ALGORITMO DE AGRUPAMIENTO GLC PARALELO

Reynaldo Gil-García[1], José Manuel Badía-Contelles[2]

[1]Universidad de Oriente, Santiago de Cuba (Cuba).

gil@app.uo.edu.cu

[2]Universitat Jaume I, Castellón (España).

badia@icc.uji.es

Resumen

En la actualidad los algoritmos de agrupamiento se enfrentan al reto de que los conjuntos de objetos a agrupar son muy grandes, cambian en el transcurso del tiempo y los objetos están descritos por un gran número de rasgos. Esto ha provocado que los algoritmos de agrupamiento tradicionales tomen tiempos demasiado grandes para resolver estos problemas desde el punto de vista práctico. Algunas de las alternativas propuestas para solucionar o aminorar estas dificultades son los algoritmos incrementales y los algoritmos paralelos. En el trabajo se expone un algoritmo paralelo que busca de forma incremental las componentes conexas del grafo de β_0-semejanza asociado a un conjunto de objetos. Este algoritmo usa el paradigma de programación paralela de paso de mensajes para la paralelización del GLC. Se realiza, además, una valoración del costo computacional del algoritmo diseñado, se implementa en *Message Passing Interface* (MPI) y se realiza un análisis experimental del comportamiento del algoritmo para datos generados de forma aleatoria. El algoritmo paralelo obtenido logra un distribución adecuada de los datos y de los cálculos entre los procesadores. La eficiencia teórica obtenida es la unidad cuando la cantidad de objetos es mucho mayor que la cantidad de procesadores. Se demuestra experimentalmente que cuando los conjuntos de datos son grandes es ventajoso utilizar el algoritmo paralelo propuesto.

Palabras clave: algoritmo de agrupamiento, algoritmo incremental, algoritmo paralelo.

1. Introducción

Los algoritmos de agrupamiento son herramientas muy utilizadas en distintos contextos como la categorización de documentos [1], agrupamiento de genes y proteínas con similar función [2], seguimiento y detección de sucesos en un flujo continuo de noticias [3], segmentación de imágenes [4], entre otras.

El objetivo del agrupamiento es dado un conjunto de n objetos, descritos a través de m rasgos, crear particiones o cubrimientos de este conjunto. Los grupos formados deben cumplir que la semejanza de los objetos dentro de un agrupamiento sea máxima mientras que la semejanza entre los objetos pertenecientes a grupos diferentes sea mínima.

Los elementos esenciales a tener en cuenta para resolver el problema del agrupamiento son: el espacio de representación de los objetos, la medida de similaridad o semejanza entre objetos (no necesariamente debe ser una métrica) y el criterio agrupamiento o heurística a aplicar. En algunos de estos métodos es necesario defi además, una medida de semejanza entre los grupos, la cual se defi ne en términos de semejanzas entre los objetos que forman dichos grupos.

En muchas aplicaciones el conjunto de objetos a agrupar cambia en el tiempo, pues es necesario agregar nuevos objetos al conjunto o eliminar objetos que ya no existen. Ejemplos de estas aplicaciones son: la detección y seguimiento de sucesos en un continuo de noticias, el análisis del comportamiento de las facturas de una empresa, entre otras.

Los algoritmos de agrupamiento clásicos requieren todo el conjunto de datos realizar el agrupamiento, por tanto, cada vez que se agrega o elimina un objeto habría que realizar de nuevo el agrupamiento. Un paliativo muy usado para resolver situación es acumular un conjunto de cambios y en el horario de menor carga al sistema realizar el agrupamiento. Esta solución hace que se trabaje con datos no actualizados, por lo que se pueden obtener resultados falsos. Además, a medida que crece el conjunto de datos esto es más irrealizable por lo costoso que resulta. Por tanto, se requieren algoritmos que actualicen el agrupamiento a medida que cambia el conjunto de datos, sin necesidad de empezar nuevamente desde el principio. Estos algoritmos reciben nombre de algoritmos incrementales, dinámicos o en línea.

