Parallelization of the Apriori Algorithm for the Search of Frequent Elements

Yonatan Mamani-Coaquira[1], Edith K. Chumpisuca Carrion[2]

[1] Universidad Nacional Micaela Bastidas de Apurímac, ymamanic@gmail.com, Abancay, Perú
[2]Universidad Nacional Micaela Bastidas de Apurímac, chumpisucakarina@gmail.com, Abancay, Perú

Abstract

There is a wide variety of techniques that increase application performance by alleviating one or more of the most important problems with today's processors. In this work, the execution time, speedup and efficiency of the linear Apriori algorithm are shown as well as parallel with the use of OpenMP. By identifying the frequent elements of transactional databases, in processing 5 thousand records the time improves in 42,078 seconds of the algorithm with openMP compared to the sequential algorithm, in the execution 8 processor cores were used.

Keywords: Apriori algorithm, frequent itemsets, parallel algorithm, openMP.

1. Introducción

La minería de datos se ha convertido en un área muy importante en las Ciencias de la Computación, esta área permite generar conocimiento a través de conjunto de datos proveniente de bases de datos estructurada o no estructurada. Los grandes volúmenes de información que son generadas por usuarios a nivel mundial sigue creciendo y llega hacer de manera exponencial, publicando a diario millones de datos por medio de redes sociales, blog, aulas virtuales, sistemas de información empresariales donde incluyen texto, imágenes, vídeos, músicas, etc., para realizar la selección información valiosa requiere de sistemas que procesen de manera automática. En diferentes trabajos se proponen algoritmos para resolver esta tarea como: Apriori, Eclat, Relim y FP-Growth. Sin embargo aún existe el problema de la búsqueda de elementos frecuentas en conjunto de datos masivos y aún existe la necesidad de optimizar la eficiencia y la escalabilidad de datos, por estas razones varias investigaciones han propuesto métodos paralelos para brindar solución a estos problemas, sin embargo; aún continúan las propuestas de algoritmos paralelos para optimizar las necesidades descritas.

Los algoritmos más conocidos para minería de reglas de asociación a partir de una base de datos son: Apriori [1], Relim [2], Eclat [3], FP-Growth [4] y FIN [5]. Por lo expuesto anteriormente aún existe la necesidad de aplicar algoritmo paralelos para optimizar los tiempos de ejecución. En el presente trabajo se implementó el algoritmo Apriori basado en la tecnología de paralelización de OpenMP que permite reducir el tiempo de espera en el procesamiento de los datos para encontrar elementos frecuentes en conjunto de datos masivos.

2. Concepción Teórica

2.1. Algoritmo Apriori

La idea principal del algoritmo Apriori es encontrar conjuntos de elementos frecuentes y que tenga mayor frecuencia en conjunto de datos masivos. Según Agrawal [1] enuncio como propiedad esencial para el algoritmo Apriori lo siguiente: logra afirmar que el subconjunto de un conjunto de elementos frecuentes también será un conjunto de elementos frecuentes. Por tal motivo; con este algoritmo se obtiene como primer paso los conjuntos de ítems o elementos frecuentes de dimensión 1, luego los de dimensión 2, así sucesivamente hasta que no se encuentren más conjuntos.

El proceso para encontrar elementos frecuente mediante el algoritmo Apriori es el siguiente [6]:

a) Paso 1: Establecer el soporte mínimo y la confianza de acuerdo con la definición del usuario.

b) Paso 2: Construye el candidato 1-itemsets. Y entonces generar los conjuntos frecuentes 1-itemsets mediante la poda de algunos 1-itemssets conjuntos de elementos si sus valores de soporte son inferiores a soporte mínimo.

c) Paso 3: Unir los conjuntos de 1-itemsets frecuentes entre sí para construir candidatos de conjuntos 2-itemsets y pasar algunos conjuntos de elementos poco frecuentes de los 2 conjuntos de elementos candidatos a crear los 2-itemsets frecuentes.

d) Paso 4: Repita los pasos igualmente paso 3 hasta que no haya más se pueden crear conjuntos de elementos candidatos.

