Clustering de un flujo de datos usando MapReduce

María José Basgall [1,2], Waldo Hasperué [2], Cesar Estrebou [2], Marcelo Naiouf [2]
{mjbasgall, whasperue, cesarest, mnaiouf}@lidi.info.unlp.edu.ar

[1] UNLP, CONICET, III-LIDI, La Plata, Argentina
[2] Instituto de Investigación en Informática (III-LIDI)
Facultad de Informática - Universidad Nacional de La Plata

Resumen Las técnicas de agrupamiento (*clustering*) sobre flujo de datos (*data stream*) son una poderosa herramienta para determinar las características que tienen en común los datos provenientes del flujo. Para obtener buenos resultados es necesario almacenar gran parte de éste en una ventana temporal. En este artículo medimos una técnica que maneja el tamaño de la ventana temporal de manera dinámica utilizando un algoritmo de clustering implementado en el framework MapReduce. Los resultados obtenidos demuestran que esta técnica alcanza una ventana de gran tamaño logrando así que cada dato del flujo se utilice en más de una iteración del algoritmo de clustering permitiendo conseguir resultados similares independientemente de la velocidad de los datos del flujo. Los centroides resultantes de cada flujo de datos son semejantes a los que se consiguen haciendo un clustering sobre el conjunto de datos completo.

Keywords: Big Data, Stream Processing, MapReduce, Clustering

1. Introducción

El procesamiento de flujos de datos (*stream processing*) es un área muy estudiada en los últimos años. Stream processing permite llevar a cabo tareas mediante el análisis de un flujo continuo y potencialmente infinito de datos [1][2][3][4][5][6][7][8][9].

El objetivo de stream processing es permitir que las tareas analicen los flujos de datos de forma online, brindando respuestas en tiempos muy cercanos al tiempo real. La principal característica de este tipo de procesamiento es que los datos del flujo llegan a una velocidad tal que no es posible almacenarlos en su totalidad y, si se pueden almacenar, el volumen de datos es tan grande que presenta la dificultad de analizarlo en tiempos de respuesta cortos [10].

Para poder realizar análisis a partir de grandes volúmenes de datos resulta sumamente útil hacer uso de tecnologías multiprocesador así como de las técnicas de paralelización de algoritmos que tomen ventaja de las mismas. Actualmente es posible encontrar diversos aportes donde se aplican conceptos de Cómputo de Altas Prestaciones (*High Performance Computing, HPC*) en el área de stream

processing; de igual manera, es posible encontrar trabajos realizados en entornos de computación en la nube (*Cloud Computing*). Estos últimos introducen un nuevo foco desde el punto de vista de la computación de altas prestaciones, ya que brindan un soporte "a medida" para la ejecución de aplicaciones sin la necesidad de adquirir el hardware [11][12][13][14].

Una de las herramientas más utilizadas para el procesamiento de grandes cantidades de datos es Hadoop MapReduce [15][16][17], construida sobre los principios bien conocidos del procesamiento paralelo y distribuido. MapReduce trabaja sobre el sistema de archivos HDFS [18], el cual ofrece eficiencia, es distribuido, brinda almacenamiento tolerante a fallos y es adecuado para aplicaciones que utilizan grandes volúmenes de datos ya que proporciona un alto rendimiento de acceso a los mismos. El framework MapReduce realiza las tareas de distribución y paralelización de manera automática y transparente al programador lo que lo convierte en una poderosa herramienta para implementar soluciones distribuidas.

Una de las tareas más comunes al realizar stream processing es la de clustering. Esta tarea consiste en agrupar los datos provenientes del flujo en diferentes grupos con la finalidad de encontrar características similares en los datos. En los últimos años se han propuesto varios trabajos que realizan clustering sobre un flujo de datos. Estos aportes se pueden dividir básicamente en dos categorías, los que no almacenan datos utilizando únicamente un conjunto de descriptores por grupo (*cluster*) y los que utilizan una pequeña ventana temporal del flujo de tamaño fijo. La principal desventaja de estos algoritmos es que al no poder trabajar con todos los datos, ya sea porque no los almacenan o solo almacenan una pequeña porción del flujo, no pueden garantizar que los grupos de datos obtenidos reflejen la verdadera distribución de los datos. Este problema puede ser disminuido almacenando la mayor cantidad de datos del flujo posible y para ello es necesario contar con algoritmos que sean capaces de ejecutarse en entornos de trabajo distribuidos y paralelos [19].

