RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Una revisión de los algoritmos paralelos para el procesamiento de grandes volúmenes de datos

Airel Pérez Suárez

RT_032 **febrero 2016**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Una revisión de los algoritmos paralelos para el procesamiento de grandes volúmenes de datos

Airel Pérez Suárez

RT_032 **febrero 2016**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Tabla de contenido

Una revisión de los algoritmos paralelos para el procesamiento de grandes volúmenes de datos

Airel Pérez Suárez

Equipo de Investigaciones de Minería de Datos, CENATAV - DATYS, La Habana, Cuba
asuarez@cenatav.co.cu

RT_032, Serie Gris, CENATAV - DATYS
Aceptado: 11 de Noviembre de 2015

Resumen. El agrupamiento es una de las técnicas de Minería de datos, que permite la extracción de conocimiento útil a partir de grandes volúmenes de datos. No obstante, el tamaño de las colecciones que se generan hoy en día dificulta la aplicación de la mayoría de los algoritmos de agrupamiento existentes, debido al costo computacional o en memoria que estos tienen. En este trabajo, se hace una revisión de los principales algoritmos paralelos y/o distribuidos reportados en la literatura que permiten procesar conjuntos de datos muy grandes e incluso, a los denominados *Big data*.

Palabras clave: minería de datos, agrupamiento, agrupamiento paralelo, big data.

Abstract. Clustering is an essential data mining technique that allows to extract useful knowledge from very large datasets. Nevertheless, the size of the collections that are generated nowadays makes it difficult to apply most of the reported clustering algorithms, either by their computational complexity or they memory compsumtion. In this work, we present a review of the most significant parallel and distributed clustering algorithms able to deal with very large datasets or even *Big data*.

Keywords: data mining, clustering, parallel clustering, big data.

1. Introducción

En la actualidad, redes sociales como Facebook y Twitter tienen billones de usuarios que producen más de cientos de gigabytes de información por minuto [1]. Las tiendas on-line se mantienen todo el tiempo almacenando datos acerca de las compras de sus clientes, mientras que el servicio de procesamiento de contenido de Youtube, con un billón de usuarios que generan 100 horas de vídeo cada hora, procesa más de 400 años de vídeo cada día [1,2]. Para ser capaces de procesar eficientemente estos grandes volúmenes de datos y poder extraer conocimiento de los mismos, es necesario utilizar técnicas de Minería de Datos. Una de las técnicas que se han usado para este propósito es el *agrupamiento*.

El agrupamiento es una de las técnicas más importantes dentro de la Minería de Datos y el Reconocimiento de Patrones. Esta técnica se encarga de organizar una colección de objetos en clases o grupos, de forma tal que los objetos de un mismo grupo se parezcan más entre sí que objetos de grupos diferentes [3]. De forma general, se asume que los objetos a procesar por un algoritmo de agrupamiento están descritos por m rasgos o variables, cuyos valores pueden ser cuantitativos y/o cualitativos. A su vez, las variables pueden ser univaluadas, si toman un solo valor del dominio de valores posibles, o multivaluadas, si pueden

tomar más de un valor. Adicionalmente, existen problemas en los que se puede desconocer el valor para algunos de los rasgos (i.e., rasgos incompletos) para uno o más objetos [4].

En la literatura se han propuesto muchos algoritmos de agrupamiento. Aunque de forma general los investigadores han tratado de reducir la complejidad de los algoritmos, así como de aumentar la escalabilidad y velocidad de los mismos, la mayoría de los algoritmos existentes son incapaces de procesar lo que los científicos han denominado *Big Data*. Bezdek y Hathaway han definido en [5], aceptándose por la comunidad internacional, que los conjuntos denominados *Big data* son aquellos cuyo tamaño es superior a 10^{10} bytes. Algoritmos tradicionales y poco costosos como el Kmeans, incluso para $k = 2$, son incapaces de procesar semejantes conjuntos de datos, en un tiempo razonablemente corto.

Una de las técnicas que se han utilizado recientemente en la literatura para agilizar tareas de cómputo es el procesamiento paralelo y/o distribuido. En el cómputo paralelo, un problema es usualmente dividido en varias partes discretas que pueden ser solucionadas concurrentemente, las cuales a su vez se pueden descomponer en un conjunto de instrucciones. En tal sentido, el principal objetivo de los investigadores ha sido aumentar la escalabilidad y velocidad de los algoritmos sin sacrificar mucho la calidad de los mismos.

En este trabajo se hace una revisión de los principales algoritmos paralelos y/o distribuidos que se han reportados en la literatura para procesar conjuntos de datos muy grandes e incluso, los denominados *Big Data*.

El resto del trabajo está estructurado como sigue: en la sección 2 se describen los algoritmos paralelos más importantes reportados en los últimos años para el procesamiento de grandes volúmenes de datos, resaltándose las limitaciones de los mismos. Finalmente, en la sección 3 se dan las conclusiones del trabajo y se enuncian áreas de interés científico que es necesario desarrollar.

2. Algoritmos paralelos y/o distribuidos para conjuntos de datos muy grandes

En esta sección se describen los algoritmos paralelos reportados para el procesamiento de grandes colecciones de datos. Es preciso aclarar que en este estudio solo fueron tomados en cuenta aquellos algoritmos que fueran capaces de procesar al menos 500000 objetos. Para una mejor comprensión, los trabajos analizados fueron organizados de acuerdo al enfoque principal seguido por los mismos.

