Reporte técnico

Juan Pablo Gaviria, Felipe Montoya, David Alejandro Gomez

June 7, 2017

1 Introducción

En el siguiente documentos se describirá la solución dada por el equipo 11 al problema de obtener los documentos mas relacionados para cada documento de un conjunto. A su vez se realizará un análisis de los datos obtenidos en la solución y una comparativa entre la solución serial y la solución paralela y el rendimiento de la solución paralela en varios nucleos.

2 Desarrollo

2.1 Algoritmo Serial:

2.1.1 Diseño:

La aplicación serial se planteó para que siguiera una serie de pasos que al final dieran por resultado una lista de los documentos más relacionados para cada documento, esta serie de pasos se construyó en base a la investigación de Anna Huang [Hua08].

1. Hacer el preprocesamiento de los documentos, esto es, limpiar las palabras de tildes y caracteres que no pertenezcan al alfabeto, signos de puntuación, stopwords que no son relevantes para describir el texto y aplicar un algoritmo para sacar las raíces de las palabras, con el fin de hacer relaciones más fuertes entre documentos.

2. Realizar la representación de los documentos como un vector que describa la frecuencia de todas las palabras en base a en cuantos documentos más aparece. Para hacer esto se debe de construir los vectores de frecuencia de términos para cada documento y un vector de todos los términos y el número de documentos en los que aparecen. La representación del documento está dada por:

   $tfidf(d,t) = tf(d,t) \times log(\frac{|D|}{df(t)})$

   Donde:

   $tfidf(d,t) =$ Es la frecuencia del termino t en el documento d, teniendo en cuenta en cuantos documentos más aparece.

   $tf(d,t) =$ Es la frecuencia del termino t en el documento d.

   $D =$ Es el vector de documentos

3. Aplicar el algoritmo de Jaccard con el fin de obtener el coeficiente de correlación de un documento con otro. Este algoritmo se aplica en una matriz en el que los elementos por encima de la diagonal serán iguales a los debajo de la diagonal y la diagonal será 1, por lo que se puede ahorrar procesamiento solo calculando los elementos encima de la diagonal. Esta matriz relaciona cada documento con los demás. El coeficiente de correlación de Jaccard se calcula con:

   $SIM(ta,tb) = \frac{ta.tb}{|ta|^2+|tb|^2-ta.tb}$

4. Construir una lista por cada documento donde se encuentren, organizados de mayor a menor, los documentos más relacionados. Esta lista se construye en base a la matriz construida en el paso 3.

2.1.2 Arquitectura:

La implementación de los pasos descritos anteriormente se hizo, en el programa serial, dividiendo el algoritmo en las siguientes partes:

1. Se toman un parámetro enviado por línea de comandos al ejecutar el programa, el cual dice en que carpeta están los archivos de texto a procesar. De esta ruta se saca todo el listado de archivos que tiene y es guardada en la lista $listaArchivos$.

2. Se realiza la construcción del vector tf (term frecuency) para cada documento. Esto se hace recorriendo la $listaArchivos$, abriendo el documento correspondiente y llamando a la función $frecuencia_termino$. Esta función recorre cada línea del documento, saca las palabras y recorre cada palabra limpiándola, aplicando el paso 1 propuesto en la sección de diseño. Una vez la palabra está limpia, la función la guarda en un diccionario, donde la llave es la palabra y el valor la frecuencia de aparición en ese documento. Al finalizar la función retorna el diccionario, que es tomado como vector de frecuencia de términos para ese documento y guardado en el diccionario $vectores_tf$, donde la llave es el nombre del documento y el valor el vector de frecuencia de términos de ese documento. Antes de pasar al siguiente documento en $listaArchivos$, el programa recorre el vector de frecuencia de términos del documento sacando las palabras que aparecen en el documento y construyendo el diccionario $vector_terminos$, donde se guarda la palabra como llave y el número de documentos en los que aparece como valor.