Recientemente se ha propuesto una técnica que permite el manejo de una ventana temporal dinámica del flujo utilizando el framework MapReduce [20]. Esta técnica permite almacenar una gran cantidad de datos provenientes del flujo administrando el tamaño de la ventana de manera dinámica en función de la frecuencia de llegada de los datos del flujo y del tiempo que le lleva al algoritmo que realiza el análisis de los datos en completar una etapa de su tarea.

En el presente trabajo se midió el rendimiento de la técnica presentada en [20] aplicando una tarea de clustering sobre un flujo de datos proveniente de Twitter. La tarea de clustering utilizada está basado en el algoritmo K-means [21] al cual se le realizaron las modificaciones necesarias para poder ejecutarlo en el framework Hadoop MapReduce. Se eligió el algoritmo K-means por su simplicidad en la implementación pero esta técnica puede ser aplicada a cualquier algoritmo de clustering.

Se midió la respuesta final de los centroides obtenidos con esta técnica utilizando un flujo de datos proveniente de Twitter, empleando diferentes frecuencias de llegada de los datos. Se compararon los resultados obtenidos en cada

experimento con el resultado final producto de ejecutar de manera secuencial el algoritmo K-means convencional sobre el conjunto completo de los datos del flujo recolectado.

El artículo está organizado como sigue. La sección 2 se refiere al clustering sobre un flujo de datos y los diferentes enfoques. En la sección 3 se presenta el método utilizado. En la sección 4 se muestran los experimentos realizados y los resultados obtenidos. Finalmente en la sección 5 se presentan las conclusiones y los trabajos a futuro.

2. Clustering sobre un flujo de datos

El área de minería de flujo de datos es la encargada de procesar flujos de datos con la intención de extraer conocimiento útil, en la mayoría de los casos, para la toma de decisiones. Dentro de la minería de datos el clustering es una tarea muy utilizada para descubrir cómo se relacionan los datos analizados entre sí. Cuando el volumen de datos a analizar es muy grande aparecen nuevas técnicas para realizar clustering sobre flujos de datos.

La tarea de clustering sobre flujos de datos consiste en agrupar los datos disponibles en un número de grupos preestablecido utilizando para ello alguna métrica de similitud. Los algoritmos convencionales de clustering, que trabajan sobre una base de datos, actualizan los correspondientes centroides de los clusters utilizando los vectores n-dimensionales de aquellos datos que están más cerca del propio centroide. Así, para cada dato d, se busca el centroide más cercano y se asigna a éste el dato d. Al finalizar una iteración del algoritmo, cada cluster actualiza su centroide utilizando todos los datos que le fuera asignado. Este proceso continúa un número significativo de iteraciones hasta conseguir un cierto equilibrio entre los centroides. En un escenario donde los datos vienen de manera continua este enfoque no es posible llevar a cabo ya que el volumen de información a almacenar es muy grande y potencialmente infinito. Por ello, se han propuesto diferentes variantes para poder realizar clustering sobre un flujo de datos.

La mayor parte de las técnicas propuestas para el procesamiento de flujos de datos se pueden enmarcar en dos enfoques principales. Uno de ellos son los algoritmos que, ante la imposibilidad de almacenar todos los datos y utilizarlos para calcular el nuevo centroide, hacen uso de descriptores de clusters. Los centroides se actualizan inmediatamente ante la llegada de un nuevo dato o bien ante la acumulación de alguna evidencia de que el centroide debe ser actualizado. Estos algoritmos son modelos que procesan cada dato una única vez [1][22][23][24].

El otro enfoque es utilizar ventanas temporales y deslizantes de tamaño fijo donde almacenan los n últimos datos recibidos, o los n más representativos [25][26] [27] [28][29]. La principal desventaja de estos algoritmos es que el tamaño de ventana se establece a priori y que éste depende fuertemente de la velocidad del flujo de datos. Cuando no se conoce de antemano la frecuencia de llegada de los datos o cuando ésta puede variar a lo largo del tiempo, elegir un tamaño de ventana adecuado representa un desafío.

2.1. Trabajos relacionados

En el trabajo presentado en [30] aplican la idea de microclusters, el objetivo es realizar muchos clusters de tamaño reducido para luego utilizar alguna técnica aglomerativa y determinar cómo quedan formados los clusters a partir de la información recolectada por los microclusters. En este trabajo proponen que cada microcluster solo guarde el vector centroide y un peso que mide la densidad del área o volumen del propio microcluster.

