Tópicos Especiales en Telemática: Proyecto 4 Clústering de Documentos a partir de Métricas de Similitud basado en Big Data

Diego Alejandro Perez-Gutierrez y Edwin Montoya-Jaramillo
Escuela de Ingeniería, Pregrado de Ingeniería de Sistemas
Universidad EAFIT, Colombia
dperezg1@eafit.edu.co
emonto15@eafit.edu.co

***Abstract*— Clustering es una técnica eficaz que permite organizar grandes cantidades de archivos sin un orden en pequeños números de grupos con significado y sentido; Buscar en el gran volumen de documentos que se cuenta es una tarea lenta y pesada, por esto se han creado tendencias como el big data, solucionando problemas de grandes volumenes de datos, donde los datos son diversos y necesitan ser procesados casi en tiempo real.**

Este documento, toma como base el mundo de big data y compara la eficiencia del agrupamiento de documentos contra la clasica y robusta aproximació del paralelismo puro de HPC; Ambos usarán K-Means como algoritmo de agrupamiento. Ambos serán sometidos a diferentes conjuntos de datos, en los cuales se cambiará el número de grupos que se les piden y se tomarán sus tiempos para luego ser comparados y analizados para determinar si Big Data iguala o mejora el tiempo de procesamiento de datos.

PALABRAS CLAVES

K-Means, Spark, HPC, Hadoop, Agrupamiento, Mineria de Datos, YARN, HDFS

I. INTRODUCCIÓN

Con el creciente uso del internet, y el número de dispositivos que se conectan a este, nos estamos enfrentando a un volumen cada vez mayor de documentos que hay a disposición de las personas. La gran cantidad de textos que hay en Internet, las grandes colecciones de documentos en librerías digitales y repositorios, información personal digitalizada como artculos en blogs y e-mails, están en constante crecimiento día a día y cada vez esta información es mas diversa y no estructurada, debido a que el contexto de una persona ya no solo esta dado por las publicaciones en redes sociales, sino tambíen las noticias que le interesa, los libros que lee, la musica que escucha y toda esta información rompe el esquema tradicional de los sistemas OLAP y presentan un nuevo reto en la eficiencia y efectividad en la organización de los documentos.

Clustering es una técnica eficaz para el agrupamiento de una serie de vectores de acuerdo a un criterio, este puede ser la distancia euclidiana o similitud. Esta técnica permite de manera automática coger un conjunto de datos de documentos de texto y dividirlo en pequeos grupos llamados clusters que tienen un significado y una similitud, permitiendo de esta manera que al momento de realizar una búsqueda especifica por un tema o un archivo, no sea necesario buscar en todo el conjunto de datos sino solo en el grupo especifico al que este documento pertenece, y tener la posibilidad de recibir recomendaciones de otros archivos donde el contenido sea similar.

La agrupación de documentos de texto, los agrupa en caso de tener algún parecido y de esta manera formar un cluster coherente. Los documentos que no tienen parecido entre si son separados en diferentes cluster. Sin embargo la definición de si dos documentos son similares o diferentes no siempre es clara y puede variar según el problema.

Para conocer que tan similares son dos documentos hay varias técnicas que permiten comparar el contenido que hay en cada uno y de esta manera saber que tan parecidos son dos documentos, las diferentes técnicas que hay son Jaccard Coefficient, Pearson Correlation Coefficient, Average Kullback-Leibler Divergence, Euclidean Distance y Cosine Similarity Para los diferentes experimentos se utilizó el estándar K-Means que organiza los diferentes textos en grupos y la distancia Euclidiana que se encarga de establecer que tan distante es un documento de otro.

