Minería de patrones en *Big Data*

Francisco Padillo, José María Luna, Sebastián Ventura

Departamento de informática y análisis numérico, Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España.
{fpadillo,jmluna,sventura}@uco.es,

Resumen El creciente interés en el almacenamiento de datos ha provocado que la cantidad de datos a tratar sea cada vez más grande, enlenteciendo el proceso de transformar datos en información útil y de interés. En este sentido, la minería de patrones se considera como una de las partes esenciales del análisis de datos. Sin embargo, con la creciente importancia en el almacenamiento de datos, el análisis y tratamiento de estos ha llegado a ser casi inmanejable, provocando que el rendimiento de los algoritmos tradicionales se vea enlentecido. El objetivo de este trabajo es proponer un nuevo conjunto de algoritmos para la extracción de patrones en *Big Data*, usando MapReduce con Apache Hadoop. Para comprobar el rendimiento, se han considerado *datasets* con más de $3 \cdot 10^9$ instancias, más de 5 millones de ítems y archivos mayores de 800GB. El estudio experimental demuestra la utilidad de considerar MapReduce para abordar *Big Data*, así como la inaplicabilidad de éste para datos pequeños.

Keywords: Minería de patrones, Big Data, MapReduce, Hadoop

1. Introducción

El análisis de datos ha recibido especial atención en los últimos años, donde se han propuesto muchas técnicas para transformar datos brutos en información útil e interesante. Con la creciente importancia de los datos en muchos dominios, la cantidad de datos a tratar ha llegado a ser casi inmanejable, provocando que muchas técnicas no sean aplicables en un tiempo razonable. El termino *Big Data* es cada vez más usado para referirse a los retos derivados del procesamiento de tales cantidades de datos de una forma eficiente[1].

La minería de patrones se considera como una parte esencial en el análisis de datos. Su objetivo es extraer información desconocida de los datos a través del descubrimiento de relaciones ocultas entre ítems [2]. Desde la aparición de la minería de patrones frecuentes, se han propuesto una gran cantidad de algoritmos para tal fin [3,4,5]. Muchos de éstos están basados en el algoritmo Apriori [6]. Aunque todos estos algoritmos trabajaban bien en *datasets* pequeños, hoy en día no son aplicables en *Big Data* sin una adaptación a las tecnologías emergentes [1].

Una de éstas tecnologías es MapReduce [7]. En ella se propone el uso de un modelo de programación que ofrece un método simple y robusto de procesar enormes cantidades de datos, todo ello haciendo uso de *keys-value* $\langle k, v \rangle$.

Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial, pp. 769-778

El objetivo de este trabajo es proponer un conjunto de algoritmos para la extracción de patrones en *Big Data*, haciendo uso del *framework* MapReduce y de la implementación Apache Hadoop [7]. Se han propuesto cuatro algoritmos diferentes, todos ellos haciendo uso de MapReduce. Para comprobar su escalabilidad, se han llevado a cabo una gran cantidad de experimentos. En este sentido, se han considerado hasta $3 \cdot 10^9$ transacciones y más de 5 millones de ítems. También, se incluyen comparaciones con algoritmos secuenciales, así como con otros algoritmos de MapReduce. Los resultados revelan la importancia de considerar algoritmos de MapReduce cuando se trabaja con *Big Data*.

El resto del artículo está organizado como sigue. Sección 2 presenta algunas definiciones relevantes relacionadas con esta tarea; Sección 3 describe los algoritmos propuestos; Sección 4 presenta los *datasets* usados así como los resultados obtenidos. Finalmente, las conclusiones son expuestas en la Sección 5.