Esa misma idea es utilizada en [31] que además explota el uso del paradigma Hadoop MapReduce para llevar a cabo la tarea. Este trabajo utiliza una segunda fase para realizar un pulido de los microclusters y eliminar aquellos que ya no son relevantes. En [32] se profundiza en esta idea almacenando en cada cluster, el vector centroide, la cantidad de elementos pertenecientes al cluster, una medida de distorsión del cluster y el timestamp del último elemento incorporado en el cluster. Un trabajo de características similares se puede encontrar en [33], que además de almacenar tres descriptores por cluster también presenta un algoritmo de dos etapas, una online y otra offline de pulido de información.

Todos estos trabajos tienen el mismo problema que cada cluster sólo guarda información resumida de todos los elementos pertenecientes en él. Ello representa un gran inconveniente al momento de realizar una actualización de los clusters ya que solo poseen parte de la información y cada cluster está sesgado básicamente por un centro y un radio, sin poder guardar la topología de los datos que pertenecen a un cluster.

Por otro lado, en [34] se utiliza más información de los datos aplicando una ventana deslizante de tamaño fijo mientras que en [35] se emplea support vector machine (SVM) para realizar el clustering tratando el flujo en porciones de tamaño fijo.

Cualquier algoritmo de clustering consigue mejores resultados si cuenta con todos los datos para poder actualizar los centroides. En un entorno de tratamiento de un flujo resulta útil poder contar con un gran número de datos el mayor tiempo posible. En este trabajo se implementó el algoritmo K-means [21] el cual fue adaptado para ser ejecutado en el framework Hadoop MapReduce utilizando la técnica del manejo de ventana temporal presentado en [20].

3. Clustering sobre MapReduce

En [20] se presenta una técnica que gestiona de manera dinámica una ventana de datos recolectados de un flujo. Esta técnica permite que en cada iteración del algoritmo de clustering cuente con la mayor cantidad de datos posible. El tamaño de la ventana temporal es adaptada en función de la frecuencia del flujo, la cantidad de datos almacenados para su procesamiento y el tiempo que le lleva a la tarea de clustering lograr el resultado parcial con el lote de datos actual, manteniendo el mayor tamaño posible del flujo disponible dentro de la ventana. De esta manera, el algoritmo que realiza la tarea de clustering utiliza cada dato durante un número significativo de iteraciones.

La técnica propuesta en [20] consiste en dos tareas. La primera se basa en la captura del flujo de datos y su correspondiente almacenamiento, que se ejecuta de manera continua. El data stream es leído de forma online, y los datos recolectados se van guardando en un buffer el cual es almacenado en un archivo dentro del directorio de trabajo en el HDFS.

La segunda tarea lleva a cabo el procesamiento del algoritmo de *stream mining* (en el caso puntual de este trabajo, una tarea de clustering) la cual realiza su ejecución en el framework MapReduce utilizando como entrada todos los archivos que están dentro del directorio de trabajo en el HDFS. A medida que se colectan datos del flujo, estos son almacenados en un nuevo archivo. Cuando el tamaño total de los mismos supera un umbral, los archivos más antiguos son eliminados, generando así el corrimiento de la ventana de datos.

El funcionamiento de un trabajo MapReduce está basado en dos fases. La primera etapa (Map) consiste en asociar cada dato leído con una clave, es decir, toma un conjunto de datos de entrada y los convierte en otro conjunto de datos donde los elementos son convertidos en tuplas (par de clave/valor). Finalizada esta etapa se ejecuta la segunda fase (Reduce) donde cada proceso reductor (reducer) recibe todos los valores asociados con una misma clave para finalmente realizar la tarea propiamente dicha, la cual es específica del problema a resolver y escrita por el programador.

En este trabajo, en la etapa de MapReduce se realiza el procedimiento de una iteración del algoritmo de clustering el cual está basado en el algoritmo K-means. El número de clusters a encontrar es determinado al comienzo del proceso y los centros iniciales son seleccionados al azar.

La ejecución del trabajo MapReduce comienza cuando el buffer del stream almacena los primeros b datos del flujo. En ese momento se guardan todos los datos acumulados en un archivo dentro de un directorio de trabajo en el HDFS y se lanza su ejecución.