Con el creciente tamaño del conjunto de datos a agrupar, en muchas aplicaciones una sola computadora no puede resolver el problema del agrupamiento en un tiempo razonable. Además, muchas veces una computadora tampoco tiene la memoria necesaria para almacenar los datos que necesita el algoritmo, por lo que hay que recurrir datos en disco, lo que hace más lento aún el procesamiento. Esta es una situación característica donde se puede intentar resolver el problema explotando el poder de cómput y la memoria disponible en un conjunto de procesadores. Así, la paralelización de algoritmos de agrupamiento es la opción para obtener los grupos cuando se tiene conjuntos de datos muy grandes o se necesita utilizar algoritmos de agrupamiento elevada complejidad computacional.

Se han desarrollado paralelizaciones de algunos algoritmos de agrupamiento como por ejemplo, del jerárquico aglomerativo [5] y [6], del *K-Means* [7], del *MAFIA* entre otros.

En el trabajo se expone la paralelización de un algoritmo de agrupamiento incremental que busca las componentes conexas del grafo de β_0-semejanza asociado conjunto de datos y se hace un análisis de su complejidad computacional. El algoritm paralelo obtenido logra un adecuado balance de carga entre los procesadores al repart equitativamente los objetos entre los procesadores y lograr que cada procesador reali aproximadamente el mismo trabajo. Con este algoritmo paralelo se obtiene una efi cia teórica cercana a la unidad cuando la cantidad de objetos es mucho mayor que cantidad de procesadores.

El artículo se estructura como sigue. En la sección 2 se explican las caracterís cas del algoritmo de agrupamiento GLC. La sección 3 detalla el algoritmo parale propuesto, así como el análisis de su costo computacional. En la sección 4 present mos una valoración de los resultados experimentales obtenidos. Finalmente, damos

conclusiones de nuestro trabajo.

2. Algoritmos incrementales

Los algoritmos de agrupamiento incrementales realizan el agrupamiento a medida que van llegando las descripciones de los objetos. Por ello, están en desventaja respecto a los no incrementales que conocen previamente el universo de objetos a agrupar. Esto puede ocasionar que sean menos efi cientes que los no incrementales cuando todos los datos están disponible desde el comienzo (problema estático). Sin embargo, tienen sentido si son más efi cientes cuando el conjunto de datos cambia en el transcurso del tiempo (problema dinámico).

Se han desarrollado diversos algoritmos de agrupamiento incrementales como, por ejemplo, *Single-Pass* [8] y el algoritmo de las estrellas [9]. Estos algoritmos tienen como principal limitación que los grupos obtenidos dependen del orden de presentación de los objetos. Esto lo consideramos una caracteristica indeseable, pues los grupos existentes no dependen del orden en que se presenten los objetos, sino que sólo dependen de las características internas de los datos. Por ello, nuestro propósito fue paralelizar un algoritmo incremental en el que el agrupamiento resultante sea independiente del orden de los objetos.

2.1. Algoritmo GLC

GLC [10] es un algoritmo de agrupamiento incremental basado en grafos. Se llama grafo de semejanzas al grafo completo donde los vértices son los objetos a agrupar y las aristas se etiquetan con las semejanzas entre los objetos. Dos objetos cuya semejanza es mayor o igual que un cierto umbral β_0 (defi nido por el usuario) se denominan β_0-semejantes [11].

El algoritmo GLC obtiene de forma incremental las componentes conexas del grafo de β_0-semejanza asociado a un conjunto de objetos. El grafo de β_0-semejanza es un subgrafo del grafo de semejanzas donde se eliminan las aristas con peso menor que β_0. Con este algoritmo se garantiza que para todo objeto en un grupo existe al menos en dicho grupo otro objeto β_0-semejante con él.