2. Trabajos relacionados

La minería de patrones es la tarea de extraer patrones de utilidad e interés para un determinado fin. Estos patrones se podrían definir de modo más formal como $\{P = \{i_j, ..., i_k\} \subset \mathcal{I}, j \geq 1, k \leq n\}$, siendo $\mathcal{I} = \{i_1, i_2, .., i_n\}$ un conjunto de ítems, y siendo P un patrón, también conocido como *itemset*, o subconjunto de $\mathcal{I}$. La longitud de un patrón P es denotada por $|P|$, siendo ésta el número de *singletons* que incluye. Por lo que, para $P = \{i_1, ..., i_j\} \subset \mathcal{I}$, su longitud es definida como $|P| = j$. Adicionalmente, dado el conjunto total de transacciones $\mathcal{T} = \{t_1, t_2, ..., t_m\}$, el soporte de un patrón P es definido como el número de transacciones satisfechas por P, $support(P) = |\{\forall t_j \in \mathcal{T} : P \subseteq t_j\}|$. Un patrón P es considerado como frecuente si y sólo si $soporte(P) \geq umbral$. Es interesante destacar que la frecuencia es una propiedad monótona, es decir, que si un patrón no es frecuente, ninguno de sus super-patrones puede ser frecuente.

El uso de fuerza bruta para determinar el conjunto de patrones en un *dataset* conteniendo j *singletons* y N transacciones es una tarea muy costosa, pues requiere generar $M = 2^j - 1$ ítems y $\mathcal{O}(M \times N \times j)$ comparaciones. Sin embargo, basada en la propiedad antimonótona, es posible descartar una gran cantidad del espacio de búsqueda. Esta técnica es conocida como poda por soporte y fue por primera vez introducida por *Agrawal et al.* [6] en el algoritmo Apriori.

El proceso de búsqueda exhaustiva también puede ser llevado a cabo usando una estructura de datos con forma de arboles donde cada ruta representa un conjunto de patrones [3]. *Zaki* [8] también propuso otro algoritmo muy eficiente, Eclat. Éste transforma el *dataset* original en un formato vertical, donde cada item es almacenado en el *dataset* junto con la lista de transacciones-ids donde el item puede ser encontrado.

Aunque estos algoritmos son eficientes, no son aplicables sobre *Big Data*. Con el objetivo de abordar este problema sobre grandes cantidades de datos, se ha propuesto el uso de otro tipo de representaciones de los datos [9] que permiten acelerar el proceso de extraer información. Otros autores, han propuesto abordar este problema paralelizándolo a través de computación GPU [10] permitiendo

de esta forma abordar una mayor cantidad de datos. Aunque estas últimas propuestas han supuesto una considerable mejora, otros enfoques han recibido más atención. *Moens et al.* [1] propuso el uso de MapReduce para afrontar este problema, los resultados obtenidos demostraron la aplicabilidad de este *framework* para la extracción de patrones sobre *Big Data*.

3. Propuestas MapReduce para la extracción de patrones en *Big Data*

En esta sección se describen los algoritmos propuestos para extraer patrones en *Big Data*, haciendo uso del *framework* MapReduce y de la implementación Apache Hadoop.

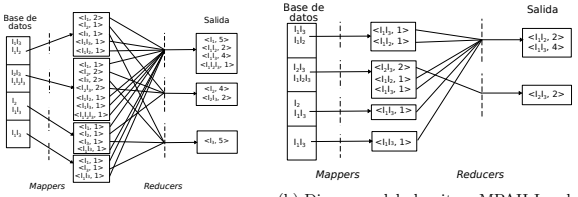
Una característica novedosa de éstos con respecto a trabajos previos [1], es que nuestras propuestas hacen uso de múltiples *reducers*. El uso de múltiples *reducers* permite lograr un mayor grado de paralelismo. Sin embargo, es necesario balancear lo máximo posible los *reducers* ya que siempre se está penalizado por el más costoso. En este sentido, proponemos el uso de 3 *reducers* con la siguiente estructura: dado un conjunto de ítems $\mathcal{I} = \{i_1, i_2, ..., i_j\}$, el primer *reducer* procesaría todos los ítems que contengan i_1; de los restantes, el segundo *reducer* procesaría los que contenga i_2; y por último, el último *reducer* procesaría los restantes. Se podría usar un número diferente de *reducers* siguiendo la distribución anteriormente mencionada.