2.1. Basados en MapReduce

En esta sección se describen aquellos algoritmos que basan su funcionamiento en MapReduce, la cual es una tecnología diseñada para realizar cómputos paralelos en la nube, utilizando un grupo de computadoras. MapReduce es muy utilizada hoy en día debido a su alta eficiencia en el procesamiento de conjuntos de datos muy grandes.

Uno de los algoritmos basados en MapReduce fue propuesto en [6]. Este algoritmo combina la teoría IB (del inglés Information bottleneck) con los algoritmos basados en el cálculo de centroides. Con este objetivo el algoritmo se divide en dos fases. En la primera se utiliza un algoritmos aglomerativo clásico, combinado con la teoría IB, para construir el conjunto de centroides iniciales. Con este propósito, el conjunto de datos es dividido en m partes y asignados a cada uno de los nodos existentes. Luego, cada nodo en su función *Map* utiliza el algoritmo aglomerativo anteriormente mencionado para determinar k sub-centroides a partir de su conjunto de datos. Cuando cada nodo termina su procesamiento, se ejecuta la función *Reduce*, que se encarga de combinar los k sub-centroides hallados en cada nodo, para determinar el conjunto k de centroides. Los centroides calculados por cada nodo son pasados al nodo principal quien se encarga de formar el conjunto total de centroides y pasarlo a cada uno de los nodos, los cuales se

encargan de determinar, para cada objeto asignado al mismo, su centroide más cercano. Esta información es posteriormente utilizada por la función *Reduce* para actualizar los centroides a partir de las nuevas asignaciones. Los dos pasos anteriores se repiten hasta alcanzar la condición de parada.

En [7] se propone una variante del algoritmo de agrupamiento CLARA, utilizando MapReduce. CLARA es un algoritmo iterativo que extiende el algoritmo de agrupamiento basado en medoids PAM [8], para el procesamiento de grandes volúmenes de datos. Inicialmente, en cada uno de los nodos existentes se ejecuta una función *Map* que se encarga de seleccionar del conjunto de datos una muestra aleatoria, la cual es procesada posteriormente en la función *Reduce*, utilizando el algoritmo PAM, buscando el conjunto de k medoids que agrupa a dicha muestra. Posteriormente, el conjunto de datos se divide en n conjuntos disjuntos, asignándose un conjunto a cada uno de los nodos existentes. A estos nodos se les asigna también el conjunto de n grupos de medoids determinados por los nodos en la primera tarea *MapReduce*. Es en este momento en el cual cada nodo ejecuta una función *Map* que se encarga de calcular para cada uno de los objetos asignados, su medoids más cercano dentro de cada uno de los n grupos de medoids previamente calculados, así como su distancia a dicho medoids. Finalmente, se ejecuta en cada nodo una función *Reduce* que se encarga de sumar las distancias calculados para un mismo conjunto de medoids. El conjunto de medoids con la menor suma representa el mejor agrupamiento.

En [9] se propuso una variante paralela de un algoritmo de agrupamiento basado en teoría de grafos, sobre MapReduce. Sea $T = \{T_1, T_2, \ldots, T_m\}$ un conjunto de números reales tales que $\forall i = 1..m-1, T_i \geq T_{i+1}$. Con este propósito, inicialmente el conjunto de datos es dividido en n conjuntos disjuntos y cada conjunto es asignado a un nodo. En cada nodo se ejecuta primero una función *Map* encargada de calcular la semejanza de cada objeto contenido en dicho nodo, respecto al conjunto de datos iniciales. Si la semejanza calculada es mayor o igual que T_m entonces se almacena la arista entre los dos objetos que tienen dicha semejanza. Posteriormente, se ejecuta una función Reduce encargada de filtrar las aristas duplicadas. Una vez construido el grafo que representa al conjunto de objetos, se lleva a cabo la etapa de construccion del agrupamiento. Para esto, se asume que el agrupamiento, denominado C, is inicialmente vacío y se divide el conjunto de aristas en n conjuntos disjuntos. Cada conjunto es asignado a un nodo que se encarga de encontrar los cuasi-cliques presentes en dicho conjunto de aristas. Para esto se ejecuta una función *Map* que convierte cada arista a un cuasi-clique de tamaño 2, si su peso es mayor que T_1. Posteriormente, se ejecuta una función *Reduce* que iterativamente va uniendo los cuasi-clique previamente formados, con algún grupo de C si el resultado es un cuasi-clique. Una vez terminado ese paso, se obtienen todos los cuasi-cliques formados por los nodos y se procesan para unir aquellos que cubran el mismo conjunto de objetos; el resultado de este último paso es el agrupamiento actual C. Los pasos anteriores son repetidos iterativamente para cada valor del conjunto T.

En [10] se propuso un algoritmo paralelo espectral para el procesamiento de grandes volúmenes de datos, utilizando MapReduce. Con este propósito, inicialmente el conjunto de datos es dividido entre los n nodos existentes. Cada nodo calcula en su función *Map* la semejanza entre los objetos asignados y aquellos objetos del conjunto de datos, que tengan un índice superior al del objeto; con esto se evita repetir cálculos de semejanza innecesarios. Posteriormente, en su función *Reduce*, se forma la matriz de semejanza. Una vez formada esta matriz, se calculan los k vectores propios de la misma, utilizando una versión paralela del algoritmo de lanczos [11]. Una vez se tienen los vectores propios de la matriz de semejanza, se ejecuta una variante paralela de Kmeans sobre la matriz formada con los vectores propios, para obtener el agrupamiento. Para esto, se seleccionan k filas que representan a los centroides de los grupos y se distribuyen las filas de la matriz entre los n nodos existentes. Cada nodo en su función *Map* se encarga de calcular la distancia de cada objeto (i.e., fila de la matriz) a los centroides calculados y determinar su centroide más cercano. Posteriormente, en la función *Reduce* se calculan los nuevos centroides con base

en las asignaciones realizadas anteriormente. Los pasos anteriores se repten hasta alcanzar la condición de parada.