3. Se construye el vector $tfidf$ (term frecuency and inversed document frecuency) para cada documento, sacando para cada palabra en $vector_terminos$ la frecuencia con la formula descrita en el paso 2 de la sección de diseño y guardándola en el diccionario $vector_tfidf$, con la palabra como llave y la frecuencia como valor. Una vez recorridas todas las palabras, se guarda este vector en el diccionario $vectores_tfidf$, con el nombre del documento como llave y el vector $tfidf$ de este como valor.

4. Se construye la matriz de correlación, llenando solo la triangular superior y haciéndola igual a la triangular inferior. La matriz se construye aplicando la formula descrita en el paso 3 de la sección de diseño y se guarda en la variable correlacion, que es una matriz de la biblioteca numpy.

5. Se llama la función top, que toma la matriz de correlación y la lista de documentos y construye, para cada documento, una lista organizada con el nombre de los documentos más relacionados. Esto lo hace construyendo una lista de tuplas para cada documento, donde el primer elemento es el coeficiente de correlación y el segundo la posición del documento con el que se está relacionando en $listaArchivos$. Esta lista es organizada con el método sorted de Python y luego recorrida para construir la lista $documentos_relacionados$, sacando el segundo elemento de cada tupla de la lista y guardando el nombre del archivo correspondiente. Esta lista es guardada en el diccionario top_dic en el que el nombre del archivo es la clave y la lista organizada es el valor. Una vez se realiza la lista para cada documento la función retorna el diccionario.

6. En base al diccionario retornado se construye un archivo JSON con la lista de cada archivo.

2.2 Algoritmo Paralelo:

2.2.1 Metodología PCAM:

Para el cambio de la aplicación serial a la paralela se aplicó la metodología PCAM. La descripción de lo que se hizo en cada uno de los pasos se da a continuación:

- Particionamiento: Se reconocen varias oportunidades de paralelización, la primera es dividir los documentos para el cálculo de los vectores de frecuencia de términos, ya que la construcción de cada uno solo depende del documento. La segunda oportunidad es en el cálculo de los vectores tfidf, ya que una vez se tienen los vectores de termino de frecuencia y el vector de palabras, la construcción del vector tfidf de cada documento es independiente de cada uno. La tercera oportunidad es en la construcción de la matriz, se puede dividir los datos por cada documento, ya que el cálculo de una fila de la matriz, es decir, de las relaciones

de un documento con todas las demás es independiente del cálculo de las relaciones de otro documento.

- Comunicación: Entre estas tres oportunidades de paralelización debe haber una sincronización de los datos. La estrategia que planteamos es que al terminar el cálculo de los vectores en la oportunidad de paralelización, estos sean unidos por el master y luego este les da la información a los demás núcleos. Se planea utilizar la orden de Gather para que lleguen los datos al master y la orden de bcast para que la información llegue a todos los núcleos.

- Aglomeración: La estrategia que planteamos es dividir la lista de documentos entre el número de núcleos con los que se está corriendo el programa. De esta forma aglomeramos un número de documentos para que las tareas que se deben ejecutar en cada una de las oportunidades de paralelización se ejecuten en un solo nucleo y solo al final se haga la comunicación con el resultado de todos los documentos procesados, reduciendo así el número de comunicaciones que se deben de hacer

- Mapeo: A cada procesador se le asigna un pedazo de la lista de documentos, este procesador se encargará de realizar las tareas propuestas en particionamiento para cada documento y realizará la comunicación con el master, en cada etapa de sincronización, con el resultado de todos los documentos que tenía asignado