3.1. MPAH

En primer lugar, se describe el algoritmo MPAH (Minería de Patrones usando Apache Hadoop). Este algoritmo divide el *dataset* de entrada en sub-partes de igual tamaño. Extrayendo todos los posibles patrones sobre cada una de ellas. Para evitar analizar patrones que no existan en la base de datos, MPAH no analiza todas las posible combinaciones de los ítems sino sólo aquellas que están presentes en el *dataset*.

Considerando las características anteriores, MPAH funciona ejecutando un *mapper* diferente para cada uno de los *sub-datasets*, siendo la función de cada uno de ellos extraer todos los posibles ítems presentes en su *sub-dataset* (Ver Figura 1a). Tras esto, el *reducer* es el encargado de combinar la lista completa de $\langle k, v \rangle$, distribuyendo éstas de acuerdo a su clave tal y como fue descrito antes.

Un hándicap importante de MPAH es la enorme cantidad de $\langle k, v \rangle$ que podría generar debido a la gran cantidad de patrones que podrían ser obtenidos. Para afrontar este problema, una opción es no crear todos los patrones de cualquier tamaño, sino, realizar un algoritmo iterativo donde en cada una de las iteraciones sólo se generen los patrones de tamaño j. Siguiendo esta filosofía surge MPAH-I (Minería de Patrones Apache Hadoop - Iterativo), como un algoritmo iterativo que divide el *dataset* de entrada en *sub-datasets* donde cada uno de ellos es analizado por un *mapper*. Cada uno de los *mapper* generará todas las combinaciones de tamaño j posibles para su *sub-dataset*. Posteriormente, los *reducers*



(a) Diagrama del algoritmo MPAH.

(b) Diagrama del algoritmo MPAH-I en la iteracción 2ª.

Figura 1: Diagrama de las dos versiones de MPAH, la clásica y la iterativa.

agruparán todas las $\langle k, v\rangle$ siguiendo la metodología ya descrita. A diferencia de MPAH, MPAH-I procesará en cada una de las iteraciones una menor cantidad de $\langle k, v\rangle$ en el *reducer*. Cabe destacar que la principal diferencia entre MPAH y su versión iterativa es que, ésta última, requiere j iteraciones para obtener todos los patrones, mientras que MPAH sólo necesita una única iteración más costosa. La Figura 1b muestra una ejecución de MPAH-I para su segunda iteración, en la que los *mappers* generan todos los patrones de tamaño 2 que pueden ser extraídos para cada una de los *sub-dataset*.

El análisis experimental revela que los dos métodos anteriores (MPAH y MPAH-I) extraen cualquier patrón que ocurre al menos una vez en la base de datos. Debido a que esta cantidad de patrones podría llegar a ser elevada, se ha introducido un modo de reducir el espacio de búsqueda.

3.2. AprioriAH

AprioriAH (Apriori Apache Hadoop) es el primer algoritmo de los propuestos en incluir un umbral de soporte. Mientras que los ya propuestos por *Moens et al.* lo introducían en el *mapper*, provocando una perdida de patrones como los propios autores indican en [1]. Nuestra propuesta introduce un umbral de soporte en el *reducer*, de forma que no se está perdiendo ningún patrón ya que cada *mapper* generará todos los posibles patrones. Luego, para introducir la propiedad anti-monótona en el *framework* MapReduce es necesario usar una serie de iteraciones, donde en cada una de ellas se obtienen los patrones de tamaño $j+1$ a partir de los patrones de tamaño j. De esta forma, si un patrón P_i no es frecuente de acuerdo al umbral de soporte prefijado, ninguno de sus superconjuntos podría ser frecuente. La Figura 2 muestra las iteraciones requeridas para el algoritmo AprioriAH para un *dataset* de ejemplo. Se ha usado un umbral mínimo de soporte de 4, por lo que cualquier patrón con menor soporte es descartado (lineas discontinuas). Este algoritmo extrae todos los *singletons* en la primera iteración, descubriendo que el patrón I_2 es infrecuente ya que sólo es satisfecho por 3 instancias. En la iteración j^{a}, el objetivo de cada *mapper*