En [12] se propuso un algoritmo llamado CPCluster, basado en esta tecnología. Inicialmente, el conjunto de datos se divide entre los n nodos existentes y cada nodo calcula un sub-conjunto inicial de grupos. Para esto, el nodo ejecuta una función *Map* que selecciona un elemento y forma un grupo con aquellos objetos que estén a una distancia menor que un umbral s_1 previamente definido. Posteriormente, en la función *Reduce* se eliminan del conjunto de objetos asignados al nodo, aquellos objetos que estén del objeto que forma el grupo, a una distancia menor que un segundo umbral s_2; $s_1 > s_2$. Estos dos pasos anteriores se repiten hasta procesar todos los objetos asignados. Cuando se termina este proceso, los centroides de los grupos hallados se toman como centroides iniciales del agrupamiento y se divide el conjunto de datos entre los n nodos existentes. A partir de este punto, cada nodo en su función *Map* se encarga de calcular para cada objeto su centroide más cercano y en la función *Reduce*, se encarga de calcular los nuevos centroides con base en las asignaciones realizadas previamente. Estos dos últimos pasos se repiten iterativamente hasta alcanzar la condición de parada.

La principal limitación de los algoritmos descritos en esta sección es que casi todos se basan en el cálculo de centroides o medoides, por lo que descubren grupos con forma esférica. Lo anterior se traduce en que cuando los objetos no sigan esta distribución, estos algoritmos no serán capaces de formar agrupamientos eficaces. Adicionalmente, la mayoría de los algoritmos necesita conocer a priori el número de grupos a formar y depende de 2 o más parámetros lo cual dificulta la aplicación de los mismos en problemas reales. Los experimentos en que fueron evaluados estos algoritmos incluyeron colecciones de diferentes contextos tales como Bioinformática, Computación en la nube, weblogs y censos. No obstante, es importante destacar que muy pocos de estos algoritmos fueron probados con colecciones de datos realmente cercanas a lo que podría llamarse *Big data*: hubo una colección de un poco más de 9 millones, tres colecciones de 5 millones de objetos y una de 2 millones y medio de objetos.

2.2. Basados en Kmeans

En esta sección se describen aquellos algoritmos paralelos que extienden el algoritmo Kmeans con el objetivo de ser capaces de procesar eficientemente grandes volúmenes de datos.

Velivela y Naik proponen una extension de enhanced Kmeans en la cual introducen el cómputo paralelo utilizando la librería OpenMP [13]. A diferencia del Kmeans clásico en el cual los centroides iniciales son calculados aleatoriamente, el enhanced kmeans ordena los objetos de acuerdo a su *rango*; el rango de un objeto se calcula como la diferencia entre el máximo y mínimo valor que los describe. Luego, se divide el conjunto resultando en k conjuntos y se calcula la media de cada conjunto, las cuales se toman como los centroides iniciales; a partir de este punto el funcionamiento es el mismo que el Kmeans original. Velivela y Naik identificaron el ciclo más costoso computacionalmente y utilizaron las directivas de Open MP para paralelizar dicho ciclo. La complejidad del algoritmo se mantiene inalterable, pero una vez el conjunto de datos a procesar crece rápidamente, el algoritmo resultante funciona eficientemente.

En [14] se propuso el algoritmo MKmeans el cual utiliza MPI como modelo para el paso de mensajes en el cómputo paralelo. MKmeans utiliza paralelismo de datos y con este propósito, divide inicalmente el conjunto de datos entre todos n procesos y cada proceso construye un conjunto de k grupos, siguiendo la estrategia clásica del algoritmo Kmeans. Posteriormente, cuando todos los procesos han terminado, los n conjuntos de k centroides son mezclados utilizando una estrategia voraz, con el objetivo de encontrar el conjunto final de k grupos.

En [15] se propone una versión paralela, basada en GPU, del algoritmo Kmeans, utilizando la librería CUDA. En este trabajo se identifica la parte de asignación de los objetos a los centroides como la más

costosa y en tal sentido, se decide calcular en paralelo el centroide más cercano a cada objeto del conjunto de datos. Con este objetivo, a cada hilo del GPU se le asigna un objeto y se calcula la distancia de este objeto a cada centroide, determinándose entonces el más cercano. El resto de los pasos no se paralelizan sino que se realizan en CPU, siguiendo la estrategia original de Kmeans.

Zhao *et al.* proponen en [16] una variante paralela de Kmeans, llamada PKmeans, que utiliza MapReduce. PKmeans, comienza seleccionando aleatoriamente k centroides del conjunto de datos y dividiendo dicho conjunto en n subconjuntos disjuntos; cada uno de estos subconjuntos se asigna a un nodo de la plataforma. En cada nodo se ejecuta una función *Map* que se encarga de calcular en paralelo, la distancia de cada objeto a los centroides y consecuentemente, determinar el centroide más cercano a dicho objeto. Con el resultado de este calculo se crea para cada objeto i un par $< key_i, value_i >$ donde key_i es el índice del centroide más cercano al objeto y $value_i$ es una representación del objeto i. Utilizando estos valores, posteriormente se ejecuta una función *combine*, en la que se calcula la suma parcial de los objetos asignados al mismo centroide. Finalmente, se ejecuta una función *Reduce* en la que se calcula los nuevos centroide, utilizando las sumas parciales calculadas anteriormente. Estos pasos se repiten hasta alcanzar la condición de parada.