La Fig. 1 muestra el algoritmo Apriori que fue propuesto por Agrawal [1], este algoritmo permite encontrar elementos frecuentes en un conjunto de datos.

```
1: Input: T //conjunto de transacciones
2: Output: L //lista de patrones encontrados en los datos
3: L ← φ
4: for all t ϵ T do
5:     for s = 1 to s <= |t| do
6:         C ← {∀P : P ← {i_j, ..., i_n} ∧ P ⊆ t ∧ |P| ← s}
   // candidate item-sets int t
7:         ∀P ϵ C, then support(P)←1
8:         if C ∩ L ≠ φ then
9:             ∀PϵL : PϵC, then support(P)++
10:        end if
11:        L ← L ∪ {C \ L} // include new patterns in L
12:    end for
13: end for
14: return L
```

Fig. 1. Algoritmo Apriori Secuencial

La Fig. 2 es la representación de los pasos mencionados anteriormente para encontrar los elementos frecuentes, esta figura está representado mediante tablas. TID viene hacer los identificadores; I1, I2,… son los elementos o ítems y Items-count es la cantidad de veces que aparece uno o varios elementos.

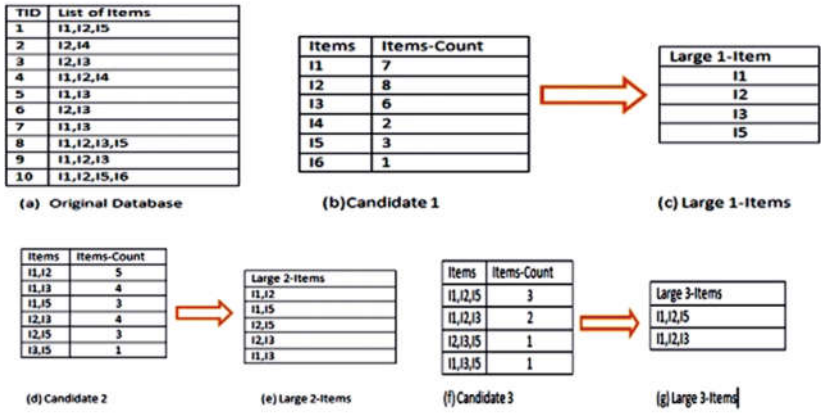

Fig. 2. Encontrar elementos frecuentes paso a paso

2.2. Paralelización con OpenMP

OpenMP (Open Multi-Processing) es un API (Application Programming Interface) para programación paralela de memoria compartida. Los MP en OpenMP significan multiprocesamiento. Este API está diseñado para sistemas en los que cada hilo o proceso puede potencialmente tener acceso a toda la memoria disponible, cuando se programa con OpenMP se puede ver que el sistema como colección de núcleos o CPUs, todas las cuales tienen acceso a la memoria principal [7] como se muestra en la Fig. 3.

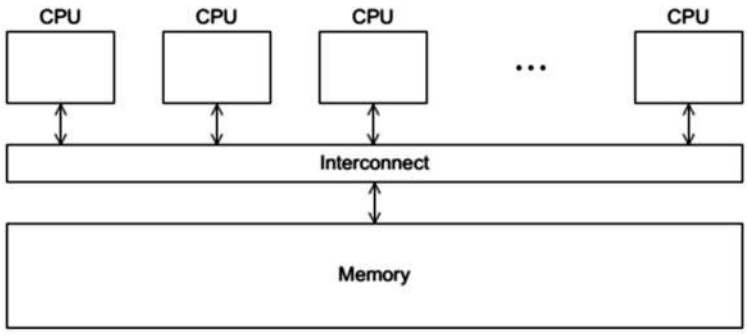

Fig. 3. Sistema de memoria compartida

A continuación se menciona algunas características para la implementación del algoritmo mediante OpenMP:

- Utilización de directivas o palabras reservadas del compilador (#pragma)
- Crear, sincroniza y destruye hilos
- Incorpora funciones especificas
- Paralelismo de memoria compartida
- Dentro de un paralelismo puede incorporar otro paralelismo