El conjunto de centroides C es pasado al proceso Map y éste busca para cada vector v leído cuál es el centroide cb más cercano. En la salida escribe pares clave-valor donde la clave es cb y como valor el propio vector v.

Para aprovechar al máximo la capacidad del cómputo paralelo se crean K reducers, los cuales cada uno recibe todos los vectores correspondientes a un mismo cluster y con ellos calcula el nuevo centroide del mismo, que resulta de calcular el vector promedio entre todos los vectores recibidos. Cada reducer escribe como salida el nuevo centroide calculado además de todos los vectores que pertenecen al cluster. Esto último permite tener información de cómo están conformados los clusters al finalizar una etapa MapReduce.

Cuando el trabajo MapReduce finaliza, se lee del directorio de trabajo el nuevo conjunto de centroides C. Luego se ejecuta un nuevo trabajo MapReduce para ir refinando el procedimiento de clustering y de esa manera lograr que los centroides converjan. Este proceso iterativo continúa hasta ejecutar un número máximo de iteraciones, o hasta alcanzar un equilibrio donde los centroides no cambian significativamente de una iteración a otra.

Mientras el trabajo MapReduce ejecuta el algoritmo de clustering, el proceso de streaming continúa con la captura de datos llenando un nuevo buffer. Se guarda hasta b datos de este buffer en un archivo nuevo. Esta condición existe para evitar que se escriban archivos con gran cantidad de datos y que la ventana sea reemplazada en su totalidad, logrando así un desplazamiento leve de la misma.

4. Experimentos realizados y resultados obtenidos

Para la realización de los ensayos se utilizó un flujo de datos públicos proveniente de Twitter, el cual posee 778072 tweets en español recolectados durante el desarrollo de la Copa América Centenario el cual se llevó a cabo del 3 al 26 de junio del 2016 en Estados Unidos. La captura de los datos se realizó desde el 30 de mayo al 10 de julio del 2016, utilizando la librería de Java Twitter4J [36] para acceder a los datos mediante la API de Twitter Streaming [37]. Estos tweets han sido procesados para remover URLs, emojis, imágenes y símbolos de puntuación.

Los textos de los tweets fueron procesados para convertirlos en vectores binarios de 95 dimensiones. Debido a que los tweets tienen una longitud muy reducida (140 caracteres como máximo), cada elemento del vector representa la ausencia o presencia de una palabra en particular. En todos los ensayos se utilizó la distancia del coseno (ecuacion 1) como medida de similitud para la ejecución del algoritmo K-means ya que es la más utilizada en estos tipos de problemas [38]. De manera arbitraria se eligió un K=5 para realizar el clustering.

$$cosine\_dist(x, y) = 1 - \frac{x \cdot y}{|x|.|y|} \tag{1}$$

La figura 1 muestra la frecuencia de llegada de los tweets. Dado que el flujo de datos recolectado duró 42 días, para llevar a cabo los experimentos se transformó el data stream reduciendo la duración entre el primer y el último tweet recolectado, cambiando sólo la duración de todo el flujo, manteniendo la frecuencia relativa de llegada de los tweets. Se realizaron cuatro modificaciones, las que se denominaron como stream50, stream100, stream250 y stream500, la duración final de cada flujo es de aproximadamente 20, 10, 4 y 2 horas, respectivamente. La cantidad de tweets manejados por el algoritmo en cada una de sus iteraciones puede verse en la figura 2.

Las pruebas se realizaron tomando una configuración de los parámetros mencionados en [20] obtenida de manera empírica a partir de la combinación de los mismos que mejor se adapta al flujo de datos analizados, teniendo en cuenta el número de etapas interrumpidas del algoritmo y los valores dado por el coeficiente de renovación (r) que se menciona en dicho trabajo. Esta configuración está dada por un tamaño de buffer inicial de 1000 tweets, un factor que determina el tamaño inicial de la ventana de 50, un factor de extensión del tamaño del buffer de 0.5, un umbral de determinación de convergencia de 0.01 y el número de iteraciones máximas por cada etapa de 4.