Este algoritmo no necesita calcular toda la matriz de semejanzas entre objetos ni requiere tener en memoria dicha matriz, lo cual es una ventaja respecto a otros algoritmos que buscan componentes conexas, pues tiene menor complejidad espacial. Además, el algoritmo GLC permite trabajar con objetos descritos por variables cuantitativas y cualitativas mezcladas, incluso con ausencia de información y no restringe el uso de familias de métricas para realizar la comparación de los objetos. Otra característica del GLC es que nunca compara dos objetos más de una vez.

El algoritmo GLC almacena por cada grupo una lista de los objetos que pertenecen a él y por cada objeto su descripción. Cada vez que llega un nuevo objeto lo compara con los objetos de los grupos existentes. Si el nuevo objeto no fue β_0-semejante con ningún objeto de los grupos existentes se crea un nuevo grupo unitario con ese objeto. En caso contrario, todos los grupos para los cuales existe al menos un objeto

β_0-semejante con el nuevo objeto se unen y forman un nuevo grupo al que se agrega también el nuevo objeto. Para conocer los grupos formados en cualquier momento basta con recorrer la lista de grupos e imprimir sus miembros. Los pasos seguidos muestran en el algoritmo 1.

Algorithm 1 Algoritmo GLC

1. Tomar el siguiente objeto o
2. Hacer $L = \emptyset$, lista de grupos a unir
3. Para cada grupo existente g'
   a) Para cada objeto $o' \in g'$
      1) Si $Semejanza(o, o') \geq \beta_0$
      Agregar g' a L e ir a 3.
4. Si $L = \emptyset$ crear un nuevo grupo g y agregar en él a o
   Sino, unir todos los grupos de L en un grupo y agregar en él a o
5. Ir a 1.

Este algoritmo tiene un costo temporal en el peor de los casos de $T_{sec} = t_c n^2 r$ (t_c es una constante que depende del procesador), pues por cada nuevo objeto hay calcular su semejanza con los restantes. No obstante, es bueno destacar que en algoritmo, en muchos casos, no se requiere comparar al nuevo objeto con todos objetos de los grupos existentes. Esto se explica debido a que al encontrar en un grup existente un objeto β_0-semejante con el nuevo objeto, no se necesita comparar con restantes objetos de dicho grupo.

El costo espacial del GLC es $E_s = cnm$ (c es una constante que depende del de los rasgos de los objetos), pues sólo hay que almacenar las descripciones de objetos y los objetos que están en cada grupo.

3. Paralelización

Existen varios paradigmas para implementar algoritmos paralelos como son la me ria compartida y el paso de mensajes [12]. Aunque con el modelo de memoria compa tida los programas son más fáciles de crear, presenta dos limitaciones fundamentale En primer lugar, requiere de un hardware menos escalable que en el modelo de so de mensajes. En segundo lugar, el hardware utilizado en los multiprocesadores memoria compartida, por lo general, es más caro. Por tanto el modelo más difundi es el de paso de mensajes.

En el modelo de paso de mensajes, cada procesador tiene su memoria local e cambia información con los restantes procesadores mediante mensajes a través de red de interconexión. El tipo de esta red de interconexión ejerce una gran influencia la posible efi ciencia a lograr en el algoritmo paralelo.

Para valorar los algoritmos paralelos suele utilizarse un modelo de costo que incluye constantes que dependen del procesador y de la red de comunicación [13]. Las más importantes son:

- t_c: tiempo para realizar un cómputo en coma flotante (flop).
- t_s: tiempo que tarda la red de comunicación en iniciar un envío de un dato.
- t_w: tiempo que tarda en transmitirse una palabra por la red de comunicación entre dos procesadores conectados directamente.
- s_w: tamaño de una palabra del procesador.

Para evaluar un algoritmo paralelo se defi nen las siguientes medidas básicas:

- T_{sec}: el costo temporal del mejor algoritmo secuencial para resolver el problema tratado.
- T_{comp}: el costo de cómputo del algoritmo paralelo.
- T_{comm}: el costo temporal de las comunicaciones del algoritmo paralelo.
- $T_{par} = T_{comp} + T_{comm}$: el costo temporal total del algoritmo paralelo.
- S_{par}: el costo espacial del algoritmo en cada procesador.