3. Trabajos Relacionados

Según [1] en su trabajo de investigación "Fast Algorithms for Mining Association Rules" enuncio como propiedad esencial para el algoritmo Apriori lo siguiente: se puede afirmar que el subconjunto de un conjunto de elementos frecuentes también será un conjunto de elementos frecuentes. Por tal motivo con este algoritmo se obtiene como primer paso los conjuntos de ítems o elementos frecuentes de dimensión 1, luego los de dimensión 2, así sucesivamente hasta que no se encuentren más conjuntos. Ventaja: algoritmo básico en minería de patrón frecuente, adecuado para base de datos densa. Desventaja: el espacio y la complejidad del tiempo son alto.

Varios trabajos [1], [8], [9], [10], [11] han sido presentados como mejora del algoritmo Apriori, las propuestas mencionadas utilizan paralelización para mejorar el tiempo de respuesta. Sin embargo después del algoritmo Apriori, surgieron otras técnicas algorítmicas que también realizan la identificación de elementos frecuentes en base de datos transacciones pero con mayor rendimiento en el tiempo de ejecución como son:

Eclat, según [3] en su trabajo denominado "Scalable Algorithms for Association Mining", el descubrimiento de reglas de asociación ha surgido como un problema importante en el descubrimiento de conocimiento y la extracción de datos. La Asociación La tarea de minería consiste en identificar los conjuntos de elementos frecuentes y, a continuación, formar reglas de implicación condicionales entre ellos. En este trabajo, se presente algoritmos eficientes para el descubrimiento de conjuntos de elementos frecuentes que forman la fase de procesamiento intensivo de la tarea. FP-Growth es el otro algoritmo, [12] la mayoría de los estudios anteriores adoptan un enfoque de

generación y prueba de conjuntos de candidatos tipo Apriori. Sin embargo, la generación de conjuntos de candidatos sigue siendo costosa, especialmente cuando existe una gran cantidad de patrones and/or patrones largos. En este estudio, proponemos una estructura novedosa de árbol de patrones frecuentes (FP-tree), que es una estructura extendida de prefijos de árbol para almacenar información comprimida y crucial sobre patrones frecuentes, y desarrollamos una FP eficiente basada en datos. Algoritmo YAFIM, según [13] en su trabajo de investigación denominado "YAFIM: A parallel frequent itemset mining algorithm with spark" realizó la propuesta de un nuevo algoritmo basado en Apriori este es el algoritmo ampliamente utilizado para extraer conjuntos de elementos frecuentes de un conjunto de datos transaccional. Sin embargo, el proceso FIM es tanto intensivo en datos como intensivo en computación. La computación paralela y distribuida es una estrategia efectiva y principalmente utilizada para acelerar los algoritmos de conjuntos de datos a gran escala. Sin embargo, los algoritmos de Apriori paralelos existentes implementados con el modelo MapReduce no son lo suficientemente eficientes para el cálculo iterativo.

Otro algoritmo propuesto fue D2P-Apriori [14] en este trabajo de investigación denominado "D2P-Apriori: A deep parallel frequent itemset mining algorithm with dynamic queue" se propuso Dynamic Queue y Deep Parallel Apriori (D2P-Apriori), un algoritmo de minería de elementos de paralelo frecuente en GPU para satisfacer el requisito de alto rendimiento. Por otro lado [15] propone el algoritmo HP-Apriori en su trabajo denominado "Hp-Apriori: Horizontal parallel-apriori algorithm for frequent itemset mining from big data", menciona que debido a la gran escala y la complejidad de los datos masivos, la minería de datos utilizando una sola computadora personal es un problema difícil. Con el aumento en el tamaño de las bases de datos, los sistemas de computación en paralelo pueden causar ventajas considerables en las aplicaciones de minería de datos mediante la explotación de algoritmos de minería de datos.