Se realizó la comparación de los centroides finales que resultaron de la ejecución del algoritmo de clustering luego de procesar cada uno de los flujos. Se

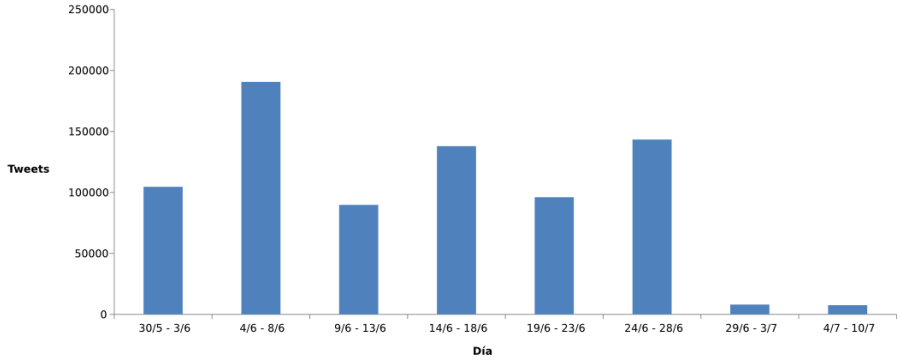
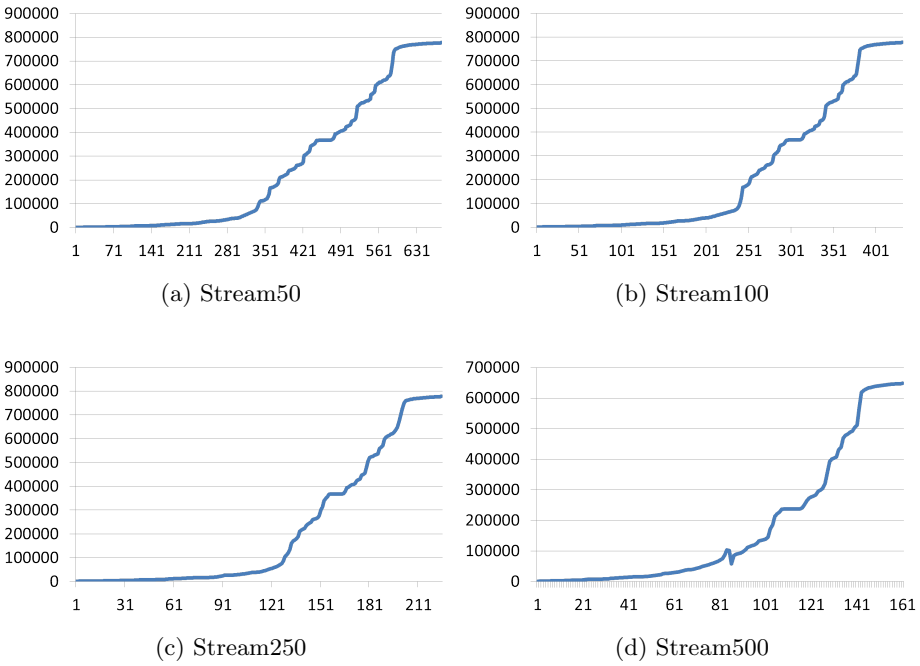


Figura 1: Frecuencia de llegada de los *tweets*



(a) Stream50

(b) Stream100

(c) Stream250

(d) Stream500

Figura 2: Cantidad de tweets manejados por el algoritmo en cada una de sus iteraciones

cotejaron los centroides finales de cada flujo todos entre sí, y también se hizo la comparación de cada uno de los flujos con el resultado obtenido por el algoritmo K-means secuencial (*Km-sec*) donde se utilizó todo el conjunto de datos en cada una de las iteraciones. La comparación entre los clusters de dos flujos se realizó midiendo la distancia coseno entre el centroide resultado de un flujo y el

centroide del segundo flujo. Esta distancia se midió en cada par de centroides, luego se sumaron todas las distancias para medir el grado de *corrimiento* entre los centroides de los dos flujos. El cuadro 1 muestra el grado de *corrimiento* de los centroides entre todos los flujos y el Km-sec. Se puede observar que la diferencia más grande ocurre entre el flujo stream500 y el Km-sec (en rojo), mientras que la menor diferencia ocurrió entre el flujo stream250 y stream50 (en azul).

