede subdividir en dos grandes funciones la primera es calcular los clústers a partir de un conjunto de documentos y K centroides, la segunda es buscar un nuevo centro dentro de los clústers ya existententes, esta última es totalmente dependiente del resultado de la primera, por lo tanto, una división funcional no es posible debido a la naturaleza del problema.

Teniendo en cuenta lo anterior se decidio dividir el problema por dominio; en la primera funcion de crear los clústers se cogen el total de documentos y se subdividen en la cantidad de núcleos disponibles de manera equitativa, con estos sub dominios de documentos y con los centroides cada núcleo es capaz de calcular sus propios sub clusters de documentos los cuales serán juntados más tarde por el master en un solo conjunto de clusters.

Luego, en la segunda función de recentrado, se coje cada cluster y se dividen los documentos del mismo equitativamente para los N núcleos de manera que cada núcleo realizará la comparación de ese subdominio de documentos que le tocó contra todos los otros, saca el promedio de distancias de documentos de su subdominio con los otros y elige el mayor. Apenas todos acaban el master reunira todos los documentos con menos distancia promedio y eligina al menor de menores y este será el nuevo centroide del clúster. Esto se repite por cada clúster que se tenga.

3.2. Comunicación

Las tareas que se generán de una partición funcional o por dominio estan propuestas para ejecutarse simultaneamente pero no pueden, en general, ejecutarse independientemente. Las funciones que cada núcleo va a realizar puede que requieran datos asociados con otras tareas por lo que es necesario establecer como será la comunicación entre tareas. En el caso del K means la comúnicación entre tareas se realiza al principio y al final de cada una de las funciones descritas en el punto anterior.

En la primera función de calcular los clústers, es necesario que el master envié a todas tareas los documentos que le corresponden a cada una y los centroides que son comunes a todos, despúes de esto cada uno calcula su sub mapa de clústers y al final cuando todos lo han calculado el master junta todos en un mapa de clústers global.

En la segunda función el master debe enviar los documentos que le corresponden a cada núcleo de un cluster y envia la lista total de documentos para que cada núcleo o tarea se encarge de calcular el promedio de cada documento de su subdominio contra todos los otros y así poder saber cuál tiene menor distancia en promedio dentro de su subdominio. Luego, cuando todos acaban, el master debe juntar todos los documentos resultantes de cada núcleo y decidir cual tiene menor distancia promedio de todos estos, encontrando así el nuevo centroide de cada cluster. Esto se repite K veces hasta obtener todos los centroides y finalmente se vuelve a hacer la primera función de la misma manera para obtener el mapa de clústers final.

3.3. Aglomeración

Las tareas y estructuras de comunicación establecidas en los dos primeros puntos del diseño deben ser evaluadas basandose en los requerimientos de ejecución y costos de implementación para definir si es necesario combinar tareas en tareas más grandes o hacer subdominios de datos más grandes para cada tarea con el fin de mejorar la ejecución o reducir costos de comunicación y sincronización.

En nuestro caso las división es por dominio en las dos funciones principales y se observó que si se tienen tantas tareas como documentos los costos de comunicación para dividir y reunir los resultados de los calcúlos individuales de cada tarea son más altos que si se aglomeran conjuntos de datos más grandes para cada tarea individual. Además si solo se tiene un documento por tarea existen algunas funciones individuales como el calculo del documento de menor distancia en su sub dominio no tendría sentido y esta responsabilidad de comparar todos y saber cual es el menos recaeria totalmente en el master afectando el rendimiento. Por esto se llegó a la conclusion de que dependiendo del tamaño del dataset es mejor dividirlo en un numero de tareas que permita que cada una de ellas pueda realizar cierto numero de operaciones que en realidad contribuya al rendimiento antes de efectuar la comunicación que en comparación puede resultar más costosa.

3.4. Mapeo

El objetivo del mapeo es asignar a cada procesador tareas que intenter satisfacer las metas, maximizando el uso del procesador y disminuyendo los costos de comunicación. En este caso todos los procesadores realizarán exactamente las mismas tareas a excepción del master que es el encargado de dividir y reunir el dominio de los datos a través de la ejecución del programa por lo que en realidad lo que más ha de tenerse en cuenta en este punto es la cantidad de documentos que se tiene y la cantidad de tareas necesarias para procesar ese dataset basado en las repercuciones que esta decisión tiene en el rendimiento como se explicó en el punto anterior. Así que no importa como se dividan las tareas y en que procesadores, solo es necesario tener un numero de tareas que no haga que los sub sets de datos de cada núcleo sean muy pequeños para garantizar un buen rendimiento y un bajo costo de comunicación.