A partir de las medidas anteriores, algunas de las medidas más utilizadas para valorar un algoritmo paralelo son:

- Aceleración: $A = \frac{T_{sec}}{T_{par}}$, mide el incremento de velocidad por utilizar la computadora paralela. Idealmente crece linealmente con el número de procesadores.
- Efi ciencia :$E = \frac{T_{sec}}{pT_{par}}$, mide el grado de aprovechamiento del poder de cómputo paralelo. Tiene el valor de 1 en el caso ideal, cuando el algoritmo paralelo es p veces más rápido que el secuencial, donde p es el número de procesadores.

3.1. Algoritmo GLC paralelo

Debido a las ventajas mencionadas anteriormente, en nuestro trabajo utilizamos como paradigma de programación paralela el modelo de paso de mensajes. Dentro de este paradigma utilizamos la estrategia de particionado de datos y el esquema de maestro-esclavos. El procesador 0 actuará como maestro y los restantes procesadores como esclavos.

Para paralelizar el algoritmo GLC repartimos uniformemente los datos entre los procesadores de forma tal que el procesador i almacena la descripción del objeto j si el resto de la división de j entre p (cantidad de procesadores) es i. Con esto, tratamos de lograr un equilibrio de carga entre los procesadores, lo cual es una característica deseable en todo algoritmo paralelo.

Cada procesador mantiene una lista de grupos y, a su vez, de cada grupo almacena una lista de los objetos del procesador que pertenecen a ese grupo. No necesariamente

todos los procesadores contienen a todos los grupos. Un procesador sólo tendrá grupos a los que pertenecen sus objetos según la distribución de carga realizada.

Cuando un grupo contenga, al menos, un objeto β_0-semejante con el nuevo objeto diremos, por convenio, que el grupo es semejante con el nuevo objeto.

Cada vez que el procesador 0 recibe un nuevo objeto o lo difunde a todos los procesadores. Posteriormente, cada procesador verifi ca si el nuevo objeto es β_0-semejante con alguno de sus objetos. Al fi nalizar este proceso, todos los procesadores informan al procesador 0 cuáles de sus grupos son semejantes con o mediante una operación de comunicación global del tipo recogida. El procesador 0 procesa esta información realiza una operación de comunicación global de tipo difusión donde informa a todos los procesadores los grupos globales a los que fue semejante el nuevo objeto. Con información, cada procesador une a todos sus objetos de los grupos que fueron semejantes con o en un solo grupo. El procesador al que pertenece el nuevo objeto o (según la distribución de carga realizada) lo agrega al nuevo grupo unión formado. En el caso de que o no fue semejante con ninguno de los grupos existentes, en el procesador dueño de o se crea un nuevo grupo donde se incluye a o.

Los pasos seguidos se muestran en el algoritmo 2.

Es bueno destacar algunas características del algoritmo paralelo diseñado, como son:

1. Existe un equilibrio de carga entre los procesadores, pues cada procesador tiene aproximadamente $\frac{n}{p}$ objetos.

2. El cálculo de la semejanza entre el nuevo objeto y los objetos de los grupos istentes se realiza entre todos los procesadores. Cada procesador calcula aproximadamente la misma cantidad de semejanzas.

3. El procesador 0 actúa como maestro, pues él difunde los objetos entre los proce sadores y con la información obtenida entre todos los procesadores determin los grupos que fueron semejantes al nuevo objeto.

4. En un procesador, cuando se encuentra en un grupo un objeto β_0-semejante cc el nuevo objeto, ya no se compara con ningún otro objeto de ese grupo.