En [17] se introduce otra variante paralela de Kmeans que utiliza MapReduce. Esta variante sigue la misma estrategia de Kmeans aunque propone dos modificaciones con el objetivo de acelerar el algoritmo. La primera es el uso alternativo de la distancia de Manhattan o la euclideana, en dependencia de un valor calculado a partir de las características de los objetos y los centroides. La segunda y última modificación es la de no seleccionar los centroides aleatoriamente, sino seleccionar los k objetos más lejanos entre sí.

Otra variante paralela de Kmeans fue propuesta en [18] para la identificación de eco-regiones geográficas con semejantes características, utilizando la interfaz de transmisión de mensajes MPI. En esta variante, inicialmente se seleccionan k centroides aleatoriamente y se divide el conjunto de datos entre los procesos existentes. Posteriormente, en cada proceso se calcula la distancia de cada uno de los objetos asignados a los centroides y se determina el centroide más cercano a cada objeto. En este paso se utiliza la desigualdad triangular para evitar cálculos innecesarios y consecuentemente, acelerar el proceso de asignación. Una vez se determina el centroide más cercano de cada objeto, esta información se transmite al resto de los procesos con el objetivo de determinar los nuevos centroides de los grupos. En este último paso, si producto de las nuevas asignaciones aparece un grupo vacío, entonces se selecciona del grupo con la mayor distancia promedio, al elemento más alejado del centroide y se coloca dicho elemento en el grupo vacío; este elemento pasa a ser el nuevo centroide de dicho grupo. Los pasos anteriores se repiten hasta alcanzar la condiciones de parada.

En [19] se propone una variante paralela del algoritmo *Two Phase Kmeans* (TPKM) [20], llamada Par2PK-means, utilizando MapReduce. Esta variante procesa el conjunto de datos incrementalmente en segmentos de un tamaño predefinido. Con este propósito cada nodo existente lee un fragmento de datos y ejecuta una función *Map* en la cual se utiliza la estrategia original de Kmeans, para agrupar los objetos contenidos en el segmento leído, en un número k' de grupos. Cada vez que se termina la función *Map*, se ejecuta una función *Reduce* en la que se va almacenando la suma parcial de los objetos contenidos en cada uno de los k' grupos formados por el nodo y a continuación, se lee otro segmento de datos y se vuelve a ejecutar la función *Map*. Una vez todo el conjunto de datos fue procesado. Se utilizan los k' grupos formados en cada nodo como una representación más compacta del conjunto de datos y con ellos se construye un conjunto de k grupos utilizando la estrategia original de Kmeans. Los pasos anteriores son ejecutados iterativamente hasta alcanzar la condición de parada.

La principal limitación de estos algoritmos es que solo pueden procesar objetos descritos por atributos numéricos. Por otra parte, necesitan conocer a priori el número de grupos a formar, por lo cual su aplicación en problemas reales puede verse limitada. Por otra parte y no menos importante, varios de los

algoritmos descritos en esta sección tienen una carga alta de traspaso de mensajes entre los nodos, lo cual puede ralentizar su funcionamiento. Es importante señalar que solo tres de los métodos descritos en esta sección fueron probados con colecciones realmente cercanas al *Big data*. Estas colecciones usadas eran de contextos geográficos o sintéticas: hubo una colección de más de mil millones de objetos, una de más de 29 millones de objetos y dos de más de 7 millones de objetos.

2.3. Jerárquicos

En esta sección se describen aquellos algoritmos paralelos que extienden algoritmos jerárquicos para el procesamiento de grandes colecciones de datos.

En [21] se propone un algoritmo paralelo y jerárquico, llamado PARC, diseñado para un clúster de computadoras. Con este objetivo, inicialmente el conjunto de datos se divide en n conjuntos disjuntos y cada uno de estos es asignado a los n nodos o computadoras existentes. A partir de este punto y en paralelo, cada nodo ejecuta una variante del algoritmo BIRCH para obtener un agrupamiento del conjunto de objetos asignados al nodo. BIRCH procesa los objetos uno a uno y los inserta en una estructura de datos en forma de árbol llamada *CF-Tree*, a partir de la cual se extrae el agrupamiento. Una vez cada nodo procesa un conjunto predeterminado de objetos se detiene este procesamiento y cada nodo envía al nodo principal el agrupamiento formado. El nodo principal se encarga de procesar todos los conjuntos de grupos, uniendo iterativamente aquellos cuya semejanza exceda un umbral. Posteriormente se envía el agrupamiento resultante del paso anterior a cada nodo y se toma este como punto de partida para procesar al resto de los elementos, utilizando la variante de BIRCH. Al terminar de procesar todos los elementos del nodo, cada nodo devuelve otra vez el agrupamiento resultando al nodo principal el cual obtiene el agrupamiento final mezclando iterativamente los grupos cuya semejanza exceda un umbral predefinido.