3.5. Algoritmos y estructuras de datos

El algoritmo empieza sacando los centroides del total de documentos y dividiendo los documentos en la cantidad de núcleos que se tienen; esto se hace en un arreglo de

N posiciones con N = Numero de núcleos disponibles, en donde cada posición de este arreglo tendrá otro arreglo que contiene el sub dominio de documentos que le corresponde a cada núcleo. Despúes de dividir los documentos el master envia a cada núcleo su respectivo subdominio y los centroides globales, cada núcleo se encarga de coger su subdominio y comparar cada elemento de este con todos los centroides; este resultado se almacena en un mapa en donde la clave es el nombre del documento y el valor es un arreglo de K posiciones en donde cada posición equivale a la distancia de ese documento contra el centroide 0, 1, ... y K respectivamente. Despúes de esto cada núcleo se encarga de mirar cada documento a que centroide es menos distante y lo agrega al clúster correspondiente en un mapa de clústers en donde la clave es el numero del cluster y el valor es el arreglo de documentos pertenecientes a ese clúster. Despúes de esto cada núcleo tiene su propio mapa de clústers y el master se encarga de reunirlos todos y crear un mapa de clústers global con las mismas propiedades.

Despúes de esto se procede a recentrar, aquí el master separa los documentos de un clúster en un arreglo de N posiciones, en donde cada posición de este arreglo tendrá otro arreglo que contiene el sub dominio de documentos que le corresponde a cada núcleo y se envia el subdominio a cada nucleo como se hizo en un principio; despues de esto cada núcleo se encarga de comparar su sub dominio de documentos contra todos los otros en una matriz de ixj con i = numero de documentos del sub dominio, y j = numero total de documentos, en donde cada posición tiene la distancia de los documentos comparados. Despues de calcular toda la matriz se hace un promedio de distancias de cada documento del sub dominio y se decide cual es el que tiene menor distancia promedio con todos los otros. Cuando todos los núcleos acaban, el master reune el documento resultante de cada uno y calcula el que tiene menor distancia en promedio de todos y este se convierte en el nuevo centroide de ese cluster. Estos pasos se repiten por cada clúster y al final el master obtiene un arreglo con todos los centroides que envia a todos los núcleos para realizar de nuevo el calcúlo de los clusters como se hizo en un principio dividiendo el total de documentos equitativamente en todos los núcleos disponibles.

4. Implementación

En ambos casos se toma el tiempo al iniciar y terminar la ejecución para saber exactamente cuanto tiempo requirió el algoritmo para procesar todo el dataset usado.

Lo primero que se implementó en ambos casos, serial y paralelo, fue la apertura de los archivos, de los cuales se extraen todas las palabras y se agregan a un arreglo en donde se excluyen las stop words. Después de calcular el arreglo, se agregan a un mapa en donde la clave es el nombre del documento y el valor es el arreglo de palabras del mismo así:

mapa [doc1] = [perro, perro, pavo, pavo]

Luego, en ambos casos se implementó la función de distancia entre dos documentos basándose en el algoritmo de jaccard la cual recibe el nombre de los dos documentos a comparar, los busca en el mapa anterior, saca sus arreglos, se crean dos mapas que tendrán como clave la palabra j en el documento i para i: 0..1 y j: 0.. n palabras, estos dos mapas se rellenan teniendo como valor la suma de frecuencias de aparición de la palabra j en el documento i así:

mapa [doc1] = [perro,perro,pavo,pavo]

mapa [doc2] = [gato,gato,perro,perro]

mapaDeFrecuencias = {perro : 2, pavo : 2}

mapaDeFrecuencias = {gato : 2, perro : 2}

Ya con estos mapas se calcula la intersección entre los arreglos de palabras de cada documento, se suman las frecuencias de las palabras que aparecen en ambos documentos, se suma el tamaño de la cantidad de palabras en ambos documentos y por último se divide la suma de las frecuencias por el número de palabras en ambos documentos lo cual nos da la distancia o coeficiente de similitud entre estos dos documentos así:

intersección = [perro]

suma_de_frecuencias_en_la_interseccion = 4

tamano_de_palabras_en_ambos_docs = 8

resultado = 4/8

resultado = 0.5

Después de tener las funciones de similitud y de apertura de archivos listas se procedió a implementar el K-Means en forma serial y paralela:

4.1. Serial

Lo primero que se hace es abrir todos los documentos que se tienen y sacar el mapa de arreglos como se indicó anteriormente. Luego se sacan al azar K centroides de estos documentos, se crea un set vacío que contendrá los documentos que no tienen relación con ningún centroide y se procede a llamar la función de clústering.