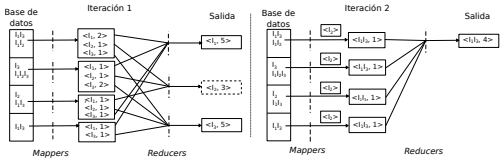


Figura 2: Diagrama del algoritmo AprioriAH.

es doble: primero, éstos reciben dos conjuntos: los patrones infrecuentes en la iteración previa ($|P| = j - 1$) y un conjunto de datos de entrada. A partir de estos dos, obtiene un conjunto de $\langle k, v \rangle$, siendo k de tamaño j, es decir, $|c| = j$.

Aunque este algoritmo supone una mejora con respecto a los anteriores, este algoritmo aún tiene una gran complejidad computacional, especialmente cuando la cantidad de patrones infrecuentes es grande. En esta situación, se requieren una gran cantidad de iteraciones, enlenteciéndose el proceso de extracción.

3.3. MMPAH

Con el objetivo de reducir el número de patrones frecuentes incluso usando umbrales altos, proponemos un algoritmo que no siempre obtiene la lista completa de patrones frecuentes pero siempre garantiza que los patrones obtenidos son aquellos con mayor soporte. Este algoritmo, conocido como MMPAH (Minería de Mejores Patrones usando Apache Hadoop) funciona de forma similar a AprioriAH, requiriendo un conjunto de iteraciones donde en cada una de ellas se calculan los patrones de tamaño j. La principal característica de MMPAH es su procedimiento para reducir la base de datos de entrada eliminando todos los ítems infrecuentes encontrados hasta el momento. Esto implica que el *dataset* va disminuyendo su tamaño conforme las iteraciones avanzan, reduciendo de esta forma los requisitos computacionales.

La Figura 3 muestra un ejemplo de ejecución de este algoritmo para un *dataset* de ejemplo. En la primera iteración, todos los *singletons* son analizados, y todos aquellos frecuentes son almacenados en un archivo independiente. En la

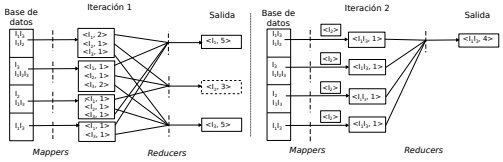


Figura 3: Diagrama del algoritmo MMPAH.

segunda iteración, el algoritmo distribuye el *dataset* entre los *mappers* junto con el archivo de patrones infrecuentes, siendo el objetivo de cada *mapper* eliminar el conjunto de infrecuentes del *sub-dataset*, y extraer todos los patrones para su porción de base de datos de tamaño $j = 2$. Es importante destacar que para este ejemplo no es necesario ejecutar $j = 3$ desde que ningún patrón de este tamaño podría ser extraído a partir del nuevo *dataset*.

4. Experimentación

En esta sección, se ha estudiado el rendimiento de los algoritmos propuestos usando diferentes tamaños de datos. Es interesante destacar que todos los tiempos de ejecución son la media para 10 ejecuciones diferentes, aunque todas ellas obtengan las mismas soluciones. El objetivo es aliviar las variaciones en tiempo de ejecución derivadas de la administración de recursos. No se ha necesitado realizar ningún tipo de comparación sobre la eficacia de los algoritmos ya que se ha seguido un enfoque exhaustivo.

Se han considerado *datasets* donde el número de instancias varia desde $2 \cdot 10^6$ hasta $3 \cdot 10^9$. Con respecto al tamaño de archivo, varían desde 9,6 MB hasta 814 GB. El espacio de búsqueda varía desde 2^4 hasta $2^{70.000}$ patrones. Finalmente, se han ejecutado una serie de experimentos sobre base de datos reales, considerándose uno de los *datasets* más grandes dentro de la comunidad [1], el cual contiene un espacio de búsqueda de $2^{5.267.646}$ patrones, con un tamaño de archivo de 1,5 GB.