	Stream250	**Stream100**	**Stream50**	**Km-sec**
Stream500	6.57E-11	3.28E-11	6.44E-11	6.81E-11
Stream250		3.51E-11	4.20E-12	6.53E-11
Stream100			3.00E-11	5.20E-11
Stream50				6.39E-11

Cuadro 1: Grado de *corrimiento* de los centroides

Al realizar la comparación entre los centroides individualmente se observa que la mayor diferencia ocurre entre el cluster 3 del flujo stream500 y el stream250 con una diferencia de 2.86E-11. La menor diferencia ocurre en el cluster 1 entre el flujo de stream250 y stream50 con una diferencia de -2.22E-16. La tabla con estos valores no está incluida por falta de espacio.

Para la experimentación se utilizó una hoja de un Blade de 8 hojas, con 2 procesadores quad core Intel Xeón e5405 de 2.0 GHz en cada una de ellas. Cada hoja posee 10Gb de RAM (compartido entre ambos procesadores) y cache L2 de 2 x 6Mb entre par de núcleos.

5. Conclusiones

En este trabajo se utilizó la técnica para el manejo dinámico de una ventana temporal de un flujo de datos presentado en [20], donde la característica es maximizar el tamaño de dicha ventana permitiendo que cada dato recolectado del flujo sea utilizado por el algoritmo de clustering la mayor cantidad de veces posibles.

Se midió el manejo del buffer utilizando una tarea de clustering con el algoritmo K-means [21] implementado en el paradigma MapReduce. El algoritmo se evaluó con un flujo de datos donde se aumentó la frecuencia de llegada de los datos en 50, 100, 250 y 500 veces. Los resultados obtenidos permiten observar un manejo eficiente de los datos produciendo resultados muy similares a los que se obtienen con el algoritmo de clustering secuencial utilizando el conjunto de datos completo.

Referencias

[1] N. Takahashi y col. "A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance". En: *2009 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009, Fukuoka, Japan, March 16-19, 2009*. 2009, págs. 1100-1105.

[2] Y. Noh, D. Han e Y. Byun. "Real-Time Data Stream Processing for Ubiquitous Home Network Systems". En: *4th International Conference on Multimedia and Ubiquitous Engineering, MUE 2010, Cebu, Philippines, 11-13 August, 2010*.

[3] C. Kuka. "Processing the uncertainty: Quality-aware data stream processing for dynamic context models". En: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. 2012, págs. 560-561.

[4] D. Bonino y F. Corno. "spChains: A Declarative Framework for Data Stream Processing in Pervasive Applications". En: *Procedia Computer Science* 10 (2012). {ANT} 2012 and MobiWIS 2012. ISSN: 1877-0509.

[5] J. Stefanowski, A. Cuzzocrea y D. Slezak. "Processing and mining complex data streams". En: *Inf. Sci.* 285 (2014), págs. 63-65.

[6] R. Agerri y col. "Big data for Natural Language Processing: A streaming approach". En: *Knowledge-Based Systems* 79 (2015). ISSN: 0950-7051.

[7] Y. Ma y col. "Remote Sensing Big Data Computing". En: *Future Gener. Comput. Syst.* 51.C (oct. de 2015), págs. 47-60. ISSN: 0167-739X.

[8] P. ZareMoodi, H. Beigy y S. Kamali Siahroudi. "Novel Class Detection in Data Streams Using Local Patterns and Neighborhood Graph". En: *Neurocomput.* 158.C (jun. de 2015), págs. 234-245. ISSN: 0925-2312.

[9] D. Desai y A. Joshi. "A Deviant Load Shedding System for Data Stream Mining". En: *Procedia Computer Science* 45 (2015). International Conference on Advanced Computing Technologies and Applications (ICACTA). ISSN: 1877-0509.

[10] A. Rajaraman y J. D. Ullman. *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107015359, 9781107015357.

[11] G. Hager y G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. ("Chapman and Hall/CRCÇomputational Science), CRC Press, 2010. ISBN: 97814398119241.

[12] P. Pacheco. *An Introduction to Parallel Programming*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 9780123742605.

[13] S. Zhang y col. "Cloud Computing Research and Development Trend". En: *Future Networks, 2010. ICFN '10. Second International Conference on*. 2010.

[14] S. S. Saurabh Bilgaiyan y S. S. Sahu. "Cloud Computing: Concept, Terminologies, Issues, Recent Technologies". En: *Research Journal of Applied Sciences* 9 (2014), págs. 614-618.

[15] Apache Hadoop. `https://hadoop.apache.org/`. Accedido en 07/2016.