II. MARCO TEÓRICO

A. Representación de los documentos:

Para poder comparar un par de documentos mediante el uso de computadores hay que llevarlos a alguna estructura de datos conocida, pueden variar entre vectores, diccionarios, conjuntos y cualquier otra estructura de datos, inclusive se pueden crear algunas nuevas para su representación. La mejor forma de hallar la similitud entre dos documentos es mediante el peso de sus palabras, es decir, que tantas veces un término se repite y que tan relevante es dentro de todos los documentos; Si analizamos estos pesos de las palabras en cada documento podemos determinar si dos documentos hablan de lo mismo, esto se evidencia claramente en conjuntos de datos basados en noticias, donde las palabras que más se repiten en cada noticia son casi las mismas cuando se comparan varias noticias de la misma sección. En teoría las palabras más frecuentes describen el tema del documento, pero en la práctica, cuando se analiza un documento sin

procesar, realmente las palabras más frecuentes son los conectores, palabras como la, en, mas, lo, se, de, etc. Por esto es completamente necesario removerlas antes de llevarlas a alguna estructura de datos, existen conjuntos de datos dedicados a estas palabras para su fácil remoción.

B. Preprocesamiento de los documentos:

Para aplicar acertivamente un algoritmo de agrupamiento luego de tener los documentos en alguna estructura de datos, se necesita representar con un mayor nivel de relevancia los documentos, es decir, la forma en que se represente un documento debe ser mas allá de un arreglo que contenga todas sus palabras, se tiene que llegar a una forma estandar de representación donde un número pueda describir la relevancia de cada palabra tanto dentro del documento como dentro del conjunto total de datos. La propuesta que presentan normalmente las librerias de agrupamiento de las diferentes tecnologías es un estandar llamado matriz td-idf. Esta matriz se compone de dos estadisticos: El primero, es la frecuencia de una palabra dentro del documento o tf (Term Frecuency), esta puede realizarse de varias maneras, la más común es un conteo crudo de las veces que se encuentra un término en el documento, también se puede realizar una frecuencia relativa del término dentro del documento, incluso se puede comparar la frecuencia cruda de cada palabra con la frecuencia más alta, entre muchos otros estadisticos.

binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$

El segundo estadístico consiste en que tan raro es el término dentro de todos los documentos, o frecuencia inversa de documento (Inverse Document Frecuency por sus siglas en inglés); Este usa la siguiente fórmula [7]:

$$\operatorname{idf}(t, D) = \log \frac{N}{1 + |\{d \in D : t \in d\}|}$$

Donde:

- idf(t,D) representa la frecuencia inversa del término t en el conjunto de documentos D.
- N representa el número total de documentos.
- 1+ $|\{d \in D : t \in d\}|$ representa el número de documentos d en los cuales está el término t. Se le suma uno para evitar la división por cero en caso que la palabra no exista

El proposito del paso idf se lo asignó Karen Sparck Jones estableciendo que las palabras más frecuentes en todos los documentos resultarán siendo las menos importantes y las no tan frecuentes serán las más representativas.

Este par de estadisticos se iteran por cada palabra y por cada documento, generando la matriz, donde las filas representan los documentos y cada columna representa cada término. Algunas veces la cantidad de términos es excesiva y se vuelve muy complejo el manejo de la matriz, por lo que se pueden realizar heuristicas para la reducción de los términos, por ejemplo, es muy usual remover esas palabras que son muy frecuentes, o las que no se repitan más de una vez, o incluso acotar a ambos lados, todo esto depende del conjunto de documentos por lo que se debe conocer con anterioridad para establecer un buen criterio de acotamiento.

C. Agrupamiento de documentos:

El procedimiento de agrupamiento para este caso será no supervisado, lo que quiere decir que no hay un entrenamiento previo del algoritmo, lo que nos lleva a implementaciones como el k-means; Este algoritmo consiste en seleccionar k elementos del conjunto de datos que denominaremos centroides, estos son los puntos de partida para los grupos, luego comparamos cada documento del conjunto y decidimos (de acuerdo con el índice de similitud) a cual grupo pertenece cada documento, resultando en grupos preliminares. El siguiente paso es recalcular el centro, esto es esencial ya que es posible que el centroide que se seleccionó aleatoriamente no sea el centro real del grupo, por lo que se deben comparar todos los documentos de cada grupo entre ellos, luego podemos definir el documento centro. Se considera que la clasificación esta lista cuando la relación intragrupo es máxima e intergrupal es mínima; es complicado establecer cuando estas condiciones se cumplen, esto depende completamente del criterio humano, por ejemplo, si se quiere ser muy estricto se podría establecer que únicamente un elemento pertenece a un grupo cuando su índice de similitud con todos los demás elementos es mayor al 70%, y solo aceptar la cantidad de grupos cuando se seleccionen dos elementos de grupos diferentes y tengan un índice de similitud inferior al 30%. Estos márgenes pueden añadirle mucha precisión al resultado pero puede dejar muchos documentos por fuera. Este algoritmo se considera NP-Hard lo que significa que es computacionalmente difícil de abarcar, esta clasificación se deriva desde el algoritmo de similitud siendo simplemente aproximaciones, luego está la selección del k, y por último está la definición de los centroides; Estas consideraciones de precisión no están dentro del alcance del proyecto por lo que se realizan heurísticas muy básicas para su determinación.