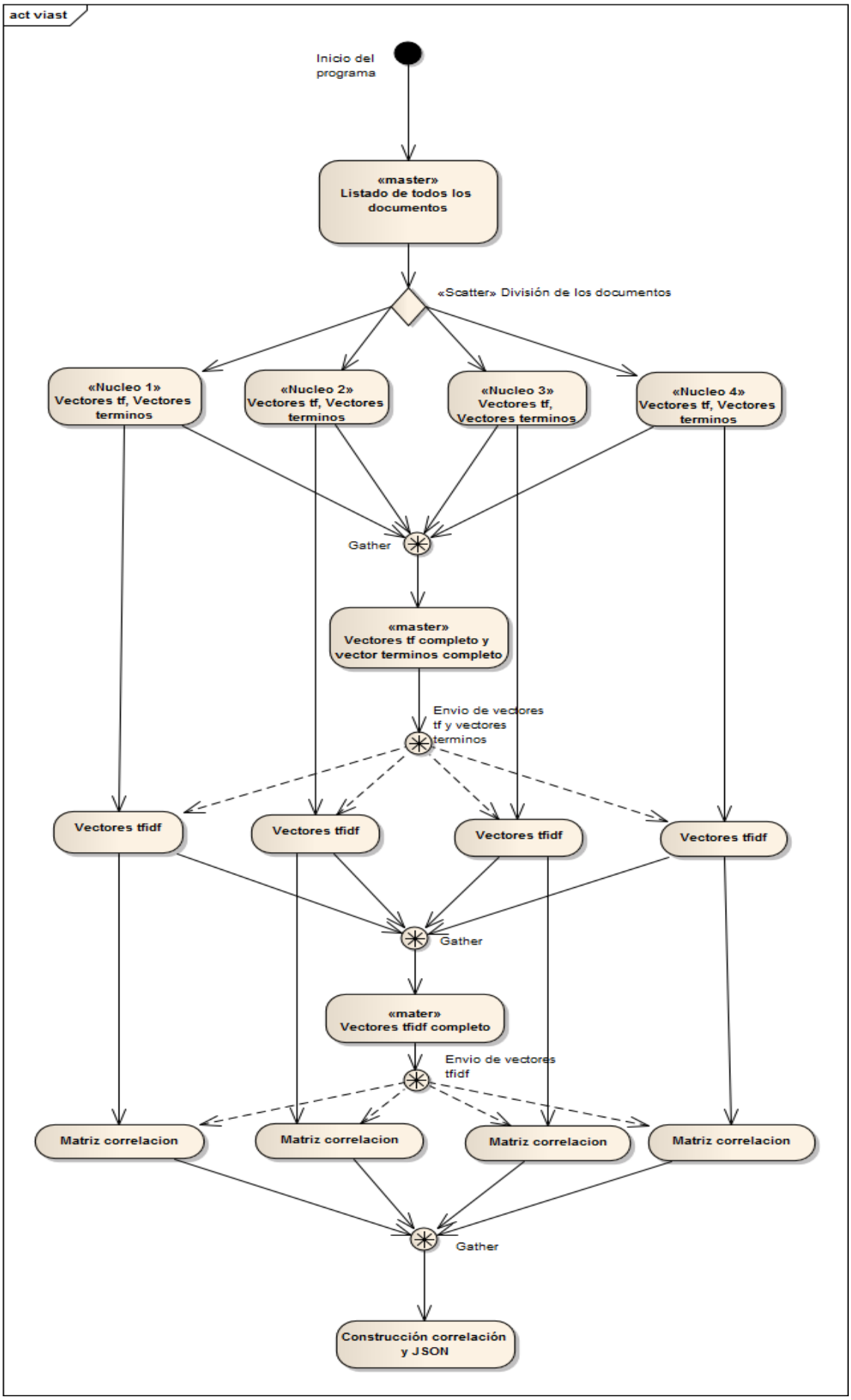
2.2.2 Diseño:

El diseño de la aplicación paralela está basado en el diseño de la aplicación serial, con algunas variaciones para realizar la paralelización. Como se describe en la sección anterior, se realiza la división de la lista de documentos en el número de núcleos. Cada uno de estos núcleos sigue los pasos descritos en el diseño, con la variación que se debe hacer la sincronización luego del cálculo de los vectores tf, de los vectores tfidf y de la matriz de correlación. El master es el encardo de unir estos vectores. La forma en cómo se implementa esta unión se describe en la sección de arquitectura.