Esta función recibe los centroides, los documentos y el set de documentos sin relación. Lo que se hace es comparar cada documento contra los K centroides que se tienen usando la función de similitud así:

	Centroide 1	Centroide 2	...	Centroide K
Doc 1	0,2	0,8	...	1

Después se coge cada documento, se mira con cuál de los centroides se tiene mayor relación y se mete al clúster correspondiente; si el documento no tiene relación alguna con ninguno de los centroides se mete al set de documentos sin relación. Así con todos hasta obtener K clústers de documentos así:

MapaClusters [K] = [Documento 1 , ... , Documento N]

Luego, se procede a buscar un nuevo centroide para cada clúster; entonces por clúster se comparan todos los documentos, se hace un promedio de distancias de cada documento contra todos los otros del clúster y así el documento con mayor distancia promedio a todos los otros documentos del clúster será considerado el nuevo centroide.

	Doc 1	Doc 2	...	Doc N
Doc 1	1	0,8	...	0,5
Doc 2	0,8	1	...	0,2
...	...	...	...	1
Doc N	0,5	0,2	...	1

Promedios:

	Doc 1	Doc 2	...	Doc K
Promedio	0,8	0,3	...	0,4

El nuevo centroide de este cluster es = Doc 1.

Después de iterar en cada clúster se obtienen todos los nuevos centroides, se limpia el set de documentos sin relación y se procede a realizar de nuevo la función de clústering.

4.2. Paralelo

La implementación paralela usa las mismas funciones de similitud, apertura de archivos, recentrado y clústering; sin embargo, se realizaron algunos cambios lógicos para lograr una separación por dominio del problema.

En la primera iteración en donde se saca por primera vez los clústers basados en los centroides obtenidos al azar, se divide la cantidad de documentos totales en los núcleos que se tienen de manera uniforme. Así cada núcleo se encargará de comparar los documentos de su sub dominio con todos los centroides, decidir a cual clúster pertenece y agregarlo a su propio mapa de clústers y luego al final, cuando todos los núcleos han calculado el mapa de clústers correspondiente a su conjunto de documentos, el master reunirá todos los sub mapa clusters y creara un mapa clústers común así:

docs = [doc1, doc2, doc3, doc4, doc5, doc6, doc7];

Con 2 nucleos:

subdocs = [[doc1, doc2, doc3, doc7],[doc4, doc5, doc6]]

Entonces al nucleo 1 le corresponde el subdominio de la posición 0 del arreglo, al nucleo 2 la posición 1 y así sucesivamente.

Después se compara cada subdominio con los centroides y se sacan los mapas por subdominio:

SubMapaClusters[K] = [docs del clúster K del subdominio 0]

SubMapaClusters[K] = [docs del clúster K del subdominio 1]

Se juntan y se obtiene un mapa global:

MapaClusters[K] = [docs que pertenecen al clúster K]

Luego se procede a buscar el nuevo centro de cada clúster, para esto se coge cada clúster y se dividen los documentos del clúster uniformemente en cada núcleo, cada núcleo se encarga de realizar la comparación de todos los documentos respectivos y sacar el promedio de distancia de cada uno contra todos los otros y elegir el documento con menor distancia promedio. Al final cuando el máster junta todos estos y mira cual es el menor eligiendo asi el nuevo centroide del clúster. Esto se repite con cada clúster obteniendo los nuevos centroides así:
Por cada nucleo:

subDominioDelCluster = [doc1, doc2, doc3, doc4];

	Doc 1	Doc 2	Doc 3	Doc 4
Doc 1	1	0,5	0,6	0,5
Doc 2	0,5	1	0,8	0,9
Doc 3	0,6	0,8	1	0,1
Doc 4	0,5	0,9	0,1	1

Promedios:

	Doc 1	Doc 2	Doc 3	Doc 4
Promedio	0,65	0,8	0,625	0,625

Entonces el mayor promedio de este subdominio del clúster es el doc 2.

El proceso anterior se repide por cada núcleo con su subdominio y luego el máster junta todos los mayores de cada sub dominio del clúster y elige el mayor de los mayores el cual será el nuevo centroide del clúster.