D. Problemática:

La clasificación de documentos parte de unos conjunto de datos base y los divide de acuerdo con los pesos de

sus palabras, donde estas divisiones suponen una división temática, este proceso se realiza para poder brindar dos soluciones: la primera es cuando ingrese un nuevo documento solo tendrá que ser comparado contra los centros de los grupos, segundo cuando alguien busque un documento se le podrá presentar a este una serie de documentos que de alguna u otra forma tienen relación con su búsqueda y puedan ayudarle a solucionar su duda (un buen ejemplo de esto es el motor de búsqueda de Google). Para que esto funcione el conjunto de datos inicial debe ser lo suficientemente voluminoso y variado para poder brindar una mayor cobertura de documentos.

Esto tiene implicaciones directas en la forma de atacar el problema de clasificacón ya que para poder atacar este tipo de problemas es necesario distribuir la información, lo cual se complejisa mucho si no se tiene un ambiente que facilite la distribuicion de esta como lo hace hadoop, al hacer uso de este el programador no tiene porque preocuparse de como distribuir la información, se debe de preocupar mas de pasarla y luego manejarla.

III. Análisis y Diseño mediante Analítica de datos (ETL-Procesamiento-Aplicación)

Los procesos ETL son un término estándar que se utiliza para referirse al movimiento y transformación de datos. Se trata del proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y cargarlos en otra base de datos (denominada data mart o data warehouse) con el objeto de analizarlos. También pueden ser enviados a otro sistema operacional para apoyar un proceso de negocio.

En definitiva, el principal objetivo de este proceso es facilitar el movimiento de los datos y la transformación de los mismos, integrando los distintos sistemas y fuentes en la organizacin moderna.

El término ETL corresponde a las siglas en inglés de extract, transform y Load.

A. Extracción:

La primera fase en el proceso ETL es la de extaer los datos desde el sistema de origen; En nuestro caso este proviene HDFS. Los datos que obtenemos provienen en un formato de texto plano facil de manejar. La extracción se encarga de convertir los datos a un formato de destino preparado para iniciar el proceso de transformación, que en este caso consiste en la creacion de una tabla en formato Hive (parecido a SQL), en donde la primera columna corresponde a la ruta del archivo y la segunda al texto que este contiene.

B. Transformación:

La fase de transformacíon aplica una serie de funciones sobre los datos extraídos para facilitar el procesamiento. Para el caso de estudio necesitamos convertir cada documento de texto a alguna forma numerica para ser procesada por el algoritmo de agrupamiento. Spark contiene una libreria llamada mllib la cual contiene el algoritmo K-means; este no trabaja con datos tipo texto por tanto es necesario transformarlos a valores numéricos que representen y estandaricen las palabras de los documentos.

En este caso se utilizó la matriz tfidf. Comenzando por separar las palabras de cada documento para ser tratadas individualmente, seguido por remover las palabras que se consideran *stopwords* que no tienen mayor relevancia acerca del contenido del documento. Una vez tenemos las palabras que suponen describir el texto pasamos a listar sus respectivas frecuencias. Luego de listar realizamos un estadistico llamado frecuencia inversa de documento que se encarga de quitar importancia a aquellas palabras que se repiten en muchos documentos ya que no es un factor diferencial entre ellos.

Una vez pasamos todos los documentos por este proceso obtenemos como resultado una matriz m x n en donde m representa la cantidad de documentos,n la cantidad de palabras y la posición [m][n] el peso de la palabra n en el documento m; esta matriz será el punto de partida para realizar el entrenamiento del algoritmo *K-Means*.