Otro algoritmo bajo esta categoría fue propuesto en [22] para extender el conocido algoritmo jerárquico Single-Link para procesar grandes volúmenes de datos. El algoritmo propuesto, llamado PINK, comienza dividiendo el conjunto de datos en k conjuntos disjuntos. Posteriormente, para todo par de subconjuntos, se forman el sub-grafo completo y bipartito que lo representa. Adicionalmente, se forma el sub-grafo completo que representa a cada conjunto. El conjunto de sub-grafos es distribuido entre los nodos existentes, los cuales se encargan de extraer de cada sub-grafo el árbol de costo mínimo que existe en el mismo. Posteriormente, cada nodo devuelve el árbol construido al nodo principal el cual se encarga de calcular en paralelo e iterativamente, la unión de todos los arboles formados, hasta formar el árbol o dendrograma final.

En [23] se propuso un algoritmo jerárquico, basado en el modelo computacional EREW, para el procesamiento de grandes conjuntos de datos. Inicialmente, se calcula la matriz de adyacencia de forma paralela. Para esto cada procesador existente se encarga de calcular la distancia entre un par de objetos del conjunto de datos. Posteriormente, se utiliza la distancia calculada y el algoritmo paralelo [24] para calcular el árbol de costo mínimo que representa al conjunto de datos. Finalmente, en cada procesador se elimina de forma paralela las $q-1$ aristas del árbol de costo mínimo, con mayor costo y el conjunto de componentes conexas resultante representa al agrupamiento final.

Otro algoritmo paralelo para grandes volúmenes de datos fue propuesto en [25]. Inicialmente se divide el conjunto de datos en el número m de procesadores existente y cada procesador se encarga de calcular concurrentemente la distancia entre cada uno de sus objetos asignados y el resto de los objetos del conjunto de datos. Una vez construida la matriz de distancia, cada proceso construye un conjunto de grupos disjuntos. El conjunto de grupos formado por un proceso es compartido con el proceso vecino y es mezclado con el construido por ese proceso. Luego, el nodo principal se encarga de construir un agrupamiento disjunto a partir de la combinación de los agrupamiento construidos por cada proceso. Este conjunto de grupos es

distribuido al resto de los procesos quienes se encargan entonces de agruparlos en forma jerárquica. Estas jerarquías son luego iterativamente mezcladas por el nodo principal hasta formar la jerarquía final.

La principal limitación de la mayoría de los algoritmos descritos en esta sección es que tienden a construir jerarquías muy grandes que resultan difíciles de utilizar en problemas reales. Hay dos algoritmos que utilizan una estructura de datos en forma arbórea para almacenar los elementos, lo cual los hace altos consumidores de memoria. Por otra parte, la mayoría de los algoritmos depende de dos o más parámetros lo cual puede limitar su aplicación en problemas reales. Las colecciones utilizados en los experimentos en su mayoría provienen de la Astronomía y el Censo de los EUA. En el primer caso, se utilizaron tres colecciones de más de 5 millones de datos y una con más de tres millones de datos, mientras que en el segundo caso la colección contaba con más de dos millones de datos.

2.4. Otras técnicas

En esta sección se describen aquellos algoritmos paralelos que siguen enfoques diferentes a los explicados en las secciones anteriores.

En [26] se propuso un algoritmo paralelo basado en el cálculo de conjuntos frecuentes, llamado SDC, para el procesamiento eficiente de colecciones de documentos muy grandes. SDC supone que se cuenta con un conjunto de nodos (CPUs en un *clúster* de computadoras) y que el conjunto de datos es accesible por cada uno de los nodos. Inicialmente, se calcula el conjunto de conjuntos frecuentes de ítems, utilizando un algoritmo desarrollado por los autores [27]. Cada conjunto frecuente calculado determina un grupo formado por todos aquellos objetos (i.e., documentos) que contienen a dicho conjunto frecuente. Posteriormente, el conjunto de datos es dividido en n conjuntos disjuntos y asignados a cada uno de los nodos existentes. Cada nodo se encarga de procesar aquellos objetos asignados que pertenezcan a más de un grupo, con el objetivo de asignarlos a un solo grupo; para esto se utiliza un algoritmo de clasificación KNN. Una vez cada nodo termina su proceso, el nodo principal se encarga de mezclar los resultados determinados por cada nodo y así dar el agrupamiento final.

En [28] se propuso una extensión paralela de un algoritmo de bi-agrupamiento, para el procesamiento de grandes conjuntos de datos. Con este objetivo, inicialmente se divide el conjunto de datos entre los N nodos existentes. En cada nodo, se normaliza el conjunto de datos y se calcula la distribución conjunta de las columnas y filas que representan a los objetos asignados al proceso. Posteriormente, en cada nodo se lleva a cabo un re-ordenamiento iterativo, tanto de filas como de columnas, tomando en cuenta el valor de la distribución conjunta calculada previamente y una función predefinida. Una vez se termina la operación anterior, cada nodo determina el conjunto de bi-grupos presentes en el conjunto de datos existentes. Para esto, se realiza un proceso que busca aquellas sub-matrices que optimicen una función de coherencia predefinida con anterioridad. Los bi-grupos determinados por cada nodo se devuelven al nodo central que se encara de procesarlos uniendo aquellos que sean muy semejantes. El conjunto resultante es el bi-agrupamiento final.