5. El algoritmo diseñado trata de lograr su objetivo minimizando la cantidad comunicaciones. Es conocido que para lograr paralelizaciones efi cientes quiere que la cantidad de cómputo sea superior a la cantidad de comunicacione

3.2. Análisis de los costos computacionales

En el algoritmo paralelo obtenido cada procesador almacena $\frac{n}{p}$ objetos. En el de los casos, por cada uno de los n objetos que llegan, en cada procesador se calcul. $\frac{n}{p}$ semejanzas entre objetos. El cálculo de cada semejanza entre un par de objetos un costo de m.

El costo de cómputo paralelo es, por tanto:

$$T_{comp} = \frac{t_c n^2 m}{p}$$

Algorithm 2 Algoritmo GLC paralelo

1. El procesador 0 toma el próximo objeto y difunde a todos los procesadores su descripción.
2. Cada procesador realizar:
   a) $ListaMarcados = \emptyset$.
   b) Recibir la descripción del nuevo objeto o.
   c) Desmarcar todos los grupos en la lista de grupos del procesador.
   d) Para cada grupo no marcado g' en la lista de grupos del procesador:
      1) Para cada objeto o' de g' hacer:
         - Calcular semejanza S entre o' y o.
         - Si $S \geq \beta_0$ hacer:
           Marcar g' e ir a 2 (d).
3. Recogida en el procesador 0 de los grupos marcados en cada procesador.
4. El procesador 0 procesa la lista para obtener todos los grupos semejantes con o.
5. Difusión desde el procesador 0 con la lista global de los grupos obtenidos en el paso anterior y el identifi cador id del nuevo grupo a formar.
6. Cada procesador que posea grupos en la lista global, une todos estos grupos en un nuevo grupo con identifi cador id.
7. Si es el procesador al que pertenece o
   a) Si ya creó el nuevo grupo id agregar a o en ese grupo
   Sino, crear un nuevo grupo unitario formado por o cuyo identifi cador será id
8. Ir a 1.

Por cada nuevo objeto se realizan dos difusiones y una recogida. La primera operación de difusión (paso 1) tiene un costo de $p(t_s + t_w m)$. Después se realiza una recogida (paso 3) con un costo de $p(t_s + t_w K_1)$, donde K_1 es la mayor cantidad de grupos semejantes al nuevo objeto existentes en un procesador. Posteriormente, se produce segunda difusión (paso 5), que tiene un costo de $p(t_s + t_w K_2)$, donde K_2 es la mayor cantidad de grupos semejantes al nuevo objeto de forma global. Note que $K_2 \geq K_1$.

En el análisis de estos costos hemos supuesto que las difusiones y la recogida efectúan mediante p mensajes, lo cual constituye el peor de los casos. Dependiendo de la red de interconexión que se utilice estas operaciones se pueden lograr con menor cantidad de mensajes. Por ejemplo, en una red de difusión la recogida se logra con mensajes pero la difusión se logra con un solo mensaje.

Por tanto, el costo de las comunicaciones, en el peor de los casos, es:

$$T_{comm} = np(3\,t_s + t_w(m + 2K_2))$$

El costo espacial es:

$$S_{par} = \frac{s_w nm}{p}$$

pues en cada procesador se almacenan los m rasgos de $\frac{n}{p}$ objetos.

El costo temporal del mejor algoritmo secuencial para resolver el problema tratado es:

$$T_{sec} = t_c n^2 m$$

Por tanto, concluimos que la eficiencia del algoritmo paralelo es:

$$E = \frac{t_c n^2 m}{t_c n^2 m + np^2(3\,t_s + t_w(m + 2K_2))}$$

Si consideramos que $t_c nm \gg t_s p^2$, entonces la eficiencia alcanzada con nuestro algoritmo paralelo tiende a 1. Esta suposición no es difícil de lograr, pues se intenta paralelizar un algoritmo, precisamente, cuando la cantidad de objetos es muy grande.

4. Resultados experimentales

El algoritmo propuesto se implementó utilizando el estándar de programación paralela de paso de mensajes MPI. Para la realización de los experimentos se utilizó un cluster de computadoras Pentium II con 256K de memoria cache, 600MHz y 128 de RAM. Estas computadoras están enlazadas por una red myrinet conectada mediante switcher. El sistema operativo utilizado fue Linux Suse y la versión de MPI empleada fue MPICH-1.2.2.