C. Carga:

La fase de carga es el momento en el cual los datos de la fase anterior (transformación) son cargados en el sistema de destino. Para el caso, este destino será el modelo de *K-Means*. Este modelo se crea apartir de un conjunto base de documentos estandarizados por el paso previo, para generar los centros de los grupos.

D. Procesamiento:

Esta etapa de las aplicaciones (especialmente aquellas que trabajan apartir de modelos) es la esencial, debido a que es la encargada de apartir de un conjunto base generar el modelo para las predicciones futuras, este punto es muy delicado debido a que es donde los parametros de entrenamiento pueden afectar rotundamente la eficiencia y veracidad del modelo, por ejemplo en caso que el conjunto base no sea lo suficientemente amplio las consecuencias serán asumidas por modelo, debido a que los grupos estarán diseñados muy especificamente para este pequeño conjunto. Otro punto a considerar es el números de grupos que se le pide modelar, sí son muy pocos existirán documentos que puedan no tener mucha relación con los demás y en caso de ser demasiados dos documentos similares pueden quedar en grupos diferentes.

E. Aplicación:

Esta fase es en la que realmente comenzamos a usar nuestro modelo; Basandonos en este podemos comenzar a ingresar nuevos documentos y estos se irán clasificando de acuerdo al modelo. Esta es la fase que realmente las personas puedan hacer uso del historial que se recolectó para recibir recomendaciones basadas en busquedas, incluso podemos tomar retroalimentación de esta fase, ya que entran nuevos documentos que posiblemente ameriten un cambio en el modelo y asi ir creciendo el conjunto de entrenamiento al punto de casi que cubrir la mayoria de los casos posibles.

IV. IMPLEMENTACIÓN

Comenzamos partiendo del hecho que usaremos un *cluster* de Big Data, la programación aquí gira en torno al mapeo y la reducción de los datos, donde el mapeo consiste en agrupar, filtrar o ordenar los datos de forma que la reducción se aplique a un subconjunto de datos que tengan algo en común, el ejemplo más representativo es tener un arreglo de estudiantes que pertenece a una materia; el mapeo sería agrupar los estudiantes de la misma materia y la reducción será la suma de los estudiantes de la misma materia.

Este clasico ejemplo se usa para explicar MapReduce nativo del proyecto de Apache Hadoop, pero con el auge de esta tecnologia, nuevas soluciones surguieron y una de estas es Spark, este lenguaje (basado en Scala) incluye la noción de programación en forma de grafo dirigido no ciclico o DAG por sus siglas en inglés, donde los vertices del grafo son los RDD (Resilient Distributed Dataset) estas estructuras de datos están optimizadas para ser distribuidas a lo largo del *cluster* y son transformadas por funciones(representando los arcos del grafo) para resultar en un nuevo RDD. Todas estas operaciones son realizadas en memoria, lo que reduce hasta cien veces el tiempo de ejecución comparado con MapReduce.