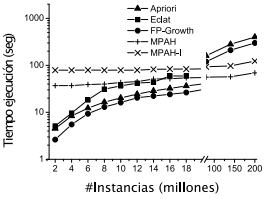
4.1. Secuencial Vs MapReduce

En este estudio experimental se ha analizado el rendimiento de un conjunto de algoritmos cuando tanto el número de atributos como el de instancias se incrementa. Con el objetivo de demostrar bajo que condiciones es aplicable MapReduce, también se han incluido comparaciones con algoritmos secuenciales. Es importante destacar que todos estos algoritmos obtienen todos los ítems posibles desde que no se ha usado ningún tipo de poda.

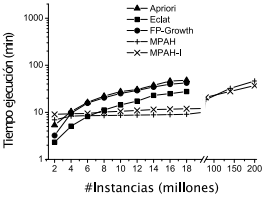
Analizando la Figura 4a, los resultados ilustran que los algoritmos clásicos como Apriori, FP-Growth o Eclat se comportan realmente bien incluso cuando se consideran un gran número de instancias (20 millones de instancias). Sin embargo, cuando el numero de instancias continúa creciendo, algoritmos basados en MapReduce como MPAH y MPAH-I se comportan mejor que los algoritmos secuenciales, obteniendo en el mejor de los casos hasta dos ordenes de magnitud de ventaja [2]. Como puede ser apreciado, para *datasets* con mas de 80 millones de instancias, el uso de MapReduce es necesario. Sin importar el número de instancias, el tiempo de ejecución de los algoritmos basados en MapReduce es casi el mismo, mientras que el tiempo de ejecución necesario para los secuenciales incrementa linealmente con el número de instancias.

[1] *Webdocs* disponible en la web: http://fimi.ua.ac.be/data/

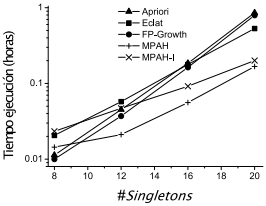
[2] Por razones de espacio, la tabla se encuentra disponible en http://www.uco.es/grupos/kdis/kdiswiki/MPAH/



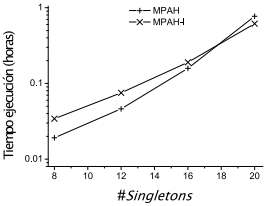
(a) Se consideraron 8 *singletons* y un número de instancias que varía desde $2{\cdot}10^6$ hasta $2 \cdot 10^8$.



(b) Se consideraron 20 *singletons* y un número de instancias que varía desde $2{\cdot}10^6$ hasta $2 \cdot 10^8$.



(c) 20 millones de instancias y un número de *singletons* que varía desde 8 hasta 20.



(d) 200 millones de instancias y un número de *singletons* que varía desde 8 hasta 20.

Figura 4: Estudio experimental considerando los algoritmos Eclat, FP-Growth, Apriori, MPAH y MPAH-I.

Continuando el estudio experimental, se ha incrementado el número de *singletons* desde 8 hasta 20, de forma que el espacio de búsqueda incrementa desde $2^8-1 = 255$ hasta $2^{20}-1 = 1.048.575$ patrones. Tal y como muestra la Figura 4b, los algoritmos de MapReduce se comportan mejor incluso cuando el número de instancias es pequeño. Esto demuestra que aunque los algoritmos secuenciales son apropiados para base de datos pequeñas, los algoritmos de MapReduce son necesarios cuando se consideran grandes cantidades de datos. En ese sentido, y considerando *datasets* con mas de 80 millones de instancias, ninguno de los algoritmos secuenciales pudo ejecutarse por limitaciones de hardware. Luego, los algoritmos de MapReduce son apropiados para manipular esta extremada cantidad de datos y de comparaciones (más de $2,19 \cdot 10^{20}$ comparaciones). Cuando se consideran los dos algoritmos de MapReduce, $0,231$ es la diferencia media de MPAH con MPAH-I. Mientras que el caso mas favorable determina que MPAH obtiene una diferencia de un orden de magnitud con respecto a MPAH-I.