4. Experimentación

Implementamos un conjunto de pruebas. Cada prueba fue ejecutada con el algoritmo secuencial y paralelo con OpenMP. Las características de la computadora utilizada fueron: Asus S46C, procesador Intel Core i5-3317U CPU@1.70GHZ (4 CPUs), Memoria Ram 6GB y el sistema operativo Ubuntu 64 bits. Por otro lado; para la ejecución de los algoritmos se utilizó el lenguaje de programación C++, donde para cada uno de los algoritmos se utilizó como datos de entrada los siguientes: 500, 1000 y 5000 registros de transacciones. Por otra parte, para hallar el tiempo de ejecución en segundos se utilizó la función de "clock t". Para obtener los resultados de medición de desempeño del algoritmo Apriori secuencial y paralelo se utilizaron las siguientes métricas: el speedup y eficiencia, con 4 núcleos para cada conjunto de datos.

4.1. Evaluación de tiempos

En la Tabla 1 se muestra el tiempo en segundos, fueron ejecutados los algoritmos secuencial y paralelo con openMP mediante la técnica del algoritmo Apriori. Se puede ver en la Tabla 1 que mientras más datos sean evaluados el tiempo es más corto de forma paralela con respecto al algoritmo secuencial.

Tabla 1. Tiempo de ejecución en segundos por cada algoritmo

Algoritmo	0.5k	1k	5k
Apriori secuencial	2.755	5.447	253.844
Apriori paralelo openMP	2.24066	4.81778	211.466

4.2. Evaluación de speedup y eficiencia

En la Tabla 2 se muestra el tiempo de ejecución en segundos mediante el algoritmo Apriori procesados en 4 núcleos del procesador, fueron ejecutados 0.5k, 1k y 5k registros con el algoritmo paralelo Apriori basado en openMP.

Tabla 2. Tiempo de ejecución en segundos del algoritmo Apriori paralelo

Datos	**4 Núcleos**
0.5k	2.24066
1k	4.81776
5k	211.766

En la Tabla 3 se puede ver el Speedup evaluados en con 4 núcleos de un procesador. La Tabla 4 muestra como la eficiencia aumenta al tiempo en que lo procesa el número de procesadores y el tamaño o dimensión de los datos de entrada, los resultados de las Tablas 3 y 4 fueron obtenidos mediante el algoritmo Apriori paralelo con OpenMP.

Por otro lado; para hallar los valores de speedup se utilizó la siguiente formula: SN = T1/TN, y para eficiencia: EN = SN/N. Dónde: SN es valor de speedup, T1 tiempo de ejecución algoritmo serial, TN tiempo de ejecución del algoritmo en paralelo, N es el número de núcleos utilizados, EN el valor de la eficiencia [16].

Tabla 3. Evaluación de Speedup

Datos	4 Núcleos
0.5k	1.23
1k	1.131
5k	1.199

Tabla 4. Evaluación Eficiencia

Datos	4 Núcleos
0.5k	0.308
1k	0.283
5k	0.3

5. Discusiones y Resultados

Cuando se analiza los resultados obtenidos de los tiempos de ejecución para un algoritmo paralelo, se debe considerar no solamente el algoritmo, sino también el sistema operativo y las características del ordenador en donde se ejecutarán los datos de evaluación. Esto va permitir variar los resultados de los tiempos de ejecución para cada conjunto de datos porque las características del equipo pueden ser diferentes.

De los experimentos realizados en la Tabla 1, el tiempo de ejecución del algoritmo serial y paralelo tienen casi el mismo valor esto implica que no hay mucha variación con la ejecución utilizando un solo núcleo. El que tiene menor tiempo de ejecución es el algoritmo en paralelo con el valor de 211.766 segundos para 5 mil transacciones. Por otro lado; como puede verse el algoritmo paralelo tiene un Speedup óptimo, pues la ganancia de velocidad tiende a ser directamente proporcional al número de procesadores a medida que "n" aumenta.