Pasando a la implementación de este articulo se comienza por establecer el contexto de Spark, este contexto es la puerta hacia el mundo HDFS, el cual nos sirve para la lectura de varios archivos por medio de la función wholeTextFiles que se encarga de abrir los archivos en el directorio que es pasado por parametros al programa. Una vez abiertos creamos un DataFrame por medio de SparkSession, objeto de la libreria sql de spark para el manejo de DataFrames y DataSets; Para esto establecemos una estructura que será parecida al mundo de bases de datos relacional, donde existen dos columnas, la primera corresponde a la ruta del archivo, y la segunda a su contenido en formato texto. Una vez tenemos los documentos en memoria pasamos a la declaración de las transformaciones que se le realizarán implementadas en la biblioteca feature contenida por mllib de spark. Comenzando con la separación de las palabras, para esto declaramos un Tokenizer que leerá la información de la columna de texto para luego agregar una nueva columna con un arreglo de las palabras de cada documento. Pasamos luego a remover las *stopWords*, aquellas palabras que se usan como conectores o no contribuyan mucho al contenido esencial del texto, esto se realizó por medio de la clase StopWordsRemover, la cúal tiene la opción de cargar el idioma de los documentos para preparar un conjunto de palabras con los *stopWords* de este, luego simplemente establecemos cúal es la columna que usará y en cual dejará el nuevo arreglo. En este punto tenemos un arreglo de palabras mas puro y relevante, para pasar al paso del conteo, aquí usamos HashingTF, este nos permitirá generar un mapa de palabras, donde su clave es la palabra y el valor es la frecuencia en el documento; Cuando los documentos son muy extensos y variados es recomendable limitar el numero de palabras diferentes, en este caso el usuario puede establecer el número de palabras diferentes con el cúal quiere trabajar, como los pasos anteriores, establecemos la columna entrada y la columna salida. Pasamos luego a establecer que tan relevante es cada palabra dentro de todos los documentos usando la clase IDF de spark, este se encargará de buscar que tan frecuente es la palabra en los diferentes documentos y castigar aquellas que son muy frecuentes, este paso recibe la salida del hashing tf y deja su resultado en una nueva columna, esta clase tiene la opcion de limitar las palabras a tener en cuenta, en este caso establecemos que tenga en cuenta las palabras que están minimo en un documento, es decir, todas, pero existe la posibilidad de incrementar el parametro para ignorar palabras no tan frecuentes en los documentos. El paso anterior deja como resultado una lista de tuplas, donde el primer valor de esta representa la palabra, y el segundo el indice luego del idf, esta salida será usada luego, por ahora solo declaramos un KMeans para establecer el número de cluster que el usuario pida por argumentos.

Hasta este punto lo único que tenemos es la declaración de las etapas que atravesará el DataFrame. Pasamos luego a declarar el *pipeline* o linea de esamblaje, esto es establecer el orden de las etapas a realizar. Por medio del método fit del pipeline podemos crear el modelo para el KMeans; Este método recibe como parametro el DataFrame base para el entrenamiento del KMeans, es decir, todos los pasos realizados anteriormente, fueron exclusivos para el entrenamiento del KMeans, lo que nos deja como resultado un modelo con solo los centros de los establecidos, luego pasamos a predecir el nuevo conjunto de datos que para el caso de esta implementación será el mismo al de entrenamiento, esto arrojará como resultado los grupos explícitos que fuerón conformados para hallar los centros; Esto se realiza por medio de la funcion transform del modelo. Para cerciorarse que los datos esten consistentes en memoria llamamos la funcion cache que asegura tener los datos en memoria. Para mostrar los datos al usuario dividimos la ruta del documento por el ultimo slash y obtenemos solo el nombre del documento, para luego aprovechar la estructra sql para realizar un select (tipico de las bases de datos relacionales) donde se agrupa por el número de grupo y se crea un vector con los nombres de los documentos que pertenecen al grupo, para luego imprimirlos en formato JSON.


```
{"prediction":"1",
  "cluster":["Abraham Lincoln___Lincoln Letters.txt",
    "Abraham Lincoln___Lincoln's First Inaugural Address.txt",
    "Abraham Lincoln___Lincoln's Gettysburg Address, given November 19, 1863.txt",
    "Abraham Lincoln___Lincoln's Inaugurals, Addresses and Letters (Selections).txt",
    "Abraham Lincoln___Lincoln's Second Inaugural Address.txt",
    "Alfred Russel Wallace___Is Mars Habitable?.txt"]}
{"prediction":"3",
  "cluster":["Alfred Russel Wallace___Darwinism.txt"]}

{"prediction":"2",
  "cluster":["Alfred Russel Wallace___Island Life.txt"]}

{"prediction":"0",
  "cluster":["Alfred Russel Wallace___Contributions to the Theory of Natural Selection.txt",
  "Alfred Russel Wallace___The Malay Archipelago, Volume 1.txt"]}
```


Grafico 1. Salida ejemplo.