Para determinar los parámetros de esta máquina paralela se realizó el test del pon y se obtuvo que $t_s = 1{,}66E-4$ y $t_w = 5{,}34E-8$. Para la estimación propio del algoritmo GLC, es decir, del tiempo que demora el procesamiento objeto en el algoritmo secuencial se midió el tiempo de ejecución del algoritmo máquina del cluster y se llegó a que $t_c = 4{,}31E-7$. Utilizando el resultado anterior (Vea 3.2), se requiere entonces que $nm \gg 385p^2$ para obtener un algoritmo GLC paralelo eficiente en esta multicomputadora.

Para llevar a cabo el estudio experimental del algoritmo diseñado se generaron varios conjuntos de datos aleatorios. Los conjuntos de datos se realizaron variando la cantidad de objetos en el rango [10000,90000] con objetos de 2 y 8 rasgos. Las componentes de los objetos se generaron utilizando el generador de números aleatorios proporcionado en la librería C de GNU, variando los rangos en que se generan los números de forma tal que la densidad promedio de los conjuntos obtenidos es la misma para todos los conjuntos de datos.

Con los datos así obtenidos se ejecutaron las variantes secuencial y paralela del algoritmo, variando la cantidad de procesadores desde 2 hasta 16. Los resultados obtenidos se muestran en el cuadro 1. En el mismo, para cada combinación de cantidad de datos, dimensión de los datos y cantidad de procesadores se especifica en segundos el tiempo de cálculo obtenido.Como puede observarse, en todos los casos se logra que los tiempos de ejecución del algoritmo paralelo sean menores que los del secuencial.

		Procesadores								
Tamaño	Rasgos	1	2	4	6	8	10	12	14	16
10000	2	159	81	57	49	48	52	51	50	49
10000	8	352	111	69	60	54	59	56	54	55
30000	2	1532	720	500	424	386	387	371	360	354
30000	8	3315	966	645	508	435	428	410	394	375
50000	2	4436	2230	1424	1973	1102	1072	1007	986	963
50000	8	10061	2968	1768	2171	1197	1186	1088	1058	1005
70000	2	9529	4542	2932	2467	2195	2120	2006	1943	1873
70000	8	17275	5878	3472	2824	2393	2334	2147	2034	1939
90000	2	14160	8225	5612	4103	3618	3553	3343	3223	3119
90000	8	28150	10456	6523	4650	3975	3847	3564	3387	3239

Cuadro 1: Resultados obtenidos en la experimentación.

En la figura 1 están graficados los tiempos obtenidos contra el tamaño de los conjuntos de datos utilizando los conjuntos de datos con objetos de 8 rasgos. En la misma se muestran las curvas para algunas cantidades de procesadores. En ella puede observarse el comportamiento cuadrático del algoritmo GLC secuencial. Además se observa cómo a medida que aumenta la cantidad de procesadores disminuyen los tiempos de cómputo. Un comportamiento similar se observa con las restantes dimensiones y cantidades de procesadores.

La figura 2 muestra las aceleraciones (A) contra la cantidad de procesadores para todas las cantidades de objetos de 8 rasgos. Como vemos la aceleración tiene un comportamiento cuasilineal cuando el tamaño del problema es suficientemente grande. En este caso se observa un comportamiento cercano al ideal.