2.2.3 Arquitectura:

La implementación de la aplicación paralela se hizo en base al código de la aplicación serial, las modificaciones que se hicieron fue para agregar el código que se debía ejecutar solo en el master, ya que el código restante, ejecutado en todos los núcleos es igual al serial. El código del master se encuentra en cuatro secciones del algoritmo. La primera se encarga de dividir *listaArchivos* en partes iguales para cada núcleo, esto lo hace tomando el número de núcleos, creando una lista para cada uno y recorriendo *listaArchivos* repartiendo cada elemento entre las listas, luego crea una lista guardando en cada posición la lista del núcleo y finalmente hace un scatter. La segunda sección de código que solo ejecuta el master se encarga de juntar los vectores tf que realizó cada núcleo. Esto lo realiza recorriendo cada diccionario que están en los vectores tf y haciendo el método *update*(), de tal modo que se cargue el diccionario de ese documento en el diccionario total. Luego recorre los diccionarios de palabras enviados por cada núcleo y crea un vector términos que tiene el número de documentos en los que aparece la palabra y la palabra. La tercera sección une los vectores tfidf realizados por cada núcleo, esto lo hace de la misma forma como unió los vectores tf. La última sección se encarga de unir los diccionarios de documentos relacionados que realiza cada núcleo y construir el JSON.

El modelo de comunicación se implementa mediante tres métodos: scatter, gather y bcast. La división de la lista de archivos se realiza mediante scatter, él envió de los vectores tf y vectores tfidf se realiza mediante gather y él envió de los vectores totales a todos los núcleos se realiza mediante bcast. El modelo de comunicación se puede ver mejor en la siguiente imagen.


act viast
Inicio del programa
«master» Listado de todos los documentos
«Scatter» División de los documentos
«Nucleo 1» Vectores tf, Vectores terminos
«Nucleo 2» Vectores tf, Vectores terminos
«Nucleo 3» Vectores tf, Vectores terminos
«Nucleo 4» Vectores tf, Vectores terminos
Gather
«master» Vectores tf completo y vector terminos completo
Envio de vectores tf y vectores terminos
Vectores tfidf
Vectores tfidf
Vectores tfidf
Vectores tfidf
Gather
«mater» Vectores tfidf completo
Envio de vectores tfidf
Matriz correlacion
Matriz correlacion
Matriz correlacion
Matriz correlacion
Gather
Construcción correlación y JSON

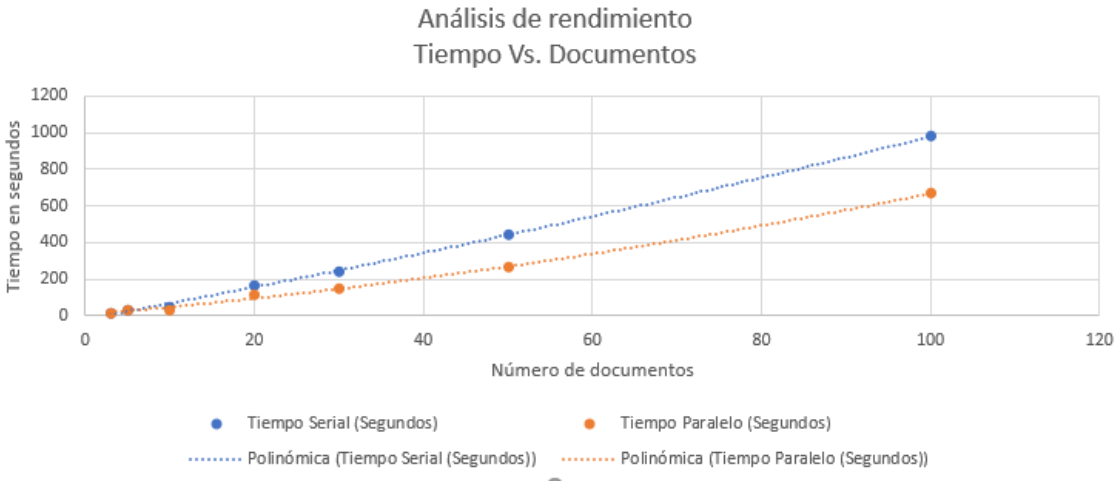
3 Análisis de estrategia:

3.1 Algoritmo serial Vs Algoritmo paralelo:

Para realizar el análisis del rendimiento de los algoritmos lo ejecutamos sobre el conjunto de documentos de Gutenberg. Para obtener datos relevantes, creamos subconjuntos de este de diferente tamaño, y tomamos los tiempos que tardaban cada uno de los algoritmos al ejecutarlos. La ejecución se realizó en un computador portátil de marca Lenovo con 4 GB de memoria RAM, 4GB de Swap y un procesador Intel Core i5-3337U de 64 bits a 1.80 GHz, 4 núcleos y tamaño de chache de 3072 KB por núcleo. A continuación, se presenta una tabla con el número de documentos a los que se enfrentaron los algoritmos, el tiempo que le tomo al algoritmo serial en segundos y el tiempo que le tomo al paralelo en segundos. Aclaramos que los documentos a los que se enfrentaron fueron los mismos.

Table 1: Tiempo del algoritmo serial Vs tiempo del algoritmo paralelo

Numero de documentos	Tiempo serial (Segundos)	Tiempo paralelo (Segundos)
3	12,9391109943	14,8496999741
5	34,4441080093	33,0745220184
10	49,0613729954	34,5520401001
20	169,485677958	113,978151798
30	239,22673893	147,615787983
50	446,585769892	262,950772047
100	977,850795984	668,934282064
150	1625,95473909	2111,34725809
200	2572,57577419	No aplica



Es importante resaltar de la tabla 1 varios elementos:

1. El tiempo que le tomó al algoritmo paralelo con muy pocos documentos (3) fue mayor que el tiempo que le tomó al algoritmo serial con la misma cantidad de documentos. Es en este caso que se puede evidenciar el tiempo que le toma al algoritmo paralelo preparar la ejecución y el tiempo que gasta en comunicación.

2. El único caso donde el algoritmo serial tiene un tiempo menor que el paralelo fue al enfrentarlo a un número muy pequeño de documentos, el algoritmo paralelo a medida que crece el número de casos, va agrandando la diferencia entre los dos tiempos (ver grafica 1). Esto se puede observar de forma más clara en la gráfica de porcentaje de cambio Vs número de documentos.

3. En el caso de 200 documentos a procesar el algoritmo paralelo no pudo terminar su ejecución (Ver sección de dificultades), pues este lleno la memoria RAM y el Swap de máquina y quedo sin recursos. En este caso se puede evidenciar que, para nuestro algoritmo, la cantidad de

memoria RAM consumida por el algoritmo paralelo es mayor a la consumida por el algoritmo serial.