V. Análisis de resultados

A. Conjunto de datos:

Existen muchos conjuntos de datos para realizar pruebas de minería de datos, clasificación, etc. Estos son conjuntos con documentos desde mil hasta ocho mil documentos perfectos para estas tareas. Para efectos de validación del algoritmo estos conjuntos son tan extensos y tardan tanto tiempo, se diseñó un pequeño conjunto con 8 documentos, todos muy sencillos con máximo 5 palabras diferentes para así poder identificar los grupos, los nuevos centros. Una vez pasadas las pruebas pequeñas, se buscó un conjunto de datos reales para que cobre algún sentido la clasificación, para esto visitamos la página llamada *Gutenberg* la cual ofrece libros digitales gratis en diferentes formatos, pdf, ebook, etc. Pero igualmente ofrecen diferentes conjuntos desde trecientos hasta más de setenta mil libros en formato de texto plano, para este proyecto se seleccionó la versión de cuatrocientos sesenta y dos documentos. Este conjunto es uno de los más comunes para realizar algoritmos de clasificación, por lo que es de interés del proyecto evaluar el rendimiento frente a este.

B. Experimentación:

Para este punto se planteará una serie de comparaciones entre las Herramientas de Big Data (El ecosistema Hadoop) y las herramientas del HPC puro (High Performance Computing) donde el algoritmo fue implementado usando las librerias de MPI para Python, lo que nos dará un punto base para ver que tan eficiente son las nuevas herramientas. Para poder realizar una comparación justa ambos algoritmos trabajaron con 10 núcleos y analizaron el mismo conjunto de datos, y se verificó que los resultados fueran veridicos en ambos casos.

Comenzando con HPC, este arrojó los siguientes tiempos con los diferentes valores de K (El número de grupos):

HPC (High Performance Computing)	
K	**Tiempo (Segundos)**
4	647.55
5	679.01
6	1008.88
7	685.85
8	899.24
9	814.88
10	565.88
11	643.43
12	348.15

Tabla 1. Tiempo de ejecución para HPC en segundos.

Luego de analizar la tabla vemos un tiempo promedio de 699.20 segundos, lo que cambiando de escala serían 11.65 minutos. Con una desviación estandar de 192.57 segundos, lo que nos dice que los tiempos varian del promedio un poco mas de tres minutos.

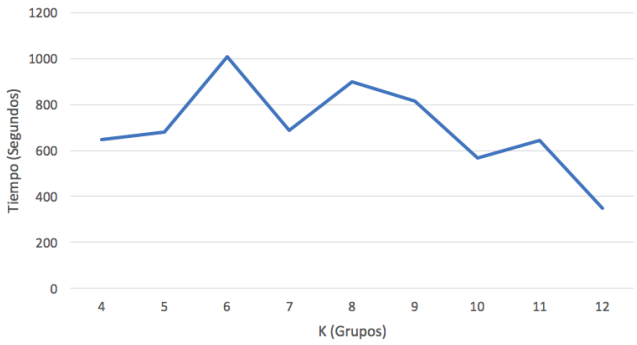

Grafico 1. Gráfico lineal para HPC que muestra como se comporta el tiempo en función de la K.

Una vez tomados los tiempos de HPC pasamos a ejecutar el algoritmo en un cluster de big data, cabe resaltar que el conjunto de datos es el mismo y las condiciones computaciónales teoricas tienen una diferencia despreciable.

Big Data (PySpark)	
K	**Tiempo (Segundos)**
4	376.36
5	485.67
6	409.52
7	527.60
8	517.88
9	315.63
10	286.96
11	311.57
12	289.27

Tabla 2. Tiempo de ejecución para Big Data en segundos.

Analizando los resultados arrojados por Big Data, vemos un tiempo promedio de 392.27 segundos, que significan unos seis minutos. Con una desviación estandar de 97.11 segundos o un poco menos de dos minutos.

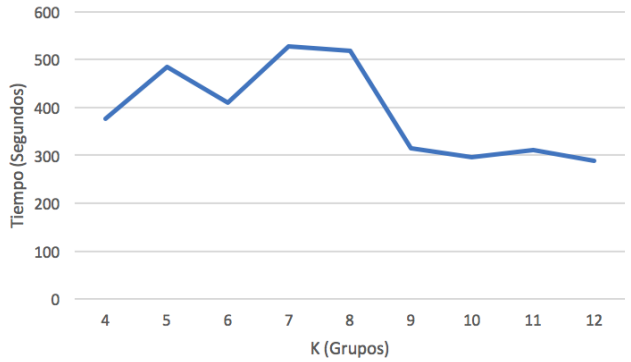

Grafico 2. Gráfico lineal para Big Data que muestra como se comporta el tiempo en función de la K.