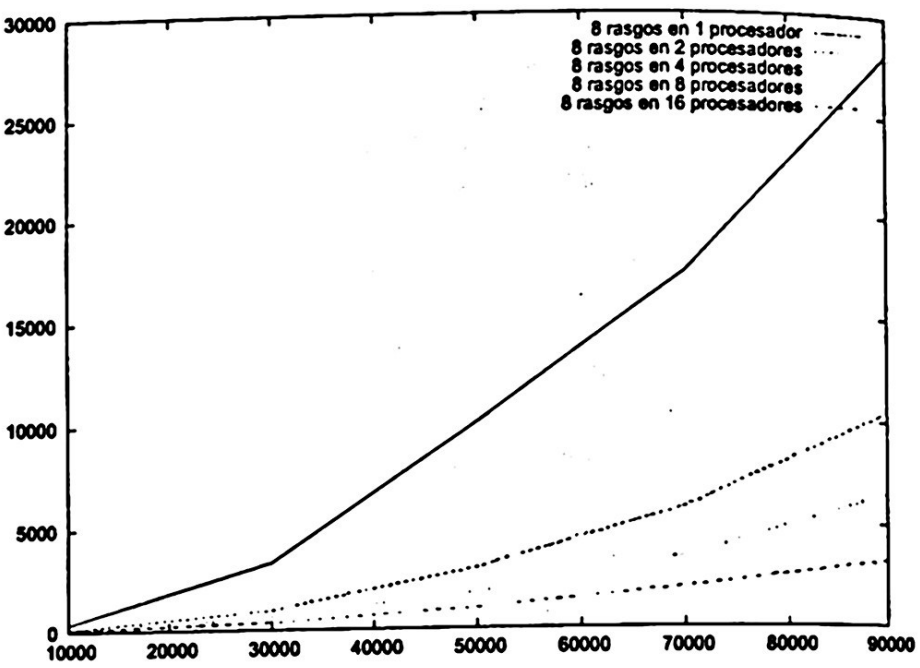

Figura 1: Tiempos obtenidos con objetos de 8 rasgos.

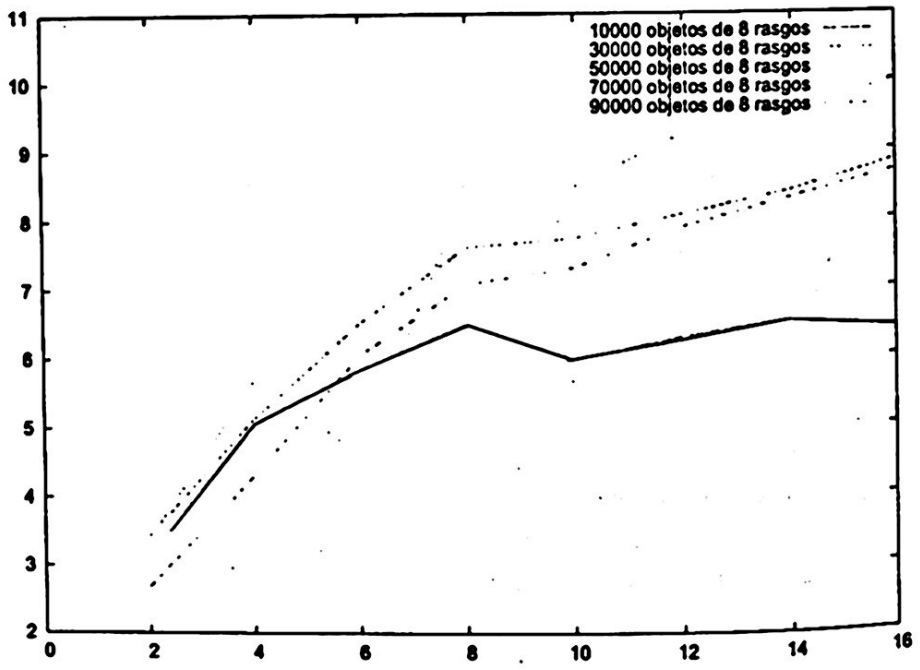

Figura 2: Aceleraciones para algunos tamaños del problema.

También puede observarse superaceleraciones cuando la cantidad de procesadores es pequeña y los conjuntos de datos son de pequeño o medio tamaño, lo que consideramos se debe a la influencia del caché del procesador. Esto puede ocurrir debido a que si al dividir los datos entre los procesadores, todos los datos del problema caben en la cache de los procesadores se aceleran los cálculos. También se observa un comportamiento anómalo del agrupamiento con 10000 objetos lo que consideramos se deba a algún factor externo que haya pertubado el comportamiento de la máquina paralela durante la ejecución de ese conjunto de datos.