Luego, al hacer todo lo anterior en cada clúster obtenemos todos los nuevos centroides y se procede a recalcular de nuevo el mapa global de clústers a partir de la división por dominio como se explicó anteriormente en este apartado de implementación paralela.

5. Análisis de resultados (serial vs. paralelo)

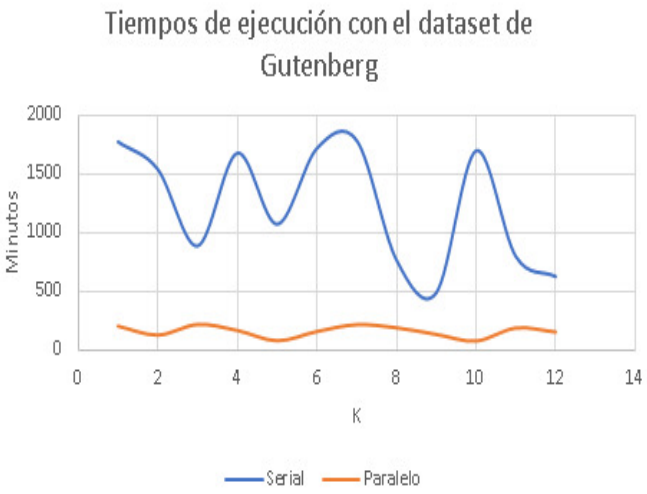
Para probar el código se usaron dos datasets; el dataset de gutenberg[1] que consta de 3036 libros en inglés y el dataset de SentenceCorpus[2] que consta aproximadamente

de unos 1000 archivos de texto relativamente cortos en comparación a Gutenberg.

Al correr las implementaciones serial y paralela en ambos datasets y variando la K se obtuvieron los siguientes resultados:

Para Gutenberg:

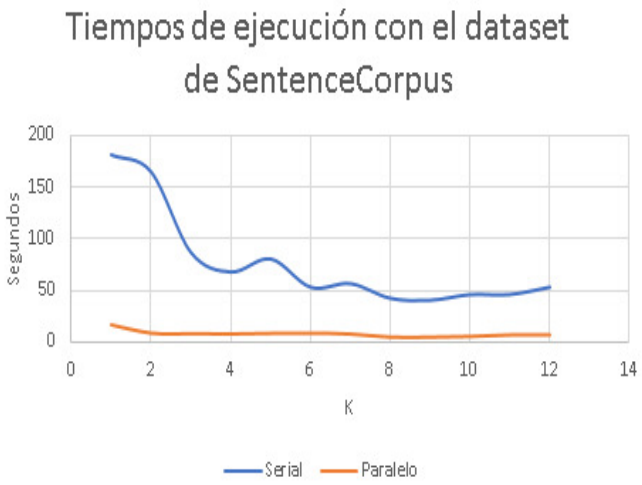
Tiempos de ejecución con el dataset de Gutenberg		
K	Tiempo Serial (mins)	Tiempo Paralelo (mins)
1	1771,57	196,60
2	1537,98	124,60
3	889,68	209,97
4	1676,40	161,48
5	1073,87	78,65
6	1716,48	153,43
7	1782,55	208,33
8	769,38	183,20
9	492,65	130,02
10	1696,20	76,82
11	799,30	181,42
12	630,22	148,73



En promedio la mejora del tiempo de ejecución paralelo fue del orden de 9,007 veces más rápido que el serial con el dataset de Gutenberg.

Para SentenceCorpus:

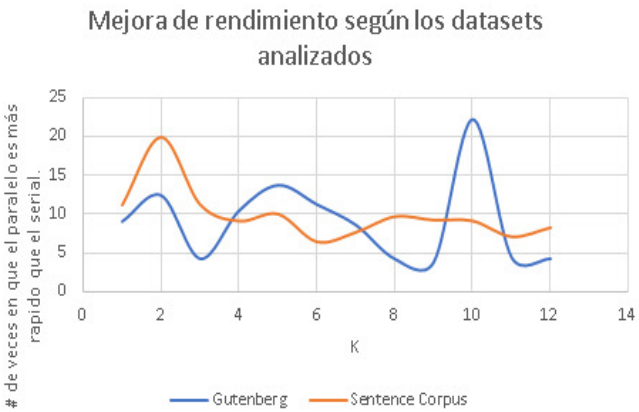
Tiempos de ejecución con el dataset de SentenceCorpus		
K	Tiempo Serial (mins)	Tiempo Paralelo (mins)
1	180,615	16,139
2	164,153	8,237
3	86,477	7,686
4	67,66	7,417
5	80,281	8,008
6	52,966	8,226
7	56,509	7,368
8	42,350	4,369
9	40,482	4,378
10	45,887	5,021
11	46,006	6,473
12	53,058	6,418



En promedio la mejora del tiempo de ejecución paralelo fue del orden de 9,923 veces más rápido que el serial con el dataset de SentenceCorpus.