Comparacíon entre Big Data y HPC		
K	Tiempo (Big Data)	Tiempo (HPC)
4	376.36	647.55
5	485.67	679.01
6	409.52	1008.88
7	527.6	685.85
8	517.88	899.24
9	315.63	814.88
10	286.96	565.88
11	311.57	643.43
12	289.27	348.15

Tabla 3. Comparación de tiempos de ejecución entre Big Data y HPC en segundos.

Teniendo la comparación de los tiempo observamos un promedio de mejora de 306.93 segundos de Big Data con respecto a HPC, que representa casi cinco minutos de diferencia beneficiando a Big Data

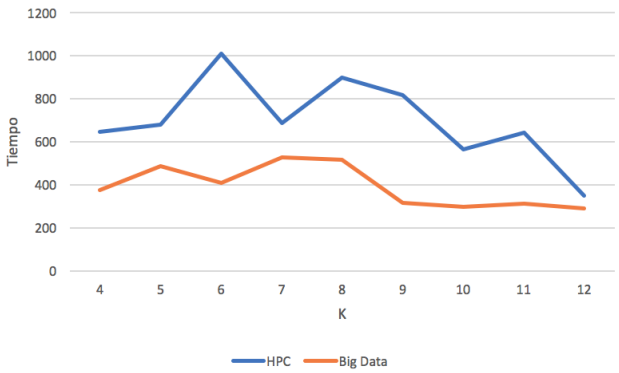

Grafico 3. Gráfico lineal para Big Data y HPC que permite evidenciar las diferencias.

En este gráfico apreciamos la sustancial mejora en tiempos para Big Data, aunque se puede apreciar una tendencia en HPC en los valores más altos de K a igualar los tiempos de Big Data.

VI. Conclusiones

- Algunas personas consideran que Big Data es un subconjunto de HPC, pero esta experimentación pudo demostrar que la tradición paralela de HPC no fue mejor que las herramientas especializadas de Big Data para el procesamiento masivo de documentos.
- No se esperó que las diferencias fueran tan contundentes entre HPC y Big Data e incluso se pensó que HPC pudo haber ganado esta comparación debido a la desigualdad en la practica de las condiciones, HPC fue ejecutado en una sola maquina con un poder computaciónal alto sin necesidad de salir a internet, mientras que Big Data se ejecutó en tres maquinas conectadas por Ethernet (1 Gbps. teórico) que suponia un incremento en la latencia, pero aún así superó a HPC.
- El uso de herramientas como Spark que optimizan apartir del concepto *in-memory* pueden ser hasta un 50% mas eficientes que otras aproximaciónes para el computo masivo.
- La distribución de los documentos por HDFS fue parte clave del exito de Big Data, ya que al distribuir los datos en las maquinas y prevalecer la localia, las porciones de datos son computados en la maquina donde residen, disminuyendo las comunicaciones.
- Para este algoritmo de clustering es mas robusta frente al numero de clusters que se le piden, siendo un poco mas confiable.

References

[1] Anna Huang (2008). "Similarity measures for text document clustering. Proceedings of the Sixth".
[2] https://www.gutenberg.org/
[3] https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/bid/312584/procesos-etl-definici-n-caracter-sticas-beneficios-y-retos
[4] https://ezproxy.eafit.edu.co/login?url=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=55715dos1
[5] Fernando Berzal "clustering" http://elvex.ugr.es/decsai/intelligent/slides/dm/D3%20Clustering.pdf
[6] $https://es.wikipedia.org/wiki/Extract,_transform_and_load$
[7] $https://en.wikipedia.org/wiki/Tf%E2%80%93idf$
[8] $http://spark.apache.org/docs/2.2.0/api/python/$
[9] $https://hortonworks.com/$
[10] $http://hadoop.apache.org/$