La eficiencia mide el uso que se ejecuta en los procesadores. El experimento realizado en la Tabla 4 se aprecia que el algoritmo paralelo Apriori al procesar con 4 núcleos el tiempo de ejecución son casi similares con una cantidad de 0.5k, 1k y 5k transacciones.

6. Conclusiones

Según los resultados obtenidos fue posible y favorable la implementación del algoritmo Apriori de forma paralela mediante OpenMP, esto permitió identificar el tiempo de ejecución, speedup y eficiencia con varios registros de transacciones de base de datos. Por otro lado; la experimentación y evaluación realizada muestra la siguiente información: cuando se implementa algoritmos de forma paralela tiende a ser escalable y eficiente para resolver problemas de ejecución de grandes cantidades de datos.

Los resultados obtenidos en la Tabla 3 y 4 speedup y eficiencia respectivamente permitieron ver la eficiencia según los distintos valores de entrada. Es decir, dependiendo de los datos de entrada: el tiempo, speedup y eficiencia será mucho mejor con la utilización de openMP para algoritmos paralelos. También el tiempo en segundos influye mucho según la cantidad de transacciones que desea utilizar durante la experimentación. En general, como se puede ver en los experimentos, el algoritmo paralelo diseñado permite la solución de un problema de tiempo para los algoritmos secuenciales. Por otro lado; mencionar que es importante la elección del computador para procesar grandes cantidades de datos.

Referencias

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," Proc. Elev. Int. Conf. Data Eng., pp. 3–14, 1995.

[2] C. Borgelt, "Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination," in Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, 2005, pp. 66–70.

[3] M. J. Zaki, "Scalable algorithms for association mining," IEEE Trans. Knowl. Data Eng., vol. 12, no. 3, pp. 372–390, May 2000.

[4] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," Data Min. Knowl. Discov., vol. 8, no. 1, pp. 53–87, 2004.

[5] Z.-H. Deng and S.-L. Lv, "Fast mining frequent itemsets using Nodesets," Expert Syst. Appl., vol. 41, no. 10, pp. 4505–4512, 2014.

[6] N. Li, L. Zeng, Q. He, and Z. Shi, "Parallel Implementation of Apriori Algorithm Based on MapReduce," in 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2012, pp. 236–241.

[7] P. Pacheco, An Introduction to Parallel Programming. 2011.

[8] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," IEEE Trans. Knowl. Data Eng., vol. 8, no. 6, pp. 962–969, Dec. 1996.

[9] Y. Wei, R. Yang, and P. Liu, "An improved Apriori algorithm for association rules of mining," in 2009 IEEE International

Symposium on IT in Medicine Education, 2009, vol. 1, pp. 942–946.

[10] X.-W. Liu and P.-L. He, "The research of improved association rules mining Apriori algorithm," in Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), 2004, vol. 3, pp. 1577–1579 vol.3.

[11] Y. Ye and C.-C. Chiang, "A Parallel Apriori Algorithm for Frequent Itemsets Mining," in Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06), 2006, pp. 87–94.

[12] J. Pei et al., "Mining sequential patterns by pattern-growth: the PrefixSpan approach," IEEE Trans. Knowl. Data Eng., vol. 16, no. 11, pp. 1424–1440, Nov. 2004.

[13] H. Qiu, R. Gu, C. Yuan, and Y. Huang, "YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark," in 2014 IEEE International Parallel Distributed Processing Symposium Workshops, 2014, pp. 1664–1671.

[14] Y. Wang, T. Xu, S. Xue, and Y. Shen, "D2P-Apriori: A deep parallel frequent itemset mining algorithm with dynamic queue," in 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), 2018, pp. 649–654.

[15] M. Nadimi-Shahraki and M. Mansouri, "Hp-Apriori: Horizontal parallel-apriori algorithm for frequent itemset mining from big data," in 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, 2017, pp. 286–290.

[16] S. Nesmachnow, "COMPUTACIÓN DE ALTA PERFORMANCE," 2010. [Online]. Available: https://www.fing.edu.uy/inco/cursos/hpc/material/clases/Tema4-2010.pdf. [Accessed: 30-Jul-2019].