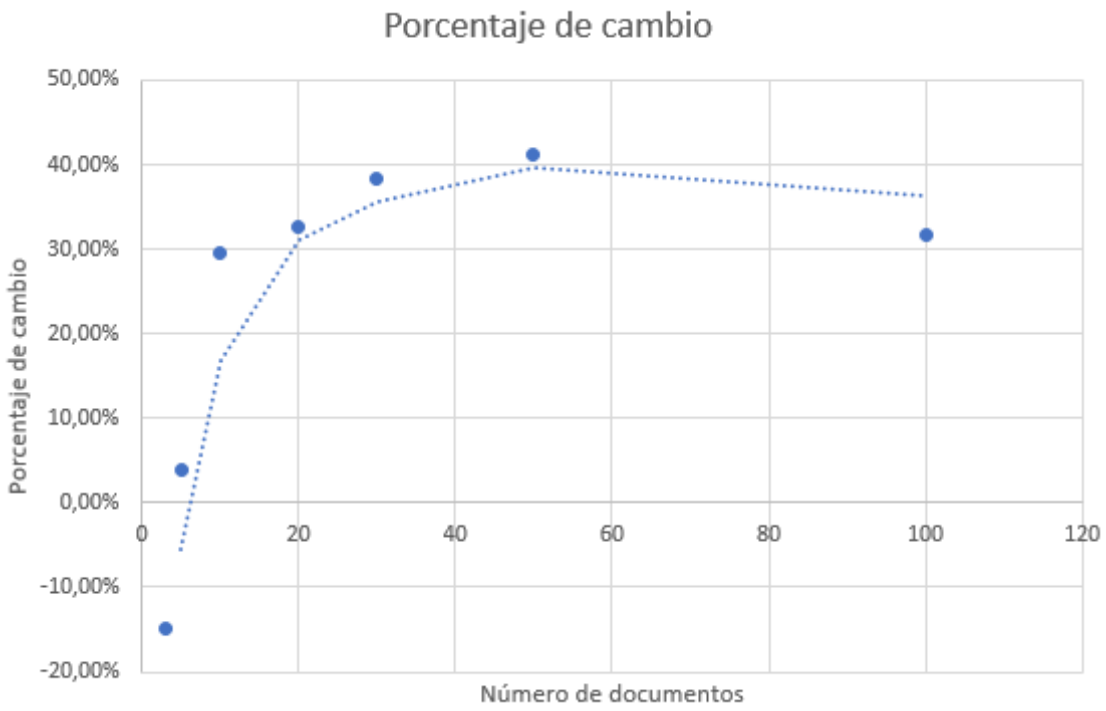
4. Los casos de 150 y 200 no son tenidos en cuenta para la gráfica pues su tiempo fue alterado por el tiempo de acceso a SWAP, ya que la memoria RAM se agotó.

5. Las líneas de tendencia encontradas en el gráfico con los datos son cuadráticas, lo que indica que la distancia entre el tiempo del algoritmo serial y del algoritmo paralelo va a ir creciendo a medida que se agregan más documentos al conjunto de datos.

En la tabla 2, se muestra el porcentaje de cambio en el tiempo del algoritmo paralelo con respecto al algoritmo serial.

Table 2: Porcentaje de cambio del tiempo del algoritmo paralelo con respecto al serial

Numero de documentos	Porcentaje de cambio
3	-14,77%
5	3,98%
10	29,57%
20	32,75%
30	38,29%
50	41,12%
100	31,59%



Es importante resaltar de la tabla 2 lo siguiente:

1. El algoritmo paralelo, con gran cantidad de datos, consume mucha RAM, por lo que el sistema operativo, al llenar la memoria RAM y empezar a consumir SWAP hace que el tiempo del algoritmo paralelo se vea alterado y los resultados no sean los esperados, este caso lo podemos ver cuando se dan 100 documentos a ejecutar. Esto sucede ya que el acceso a SWAP es mucho más lento que el acceso a RAM.