Para acabar este estudio experimental, se ha considerado también interesante mostrar como se comportan los algoritmos cuando el número de *singletons* crece.

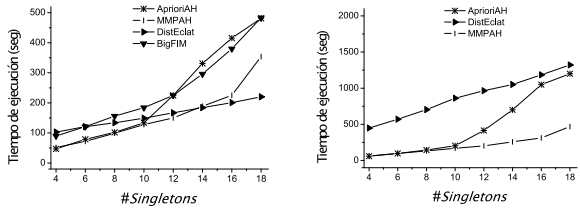
En este sentido, se ha considerado una base de datos con 20 millones de instancias (Ver Figura 4c), las propuestas de MapReduce tienden a comportarse mejor cuando aumenta tanto el número de instancias como el de *singletons*. Adicionalmente, la Figura 4d muestra el mismo análisis pero considerado 200 millones de instancias. Es este último análisis, los algoritmos secuenciales no pudieron ser ejecutados por limitaciones de hardware. Tal y como ilustra, los algoritmos de MapReduce son apropiados cuando se trabaja con enormes cantidades de datos, donde se requieren una enorme cantidad de comparaciones.

4.2. MapReduce y umbrales de soporte

En este segundo estudio, se ha analizado el rendimiento de varios algoritmos basados en MapReduce, considerando un umbral mínimo de soporte. Todos estos algoritmos reducen el espacio de búsqueda usando la propiedad anti-monótona.

En primer lugar, se ha estudiado el rendimiento de los algoritmos de MapReduce sobre base de datos extremadamente grandes. El número de *singletons* varía de 4 a 18, mientras que el número de instancias es de hasta $3 \cdot 10^7$. Cuando se analiza la Figura 5a, los resultados muestran que, para 30 millones de instancias y más de 12 *singletons*, los cuatro algoritmos se comportan de forma similar. El rendimiento de los algoritmos difiere cuando el espacio de búsqueda incrementa. Tal y como se puede apreciar, MMPAH y DistEclat son los dos algoritmos que mejor se comportan cuando el espacio de búsqueda incrementa. Como los autores describieron en [1], DistEclat es una versión distribuida de Eclat [8] que particiona el espacio de búsqueda y que es más eficiente que BigFIM.

Continuando el análisis, el número de instancias se incrementa en un orden de magnitud, considerando *datasets* con 300 millones de instancias. En este segundo análisis, BigFIM no se ha ejecutado ya que DistEclat escala mejor [1]. La Figura 5b muestra los resultados. Como se puede apreciar, MMPAH requiere un tiempo similar de ejecución para 30 millones de instancias, luego el rendimiento



(a) $3 \cdot 10^7$ instancias y un número de *singletons* que varía desde 4 a 18.

(b) $3 \cdot 10^8$ instancias y un número de *singletons* que varía desde 4 a 18.

Figura 5: Estudio experimental considerando AprioriAH, MMPAH, BigFIM y DistEclat

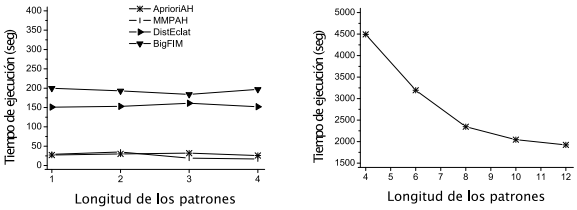
casi no se ve afectado por el número de instancias. A diferencia de MMPAH, ni ApriroiH ni DisEclat escalan de forma adecuada cuando se incrementa el número de instancias. De hecho, ambos algoritmos requieren un tiempo de ejecución de un orden de magnitud mayor que el requerido por MMPAH.

La Figura 6a muestra un análisis completamente diferente. Donde el objetivo es analizar el rendimiento de un conjunto de algoritmos midiendo el tiempo de ejecución para extraer todos los patrones de longitud $k = 1, k = 2, k = 3$ y $k = 4$. Tal y como se puede apreciar, MMPAH y AprioriH se comportan mucho mejor que BigFIM o DistEclat sin importar el tamaño de los patrones.