[16] MapReduce. `http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`. Accedido en 07/2016.

[17] J. Dean y S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". En: *Commun. ACM* 51.1 (ene. de 2008). ISSN: 0001-0782.

[18] Apache Hadoop. `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`. Accedido en 07/2016.

[19] C. C. Aggarwal. "A Survey of Stream Clustering Algorithms". En: *Data Clustering: Algorithms and Applications*. 2013, págs. 231-258.

[20] M. J. Basgall, W. Hasperué y M. Naiouf. "Tratamiento de un flujo de datos usando ventanas deslizantes con MapReduce". En: *IV Jornadas de Cloud Computing y Big Data, La Plata, Buenos Aires, Argentina* (2016).

[21] J. MacQueen y col. "Some methods for classification and analysis of multivariate observations". En: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, págs. 281-297.

[22] S.-S. Kim y H.-K. Ahn. "An improved data stream algorithm for clustering". En: *Computational Geometry* 48.9 (2015). ISSN: 0925-7721.

[23] E. Lughofer y M. Sayed-Mouchaweh. "Autonomous Data Stream Clustering Implementing Split-and-merge Concepts - Towards a Plug-and-play Approach". En: *Inf. Sci.* 304.C (mayo de 2015). ISSN: 0020-0255.

[24] A. Sancho-Asensio y col. "Improving data partition schemes in Smart Grids via clustering data streams". En: *Expert Systems with Applications* 41.13 (2014), págs. 5832 -5842. ISSN: 0957-4174.

[25] Y. Li y col. "Incremental Entropy-based Clustering on Categorical Data Streams with Concept Drift". En: *Know.-Based Syst.* 59 (mar. de 2014).

[26] Z. Miller y col. "Twitter spammer detection using data stream clustering". En: *Information Sciences* 260 (2014), págs. 64 -73. ISSN: 0020-0255.

[27] R. Mythily, A. Banu y S. Raghunathan. "Clustering Models for Data Stream Mining". En: *Procedia Computer Science* 46 (2015). Proceedings of the International Conference on Information and Communication Technologies, {ICICT} 2014, 3-5 December 2014, Kochi, India. ISSN: 1877-0509.

[28] PhridviRaj, C. Srinivas y C. GuruRao. "Clustering Text Data Streams – A Tree based Approach with Ternary Function and Ternary Feature Vector". En: *Procedia Computer Science* 31 (2014). 2nd International Conference on Information Technology and Quantitative Management, {ITQM} 2014.

[29] M. Z. ur Rehman y col. "Hyper-ellipsoidal clustering technique for evolving data stream". En: *Knowledge-Based Systems* 70 (2014). ISSN: 0950-7051.

[30] M. Hahsler y M. Bolaños. "Clustering Data Streams Based on Shared Density between Micro-Clusters". En: *IEEE Transactions on Knowledge and Data Engineering* 28.6 (2016), págs. 1449-1461. ISSN: 1041-4347.

[31] W. Hu y col. "Research on Parallel Data Stream Clustering Algorithm Based on Grid and Density". En: *Computer Science and Mechanical Automation (CSMA), 2015 International Conference on*. 2015, págs. 70-75.

[32] X. Zhang y col. "Data Stream Clustering with Affinity Propagation". En: *IEEE Transactions on Knowledge and Data Engineering* 26.7 (2014).

[33] A. Amini, H. Saboohi y T. Y. Wah. "A Multi Density-Based Clustering Algorithm for Data Stream with Noise". En: *2013 IEEE 13th International Conference on Data Mining Workshops*. 2013, págs. 1105-1112.

[34] R. Fathzadeh y V. Mokhtari. "An ensemble learning approach for data stream clustering". En: *21st Iranian Conference on Electrical Engineering (ICEE)*. 2013.

[35] C. D. Wang y col. "SVStream: A Support Vector-Based Algorithm for Clustering Data Streams". En: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2013), págs. 1410-1424. ISSN: 1041-4347.

[36] Twitter4J. `http://twitter4j.org/en/index.html`. Accedido en 07/2016.

[37] The Streaming APIs. `https://dev.twitter.com/streaming/overview`. Accedido en 07/2016.

[38] E. Rasmussen. "Information Retrieval". En: ed. por W. B. Frakes y R. Baeza-Yates. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992. Cap. Clustering Algorithms, págs. 419-442. ISBN: 0-13-463837-9.