En ambos casos las pruebas paralelas se realizaron usando 50 núcleos.

En la siguiente gráfica se puede observar las aceleraciones que se obtuvieron al paralelizar con los diferentes dataset.



En ambos datasets se logró evidenciar que el trabajo de paralelización rindió sus frutos, sin embargo, las aceleraciones no fueron uniformes en todas las ejecuciones, pues, a pesar de usarse el mismo K, el mismo código y la misma cantidad de núcleos, los centroides varían en cada ejecución, lo que hace que pocos o incluso ningún documento se relacione con alguno de ellos, reduciendo así el tamaño de ese clúster y la cantidad de operaciones necesarias para calcular el recentrado que es la parte del algoritmo que toma más tiempo.

[1]`Gutenberg dataset:`
$https://web.eecs.umich.edu/\sim lahiri/gutenberg\_dataset.html$

[2]`SentenceCorpus dataset:`
$http://archive.ics.uci.edu/ml/machine{-}learning{-}databases/00311/$

6. Conclusiones

La elaboración del proyecto permitió ampliar los conocimientos relacionados con clústering y paralelismo, que sin duda son muy importantes en la computación de alto rendimiento.

Se logró el objetivo del proyecto (uso de paralelismo) gracias al uso ordenado y planeado de la metodología PCAM.

El uso de la programación paralela combinada con estrategias y herramientas, son de vital importancia para resolver problemas que sin ellos no podrían ser solucionados o requerirían recursos computacionales mucho mayores.

Se logró reducir tiempos de ejecución mediante estrategias, tecnologías, infraestructura, herramientas y algoritmos paralelos que pudieran evitar las limitaciones a nivel de software y hardware en problemas computacionales.

Se implementó un algoritmo de clústering (k-means) para sortear los retos que presenta hoy la minería de texto.

Se utilizó el paradigma de paso de mensajes para hacer posible la paralelización del problema basado en la división por dominio.

Referencias

(2016). Parallel Algorithm Analysis and Desing.*Parallel and High Performance Computing.* Recuperado de $www.math{-}cs.gordon.edu/courses/cps343/presentations/Parallel_Alg_Design.pdf$

(2017). *Stop words*. Recuperado de $https://en.wikipedia.org/wiki/Stop_words$

(2017). *Interfaz de paso de mensajes*. Recuperado de $https://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes$

Fernando Berzal. (2011). *Clustering.* Tomado de $http://elvex.ugr.es/decsai/intelligent/slides/dm/D3\%20Clustering.pdf$

R.Subhashini y V.Jawahar Senthil Kuma. (2010).*Evaluating the Performance of Similarity Measures Used in Document Clustering and Information Retrieval.* Tomado de:$http://ezproxy.eafit.edu.co:2214/stamp/stamp.jsp?tp=\&arnumber=5571521\&tag=1$

Anna Huang. *Similarity Measures for Text Document Clustering.* Tomado de $http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.4480\&rep=rep1\&type=pdf$

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. (2009). *Introduction to Information Retrieval.* Tomado de $https://nlp.stanford.edu/IR{-}book/pdf/irbookonlinereading.pdf$

Chunzi Wu and Bai Wang. (2017). *Extracting Topics Based on Word2Vec and Improved Jaccard Similarity Coefficient.* Recuperado de $http://ezproxy.eafit.edu.co:2214/document/8005506/$

Anand Kumar Gupa and Neetu Sardana. (2015). *Significance of Clustering Coefficient over Jaccard.* Tomado de $http://ezproxy.eafit.edu.co:2214/xpls/icp.jsp?arnumber=7346726$

Jerry Scripps and Christian Trefftz. (2015).*Parallelizing an algorithm to find communities using the Jaccard metric.* Tomado de $http://ezproxy.eafit.edu.co:2214/xpls/icp.jsp?arnumber=7293371$

Programación Paralela. Tomado de $https://www.cs.buap.mx/\ mtovar/doc/ProgConc/ProgramacionParalela.pdf$