5. Conclusiones

En el trabajo se expone la paralelización del algoritmo de agrupamiento incremental GLC y el análisis de su complejidad computacional. Como resultado se obtuvo una paralelización que aprovecha efi cientemente los recursos de una computadora paralela, pues se logra un buen equilibrio de carga entre los procesadores y una efi ciencia teórica ideal cuando la cantidad de objetos es mucho mayor que la cantidad de procesadores.

Los resultados experimentales obtenidos muestran las bondades del algoritmo paralelo propuesto al lograrse disminuir los tiempos de ejecución del algoritmo secuencial según aumenta el número de procesadores empleados. Estos resultados se corresponden a los resultados teóricos obtenidos, pues se logran aceleraciones cercanas a la cantidad de procesadores y éstas se incrementan en la medida en que aumentan la cantidad de objetos y la cantidad de rasgos que los describen.

Como trabajo futuro se preve profundizar en el estudio experimental del algoritmo propuesto sobre otras arquitecturas paralelas y con una mayor cantidad de procesadores. También pretendemos emplear el algoritmo paralelo diseñado para la detección de sucesos en grandes volúmenes de noticias periodísticas.

Referencias

[1] Greengrass, E. (2000). Information Retrieval: A Survey, *Technical Report.*

[2] Karypis, G. & Han, E, & Kumar, V. (1999). Chameleon: A hierarchical clustering algorithm using dynamic modeling, *IEEE Computer*: Special Issue on Data Analysis and Mining, Vol. 32, No. 8, 68-75.

[3] Allan, J. & Carbonell, J. & Doddington, G. &Yamron, J. & Yang, Y. (1998) Topic Detection and Tracking Pilot Study: Final Report, *Proceeding of DARPA Broadcast News Transcription and Understanding Workshop*, 194-218.

[4] Nagesh, H. & Goil, S. & Choudhary, A. (1999). MAFIA: Effi cient and Scalable Subspace Clustering for Very Large Data Sets, *Technical Report No. CPDC-TR-9906-010*, Center for Parallel and Distributed Computing.

[5] Gil-García, R. & Pons-Porrata, A. (2000). Parallel Agglomerative Hierarchical Clustering Algorithm in bus based networks, *Proceeding of V Ibero American Symposium of Pattern Recognition*, 691-696.

[6] Olson, C. (1995). Parallel algorithms for hierarchical clustering , *Parallel Computing* 21, 1313-1325.

[7] Dhillon, I &Modha, B. A. (2000). Data Clustering Algorithm On Distributed Memory Multiprocesor,Workshop on Large-scale Parallel KDD Systems, 1999, 245-260.

[8] Hill, D.R. (1968). A vector clustering technique, *Samuelson (ed.), Mechanized Information Storage, Retrieval and Dissemination*, North-Holland, Amsterdam.

[9] Aslam, J. & Pelekhov, K. & Rus, D. (1998). Static and Dynamic Information Organization with Star Clusters, *Proceedings of the 1998 Conference on Information Knowledge Management, CIKM 98, Baltimore, MD.*

[10] Sánchez, G. (2001). *Desarrollo de algoritmos para el agrupamiento de grandes volúmenes de datos mezclados*, Tesis Doctoral, Centro de Investigación en Computación, IPN, México.

[11] Shulcloper, J.R. & Martínez, J.F. (1995). *Clasificación Sin Aprendizaje y Con Aprendizaje Parcial (Enfoque Lógico Combinatorio)*, Centro de Investigación de Estudios Avanzados del I.P.N, Dpto. de Ingeniera Eléctrica, México.

[12] Wilkinson, B. & Allen, M. (1999). *Parallel programing: techniques and applications using networked workstations and parallel computers,* Prentice Hall.

[13] Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley Publishing Company.