Adicionalmente, para determinar el rendimiento del algoritmo MMPAH cuando las capacidad del cluster cambian, se han ejecutado un conjunto de experimentos donde el número de nodos de cómputo varía. De este modo, se puede observar la dependencia entre el algoritmo con respecto a la arquitectura de hardware disponible. La Figura 6b muestra el tiempo de ejecución para este algoritmo cuando se varía el número de nodos de cómputo. Como se puede apreciar, la dependencia entre el tamaño del cluster y el rendimiento del algoritmo es alta. Sin embargo, el algoritmo se comporta casi igual cuando el número de nodos es mayor de 10.

Continuando este análisis con el algoritmo MMPAH, los siguientes experimentos muestran su comportamiento cuando se considera un número de *singletons* de $3 \cdot 10^4$ y $7 \cdot 10^4$. El objetivo de este análisis es demostrar que este algoritmo se comporta bien cuando el número de ítems es alto, siendo capaz de obtener 760 ítems en un espacio de búsqueda de $2^{30.000}$ ítems en menos de 300 minutos.

Para acabar este estudio, se ha ejecutado MMPAH sobre uno de los *datasets* más grandes de los disponible en la comunidad de minería de patrones, *webdocs*. Este *dataset* contiene 5.267.656 ítems, distribuidos a través de 1.692.082 instancias. Los resultados demuestran que el algoritmo propuesto se comporta bastante bien, descubriendo los patrones frecuentes en menos de 300 minutos.



(a) Estudio de la influencia de la longitud de los patrones.

(b) Estudio de la influencia del número de nodos para el algoritmo MMPAH.

Figura 6: Estudios para determinar la influencia de la longitud de los patrones a extraer, así como de las caracteristicas del harware en el rendimiento.

5. Conclusión

En este trabajo, se han propuesto un conjunto de algoritmos para la extracción de patrones en *Big Data*, haciendo uso de MapReduce y de Apache Hadoop. Para comprobar el rendimiento y la escalabilidad de los algoritmos se consideraron *datasets* con más de $3{\cdot}10^9$ instancias y más de 5 millones de *singletons*. Todos los algoritmos se compararon con algoritmos ampliamente conocidos en la minería de patrones. La experimentación demuestra que los algoritmos propuestos funcionan bastante bien cuando el espacio de búsqueda incrementa (donde se usó un espacio de búsqueda de $2^{5.267.656}$). Además, es importante destacar que los algoritmos se comportan realmente bien cuando el número de instancias incrementa, logrando un excelente rendimiento con $3 \cdot 10^9$ instancias. Los resultados también demuestran la inaplicabilidad de MapReduce con datos pequeños.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Innovación y Ciencia, y los fondos FEDER, bajo el proyecto TIN-2014-55252-P.

Referencias

1. S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013*, Santa Clara, CA, USA, 2013, pp. 111–118.
2. C. C. Aggarwal and J. Han, Eds., *Frequent Pattern Mining*. Springer, 2014.
3. J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, pp. 53–87, 2004.
4. S. Zhang, Z. Du, and J. Wang, "New techniques for mining frequent patterns in unordered trees," *IEEE Transactions on Cybernetics*, vol. 45, pp. 1113–1125, 2015.
5. J. M. Luna, "Pattern mining: current status and emerging topics," *Progress in Artificial Intelligence*, pp. 1–6, 2016. [Online]. Available: http://dx.doi.org/10.1007/s13748-016-0090-4
6. R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.
7. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 107–113, 2008.
8. M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.
9. F. Padillo, J. M. Luna, A. Cano, and S. Ventura, "A data structure to speed-up machine learning algorithms on massive datasets," in *Hybrid Artificial Intelligent Systems*. Springer, 2016, pp. 365–376.
10. A. Cano, J. M. Luna, and S. Ventura, "High performance evaluation of evolutionary-mined association rules on gpus," *Journal of Supercomputing*, vol. 66, no. 3, pp. 1438–1461, 2013.