En [29] se propuso un algoritmo paralelo para procesar grandes conjuntos de datos, combinando una estrategia de divide y vencerás con una optimización multi-objetivo. Con este propósito, el conjunto de datos se divide en N partes y cada parte es asignada a un procesador. Cada procesador se encarga de formar un conjunto de grupos a partir del conjunto de objetos asignados, utilizando un algoritmo de agrupamiento multi-objetivo. El algoritmo utilizado es basado en el cálculo de centroides y utiliza una estrategia evolutiva similar a la de la estrategia evolutiva NSGA [30]. Una vez determinado los grupos, cada procesador devuelve al nodo principal los centroides de los grupos determinados. Si el número total de centroides es muy grande, entonces este se divide entre los N procesadores y es agrupado siguiendo la misma estrate-

gia de agrupamiento evolutiva y multi-objetivo descrita anteriormente. Estos pasos se repiten mientras el tamaño de la solución sea grande.

En [31] se propuso un algoritmo paralelo espectral para el procesamiento de grandes volúmenes de datos, utilizando MPI. Con este propósito, inicialmente el conjunto de datos es dividido entre las n computadoras existentes. Cada computadora calcula en paralelo la semejanza entre los objetos asignados y los objetos del conjunto de datos. Una vez calculada la matriz de semejanza se utiliza un algoritmo paralelo llamado PARPACK [32], basado en el método de Arnoldi, para hallar los k vectores propios de la matriz calculada. Una vez se tienen los vectores propios de la matriz de semejanza, se ejecuta una variante paralela de Kmeans sobre la matriz formada con los vectores propios, para obtener el agrupamiento. Con este propósito, se seleccionan k filas que representan a los centroides de los grupos y se distribuyen las filas de la matriz entre las n computadoras existentes. Cada computadora se encarga de calcular en paralelo la distancia de cada objeto (i.e., fila de la matriz) a los centroides y determinar su centroide más cercano. Posteriormente, estos resultados son devueltos al nodo principal que se encarga de calcular los nuevos centroides con base en las asignaciones realizadas anteriormente. Los pasos anteriores se repten hasta alcanzar la condición de parada.

Una extension del algoritmo PCM [33], para el procesamiento de grandes volúmenes de datos, fue propuesto en [34], utilizando MapReduce. Inicialmente, se seleccionan k prototipos a partir de los objetos del conjunto de datos. Este mismo conjunto se divide entre los n nodos existentes, los cuales en su función *Map*, se encargan de calcular tanto la distancia como el grado de pertenencia que tienen los objetos asignados al nodo, a los grupos determinados por los prototipos actuales, de forma que se minimice la función objetivo empleada por el algoritmo PCM. Posteriormente, en la función *Reduce*, se toman las asignaciones realizadas en el paso anterior y se actualiza el conjunto de prototipos y el valor de la función objetivo, utilizando también el grado de pertenencia de los objetos a los grupos. Estos pasos anteriores, se repiten hasta alcanzar la condición de parada. Este algoritmo fue aplicado en [35] para el agrupamiento de datos de sensores.

En [36] se propuso una extension paralela del clásico algoritmo EM sobre modelos de mezclas gaussianas, aunque se puede aplicar en otras variantes de EM. Inicialmente, se selecciona una muestra del conjunto de datos y se ejecuta el algoritmo Rnd-EM [37] y se evalúa el resultado; este paso se repite 100 veces. Posteriormente, se inicializan los parámetros estadísticos del algoritmo EM utilizando los mejores valores detectados sobre las muestras y se divide el conjunto de datos entre los n nodos existentes. Cada nodo realiza entonces el paso E donde se estima la esperanza de verosimilitud de los parámetros y el paso M, donde se trata de maximizar la esperanza de verosimilitud. Los valores calculados son recogidos por el nodo principal que se encarga de realizar una suma de dichos valores y pasar el resultado a todos los nodos. Los pasos anteriores se repiten hasta alcanzar la condición de parada.

En [38] se propuso un algoritmo paralelo basado en la construcción de arboles de costo mínimo (MST, por sus siglas en inglés), utilizando MPI. Inicialmente, se divide el conjunto de objetos entre los n nodos existentes, los cuales se encargan de calcular la distancia de cada objeto asignado al resto de los objetos del conjunto. Las distancias calculadas son pasadas al nodo principal que se encarga de construir el grafo que representa al conjunto de objetos. Este grafo es posteriormente dividido en n sub-grafos, los cuales se utilizan para crear todos los grafos bipartitos que existen entre los pares de sub-grafos determinados. Una vez determinados estos grafos bipartitos, se calculan en paralelo los árboles de costo mínimo de cada grafo bipartito y de cada sub-grafo. Estos árboles son posteriormente iterativamente mezclados por el nodo principal para formar el árbol de costo mínimo que representa al conjunto de objetos. Una vez determinado este árbol, se construye una representación lineal [39] del mismo y se procesa con el algoritmo publicado en [40], para descubrir los grupos existentes en los datos.

En [41] se propuso una extension paralela sobre MPI del algoritmo WaweCluster, basado en transformadas de Wawelets. Inicialmente, el conjunto de dato se divide entre los *n* nodos existentes. Cada nodo se encarga de aplicar la transformada de wavelet sobre la representación de los objetos asignados. Una vez transformado el espacio de representación de los objetos, el nodo se encarga de construir las componentes conexas presentes en esa matriz transformada y de asignar índices a los objetos, de acuerdo a la componente a la que pertenezcan. Estas componentes conexas son devueltas al nodo principal, el cual se encarga de procesarlas y mezclar iterativamente aquellas que compartan objetos.