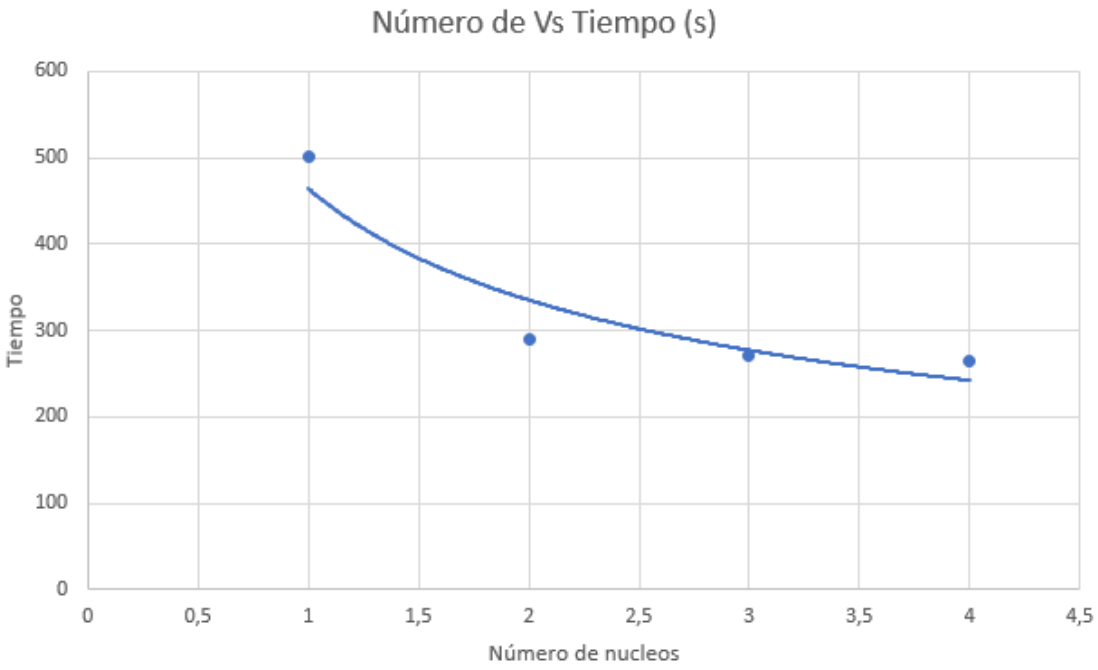
2. El porcentaje de reducción de tiempo va creciendo a medida que se va aumentando el número de documentos en el conjunto de datos.

3.2 Análisis del número de núcleos:

La tabla 3 relaciona el número de núcleos con los que se ejecuta el algoritmo paralelo para un conjunto de datos de 50 documentos y el tiempo que tarda.

Table 3: Algoritmo paralelo en diferente número de núcleos

Número de núcleos	Tiempo (Segundos)
1	501.110184908
2	286.63873601
3	279.128190041
4	267.679928064



Es importante resaltar de la gráfica anterior como al pasar de un solo núcleo a 2 núcleos el rendimiento se mejora en un alto porcentaje, sin embargo al seguir escalando en el número de núcleos la curva de mejora va descendiendo, es decir, aunque se mejora el rendimiento no es tanta la ganancia.

4 Dificultades:

En la ejecución del conjunto de datos de Gutenberg se encontraron varias dificultades:

1. El conjunto completo no se pudo ejecutar, ya que los algoritmos necesitaban más memoria RAM de la que se tenía disponible. Al ejecutar el programa, aproximadamente a las 2 horas de ejecución, la memoria RAM y el SWAP se llenaban, lo que ocasionaba que el computador se bloqueara y tratara de administrar los recursos. Al final, luego de varias horas, el sistema operativo decidía terminar el proceso, matándolo.

2. De igual forma, grandes conjuntos de documentos no fueron posible ejecutarlos. Solo se pudo llegar hasta 200 documentos con el algoritmo serial y 150 con el algoritmo paralelo.

3. El algoritmo paralelo para los conjuntos de datos de más de 100 documentos tiene variaciones en el tiempo debido a que su procesamiento requería de mucha memoria RAM, más de la disponible en el computador, por lo que el sistema operativo pasaba a utilizar el SWAP, área de intercambio de memoria mucho más lenta que la RAM.

4. Los algoritmos no se pudieron ejecutar en el cluster dado por el docente, ya que la cola de ejecución era muy larga y los procesamientos muy demorados (se pudo evidenciar un proceso que llevaba más de un día corriendo). Esto hizo que se tuviera que ejecutar en los computadores locales y los resultados no fueran tan precisos (problemas en la cantidad de datos), además que sufrieran alteraciones ya que el computador no estaba dedicado solo a esa tarea.

5 Código de Honor

Juan Pablo Gaviria, Felipe Montoya y David Alejandro Gomez declaran que este trabajo es auténtico, original, no copiado, no enviado a realizar por un tercero y se reconocen en este a los terceros que aportaron al proyecto, directa o indirectamente.

Firmas:

_____ _____ _____

References

[Hua08] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.