La principal limitación de la mayoría de los algoritmos descritos en esta sección es que solo pueden procesar objetos descritos por números. Adicionalmente, varios de los algoritmos son basados en centroides o conceptos similares, por lo que solo descubren grupos con forma esférica. La mayoría de los algoritmos depende de dos parámetros o más, por lo que su uso en problemas reales puede ser difícil. Varios de los algoritmos son basados en grafos lo cual los puede hacer altos consumidores de memoria. Otros algoritmos incluyen operaciones muy costosas desde el punto de vista numérico que los pueden llevar a errores en la formación de los grupos. Por otra parte, la mayoría de las versiones paralelas llevan consigo una gran cantidad de traspaso de mensajes entre los nodos o procesadores, lo cual puede ralentizar a los algoritmos resultantes cuando el volumen de información crece. Es importante mencionar que la mayoría de los algoritmos no se evaluó en colecciones cuyos tamaños pudieran acercarse a lo que es el *Big Data*. Las colecciones más grandes utilizadas en los experimentos eran sintéticas o provenían de sensores de gas o eran extensiones a colecciones del repositorio UCI. Varias de estas colecciones se acercan a lo que se puede denominar *Big data*: hubo una colección de 20 millones de objetos, otra de 25 millones de objetos, así como otras que superaron los mil millones e incluso cientos de millones de objetos. No obstante, es válido mencionar que en todas estas colecciones los objetos se describían por un número bien pequeño de variables.

3. Conclusiones

El agrupamiento es una de las técnicas más utilizadas para el descubrimiento de información a partir de los datos. No obstante, el rápido crecimiento en las posibilidades de almacenar y producir datos que se ha evidenciado en los últimos años ha dejado obsoletos a muchos de los algoritmos más importantes de los últimos años. Una de las técnicas utilizadas con frecuencia para agilizar tareas de cómputo es el procesamiento paralelo y/o distribuido.

En este trabajo se hace una revisión crítica de los algoritmos paralelos más importantes reportados en los últimos años para el procesamiento de grandes colecciones de objeto e incluso, para el procesamiento de los denominados *Big data*. Se señalan las principales limitaciones se los algoritmos analizados y se resaltan además, cuáles han sido los contextos en los que se han evaluado y los mayores tamaños de las colecciones utilizadas en dichos experimentos. A partir de lo analizado en la sección2 se puede decir que el desarrollo de algoritmos de agrupamiento para el procesamiento eficiente de grandes conjuntos de datos sigue siendo un problema abierto.

En particular, se pueden enunciar algunas áreas de interés científico que serían interesantes de desarrollar o profundizar. Una de las áreas es el desarrollo de algoritmos de agrupamiento que sean capaces de procesar objetos descritos por rasgos mezclados e incluso incompletos y que sean capaces de procesar colecciones catalogadas por *Big Data*. La mayoría de los algoritmos reportados, asumen que los objetos se describen por un vector numérico o ignoran las partes problemáticas de las descripciones de los objetos. Por otra parte, otra línea poco explorada es la del desarrollo de algoritmos incrementales y/o dinámicos que puedan ser capaces de procesar colecciones catalogadas por *Big Data* y actualizar eficientemente el

conjunto de grupos cuando la colección cambia. De igual forma el problema del agrupamiento con traslape y difuso no ha sido abordado en este contexto, por lo que sería útil desarrollar algoritmos con estas características.

Referencias bibliográficas

1. Shirkhorshidi, A.S., Aghabozorgi, S., Wah, T.Y., Herawan, T.: Big Data Clustering: A Review. In: Proceedings of ICCSA 2014. (2014) 707–720
2. Havens, T.C., Bezdek, J.C., Palaniswami, M.: Scalable single linkage hierarchical clustering for big data. In: Proceedings of the Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing. (2013) 396–401
3. Pfitzner, D., Leibbrandt, R., Powers, D.: Characterization and Evaluation of Similarity Measures for Pairs of Clusterings. Knowledge and Information Systems **19**(3) (2009) 361–394
4. Martínez-Trinidad, J.F., Guzman-Arenas, A.: The Logical Combinatorial Approach to Pattern Recognition An Overview Through Selected Works. Pattern Recogn. **34**(4) (2001) 1–11
5. Hathaway, R., Bezdek, J.: Extending fuzzy and probabilistic clustering to very large data sets. Comput. Stat. Data Anal. **51**(1) (2006) 215–234
6. Zhanquan, S.: A parallel clustering method study based on mapreduce. In: Proceedings of the International Workshop on Cloud Computing and Information Security. (2013) 416–419
7. Jakovits, P., Srirama, S.N.: Clustering on the cloud: Reducing clara to mapreduce. In: Proceedings of NordiCloud 2013. (2013) 64–71
8. Kaufman, L., Rousseeuw, P.: Finding Groups in Data An Introduction to Cluster Analysis. Wiley Interscience (1990)
9. Yang, X., Zola, J., Aluru, S.: Large scale metagenomic sequence clustering on mapreduce clusters. Journal of Bioinformatics and Computational Biology **11**(1) (2013)
10. Jin, R., Kou, C., Liu, R., Li, Y.: Efficient parallel spectral clustering algorithm design for large data sets under cloud computing environment. Journal of Cloud Computing: Advances, Systems and Applications **2**(18) (2013)
11. J, J.C., Willboughby, R.A.: Lanczos Algorithms for Large Symmetric Eigenvalue Computations. Birkhauser Boston Inc (1985)
12. Li, H., Yang, D., Fang, W.: Parallel based on cloud computing to achieve large data sets clustering. In: Proceedings of the 2012 International Conference on Computer Science and Electronics Engineering. (2012) 411–415
13. Velivela, G., Naik, D.S.B.: A novel approach towards parallel k-means. International Journal of Computer Engineering and Science **3**(2) (2013) 81–86
14. Zhang, J., Wu, G., Hu, X., Li, S., Hao, S.: A parallel clustering algorithm with mpi mkmeans. JOURNAL OF COMPUTERS **8**(1) (2013) 10–17
15. Farivar, R., Rebolledo, D., Chan, E., Campbell, R.H.: A parallel implementation of k-means clustering on gpus. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. (2008) 340–345
16. Zhao, W., Ma, H., He, Q.: Parallelk-means clustering based on mapreduce. In: Proceedings of CloudCom 2009. (2009) 674–679
17. Liao, Q., Yang, F., Zhao, J.: An improved parallel kmeans clustering algorithm with mapreduce. In: Proceedings of ICCT2013. (2013) 764–768
18. Kumar, J., Mills, R.T., Hoffmana, F.M., Hargrove, W.W.: Parallel k-means clustering for quantitative ecoregion delineation using large data sets. In: Proceedings of the International Conference on Computational Science, ICCS 2011. (2011) 1602–1611
19. Nguyen, C.D., Nguyen, D.T., Pham, V.H.: Parallel two-phase k-means. In: Proceedings of ICCSA 2013. (2013) 224–231
20. Pham, D.T., Dimov, S.S., Nguyen, C.D.: A two phase kmeans algorithm for large datasets. Journal of Mechanical Engineering Science **218**(10) (2004) 1269–1273
21. Feng, Z., Zhou, B., Shen, J.: A parallel hierarchical clustering algorithm for pcs cluster system. Neurocomputing **70**(4-6) (2007) 809–818
22. Hendrix, W., Palsetia, D., Patwary, M.M.A., Agrawal, A.: A scalable algorithm for single-linkage hierarchical clustering on distributed-memory architectures. In: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization. (2013) 7–13
23. Li, Z., Li, K., Xiao, D., Yang, L.: An adaptive parallel hierarchical clustering algorithm. In: Proceedings of HPCC 2007. (2007) 97–107
24. Akl, S.G.: An adaptive and cost optimal parallel algorithm for mst. Computing **3** (1986)
25. Tang, C.H., Tsai, M.F., Chuang, S.H., Cheng, J.J., Wang, W.J.: Shortest-linkage-based parallel hierarchical clustering on main-belt moving objects of the solar system. Journal of Bioinformatics and Computational Biology **34** (2014) 26–46

26. Wang, Y., Jia, Y., Yang, S.: Short documents clustering in very large text databases. In: Proceedings of WISE 2006 Workshop. (2006) 83–93
27. Wang, Y., Jia, Y., Yang, S.: Parallel mining of top-k frequent itemsets in very large text database. In: Proceedings of WAIM 2005. (2005) 706–712
28. Nisar, A., Ahmad, W., Liao, W.K., Choudhary, A.: High performance parallel/distributed biclustering using barycenter heuristic. In: Proceedings of SIAM International Conference on Data Mining. (2009) 1050–1061
29. Ozyer, T., Alhajj, R.: Parallel clustering of high dimensional data by integrating multi-objective genetic algorithm with divide and conquer. Appl Intell **31** (2009) 318–331
30. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput **6**(2) (2002) 182–197
31. Song, Y., Chen, W.Y., Bai, H., Lin, C.J., Chang, E.Y.: Parallel spectral clustering. In: Proceedings of ECML PKDD 2008. (2008) 374–389
32. Maschho, K., Sorensen, D.: A portable implementation of arpack for distributed memory parallel architectures. In: Proceedings of Copper Mountain Conference on Iterative Methods. (1996)
33. Krishnapuram, R., Keller, J.M.: A possibilistic approach to clustering. IEEE Transactions on Fuzzy Systems **1**(2) (1993) 98–110
34. Zhang, Q., Chen, Z.: A weighted kernel possibilistic c-means algorithm based on cloud computing for clustering big data. INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS **27** (2014) 1378–1391
35. Zhang, Q., Chen, Z.: A distributed weighted possibilistic c-means algorithm for clustering incomplete big sensor data. International Journal of Distributed Sensor Networks (2014)
36. Chen, W.C., Ostrouchov, G., Pugmire, D., Wehner, P., Wehner, M.: A parallel em algorithm for model-based clustering applied to the exploration of large spatio-temporal data. Technometrics **55**(4) (2013) 513–523
37. Maitra, R.: Initializing partition-optimization algorithms. IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**(2) (2009) 144–157
38. Olman, V., Mao, F., Wu., H., Xu, Y.: Parallel clustering algorithm for large data sets with applications in bioinformatics. IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**(2) (2009) 344–352
39. Xu, Y., Olman, V., Xu, D.: Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning tree. Bioinformatics **18**(4) (2001) 526–535
40. Olman, V., Xu, D., Xu, Y.: Cubic: Identification of regulatory binding sites through data clustering. Journal of Bioinformatics and Computational Biology **1**(1) (2003) 21–40
41. Yildirim, A.A., Ozdogan, C.: Parallel wavecluster: A linear scaling parallel clustering algorithm implementation with application to very large datasets. Journal of Parallel Distrib. Comput. **71** (2011) 955–962

RT_032, febrero 2016
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. A No. 21406 e/214 y 216, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba