# A Parallel Algorithm for Incremental Compact Clustering*

Reynaldo Gil-García[1], José M. Badía-Contelles[2], and Aurora Pons-Porrata[1]

[1] Universidad de Oriente, Santiago de Cuba, Cuba
{gil,aurora}@app.uo.edu.cu
[2] Universitat Jaume I, Castellón, Spain
badia@icc.uji.es

**Abstract.** In this paper we propose a new parallel clustering algorithm based on the incremental construction of the compact sets of a collection of objects. This parallel algorithm is portable to different parallel architectures and it uses the MPI library for message-passing. We also include experimental results on a cluster of personal computers, using synthetic data generated randomly and collections of documents. Our algorithm balances the load among the processors and tries to minimize the communications. The experimental results show that the parallel algorithm clearly improves its sequential version with large sets of data.

## 1 Introduction

Clustering algorithms are widely used for document classification, clustering of genes and proteins with similar functions, event detection and tracking on a stream of news, image segmentation and so on. Given a collection of $n$ objects characterized by $m$ features, clustering algorithms try to construct partitions or covers of this collection. The similarity among the objects in the same cluster should be maximum, whereas the similarity among objects in different clusters should be minimum. The clustering algorithms have three main elements, namely: the representation space, the similarity measure and the clustering criterion.

In many applications, the collection of objects is dynamic, with new items being added on a regular basis. An example of these applications is the event detection and tracking of streams of news. Classic algorithms need to know all the objects in order to perform the clustering and so, each time we modify the set of objects, it is necessary to cluster the whole collection again. Thus, we need algorithms able to update the clusters each time a new object is added to the data without rebuilding the whole set of clusters. This kind of algorithms are called incremental.

Many recent applications involve huge data sets that cannot be clustered in a reasonable time using one processor. Moreover, in many cases the data cannot

* This work was partially supported by the Spanish CICYT projects TIC 2002-04400-C03-01 and TIC 2000-1683-C03-03.

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 310–317, 2003.

be stored in the main memory of the processor and it is necessary to access the much slower secondary memory. A solution to this problem is to use parallel computers that can deal with large data sets and reduce the time of algorithms that can be very expensive. Parallel versions of some clustering algorithms have been developed, such as *Kmeans* [1], *MAFIA* [2], or *GLC* [3].

In this paper we propose a parallel incremental algorithm that finds the clusters as the compact sets in a collection of objects. This algorithm distributes the same number of objects in each processor and then balances the workload to perform in each one. The parallel algorithm was tested in a cluster of personal computers connected through a Myrinet network. Experimental results show a good behaviour of the parallel algorithm that clearly reduces the sequential time. Moreover, we have achieved near linear speedups when the collection of objects and the number of features are large.

The remainder of the paper is organized as follows. Section 2 describes the main features of the incremental compact algorithm. Section 3 includes the parallel algorithm. Section 4 shows the experimental results. Finally, conclusions are presented in Section 5.

## 2 Incremental Compact Algorithm

Incremental clustering algorithms update the clusters every time a new object arrives. They could be less efficient than the non-incremental algorithms if the whole collection is initially known. However, incremental algorithms are useful if they are more efficient when new objects are added in a regular basis. Some incremental algorithms have been developed, including, *Single-Pass* [4], *incremental K-Means* [5] and *Stars* algorithm [6]. The main drawback of these algorithms is that the obtained clusters depend on the arrival order of the objects. This is an undesirable characteristic because the clusters do not depend on the arrival order of the objects, but they only depend on the internal features of the data.

The incremental compact clustering algorithm [7] incrementally constructs the existing compact sets in a collection of objects. This algorithm is based on graphs and it produces disjoint clusters. Two objects are $\beta_0$-similars if their similarity is greater or equal to $\beta_0$, where $\beta_0$ is an user-defined parameter. We call maximum $\beta_0$-similarity graph ($max-S$) to the oriented graph whose vertices are the objects to cluster and there is an edge from vertex $o_i$ to vertex $o_j$, if $o_j$ is the most $\beta_0$-similar object to $o_i$. The compact sets are the connected components of the $max - S$ graph disregarding the orientation of the edges.

The compact algorithm stores the maximum $\beta_0$-similarity of each object and the set of objects connected to it in the $max - S$ graph. Every time a new object ($o$) arrives, its similarity with each object of the existing clusters is calculated and the graph is updated. The arrival of $o$ can change the current compact sets, because some new clusters may appear and others that already exist may disappear. Therefore, after updating the $max - S$ graph, the compact sets are rebuilt starting from $o$ and the objects in the compact sets that become

unconnected. The compact sets that do not include objects connected with $o$ remain unchanged.

During the graph updating task the algorithm constructs the following sets:

- *ClustersToProcess*: A cluster is included in this set if it has any object $o'$ that satisfies the following conditions: 1) $o$ is the most $\beta_0$-similar to $o'$ and the objects that were its most $\beta_0$-similars are not anymore. 2) $o'$ had at least two most $\beta_0$-similars objects or $o'$is the most $\beta_0$-similar to at least another object in this cluster. This set includes the clusters that could lose its compactness when the objects with the previous characteristics are removed from the cluster. Thus, these clusters must be reconstructed.
- *ObjectsToJoin*: An object $o'$ is included in this set if it satisfies: 1) $o$ is the most $\beta_0$-similar to $o'$ and the only object that was the most $\beta_0$-similar to $o'$ is not anymore. 2) $o'$ is not the most $\beta_0$-similar to any object of its cluster. The objects in this set will be included in the same compact set as $o$.
- *ClustersToJoin*: A cluster is included in this set if it is not in *ClustersToProcess* and it has at least one object $o'$ that satisfies one of the following conditions: 1) $o'$ is the most $\beta_0$-similar object to $o$. 2) $o$ is one of the most $\beta_0$-similar objects to $o'$, that is, $o$ is connected to $o'$ and no edge of $o'$ is broken in the $max-S$ graph. All the objects in *ClustersToJoin* will be included in the same compact set than $o$.

The main steps of the algorithm are the following:

1. Arrival of the new object $o$.
2. Updating of the $max-S$ graph.
   For each object $o_i$ in the existing clusters, its similarity with $o$ is calculated and the set of objects connected with $o_i$ and its maximum $\beta_0$-similarity are updated. Calculate the maximum $\beta_0$-similarity of $o$ and the set of objects connected with it. The sets *ClustersToProcess*, *ObjectsToJoin* and *ClustersToJoin* are constructed. Every time an object is added to *ObjectsToJoin* it is removed from the cluster in which it was located before.
3. Reconstruction of the compact sets.
   Let $C$ be a set including $o$ and all the objects included in the clusters in *ClustersToProcess*. Construct the existing compact sets in $C$ and add them to the existing cluster list.
   Add all the objects in *ObjectsToJoin* and all the objects included in the clusters in *ClustersToJoin* to the compact set including $o$. The clusters in *ClustersToProcess* and in *ClustersToJoin* are removed from the existing cluster list.

This algorithm is $O(n^2)$. Its main advantage over other incremental algorithms is that the generated set of clusters is unique, independently of the order of incoming objects. Besides, this algorithm makes no irrevocable cluster assignments. Thus, the mistakes made at the beginning, when little information is available, could be corrected. Another advantage of this algorithm is that it can deal with mixed incomplete object descriptions. On the other hand, the algorithm is not restricted to the use of metrics to compare the objects. It has been succesfully applied to the online event detection in a collection of digital news [8].

## 3 Incremental Compact Parallel Algorithm

Our parallel algorithm uses a message-passing architecture and a master-slaves model, where one of the processors acts as the master during some phases of the algorithm. The data is distributed among the processors, so that processor $i$ stores the description of object $j$ if $i = j \ mod \ p$. This data partition tries to balance the load among the processors in order to improve the efficiency of the parallel algorithm. Each processor also stores the maximum $\beta_0$-similarity of each of its objects and the indexes of the objects connected to it in the $max-S$ graph. Besides, each processor maintains a list of clusters (compact sets) and a list of its objects in each cluster. With this distribution of the data, a processor only stores information about the clusters containing any of its objects.

When a new object $o$ arrives, each processor computes the similarity of all its objects with $o$ and updates its part of the graph. This updating process requires the exchange of some information with other processors. The clusters that can lose their compactness are determined during this process. While updating the graph, each processor constructs the sets *ClustersToProcess*, *ObjectsToJoin* and *ClustersToJoin* in the same way as they are constructed in sequential algorithm. Besides, each processor constructs the set *LostEdges*, that contains the pairs of objects $(o', o'')$, where $o'$ is an object of the processor, $o''$ is an object of another processor and the edge between them in the $max - S$ graph is broken.

Then the compact sets are reconstructed starting from the new object and the objects included in the clusters of *ClustersToProcess*. Each processor, using its objects in the clusters of *ClustersToProcess*, constructs the compact subsets that can be formed. For each one of these subsets, a new set called *Arrival* is constructed. This set contains all the objects that do not belong to the compact subset but are connected to an object in this subset. Observe that, the set *Arrival* will only contain objects of other processors. Afterwards, processor 0 constructs the new compact sets using the compact subsets in all the processors and their associated *Arrival* sets. If a compact subset contains at least one object included in the *Arrival* set of another compact subset, then both subsets are merged into the same compact set and its associated *Arrival* set is formed. The main steps of the parallel algorithm are the following:

1. Processor 0 broadcasts the description of the new object $o$.
2. Updating of the maximum $\beta_0$-similarity graph.
   - On each processor do:
     For each object in the processor, its similarity with $o$ is calculated and its maximum $\beta_0$-similarity and the set of objects connected to it in the $max - S$ graph are updated. The sets $ClustersToProcess$, $ObjetsToJoin$ and $LostEdges$ are constructed. Every time an object is added to $ObjectsToJoin$, it is removed from the cluster in which it was located before. Construct the set $ClustersToJoin$ by including the clusters with objects in the processor for which $o$ is added to its set of maximum $\beta_0$-similarity objects. Construct the set $To$ which contains the objects in the processor for which $o$ is its most $\beta_0$-similar, and construct the set $From$

which contains the objects in the processor that are the most $\beta_0$-similar to $o$. Also compute *MaxSem*, the maximum $\beta_0$-similarity to $o$ of any of the objects in the processor.
- Join the sets *ClustersToProcess* and *LostEdges* of each processor in order to form *GlobalClustersToProcess* and *GlobalLostEdges*.
- The processor in which $o$ is stored gathers the sets $From$ and $To$ and the values $MaxSem_i$, calculates $MaxSem_o = \max\{MaxSem_i\}$, $i = 1, ..., p$, stores the edges of $o$ and broadcasts $MaxSem_o$.
- Each processor updates *ClustersToJoin* and its edges with $o$ using $From$, $MaxSem_o$, *GlobalClustersToProcess* and *GlobalLostEdges*.

3. Reconstruction of the compact sets.
   - On each processor do:
     $C$ = set of objects in the clusters included in *GlobalClustersToProcess*. If the processor owns $o$, add $o$ to the set $C$. Form the existing compact subsets in $C$ and their associated *Arrival* sets.
   - Join the sets *ClustersToJoin* of each processor in order to form *GlobalClustersToJoin*.
   - Processor 0 gathers all the compact subsets and their associated *Arrival* sets. It constructs the compact sets and broadcasts them.
   - Each processor updates its clusters and adds the objects in *GlobalClustersToJoin* or in *ObjectsToJoin* to the cluster including $o$. The clusters in *GlobalClustersToJoin* or in *GlobalClustersToProcess* are removed.

Some of the main features of our parallel algorithm are the following:

1. The load is balanced because each processor works with approximately $\frac{n}{p}$ objects and it computes approximately the same quantity of similarities. Also the $max - S$ graph is distributed and it is updated in parallel.
2. The compact sets that change due to the arrival of the new object are constructed in parallel. The construction of the sets *GlobalClustersToProcess* and *GlobalLostEdges* are carried out in parallel. The construction of the set *GlobalClustersToJoin* and the compact sets are carried out in parallel too.

## 4 Performance Evaluation

The target platform for our experimental study is a cluster of personal computer connected through a Myrinet network. The cluster consists of 32 Intel Pentium II-300MHz processors, with 128 Mbytes of SDRAM each one. The communication switch has a latency of 16.1 milliseconds and the bandwidth is 1 Gb/sec. The algorithms have been implemented on a Linux operating system, and we have used an used a specific implementation of the message-passing library MPI that offers small latencies and high bandwidths on the Myrinet network. We have executed the parallel algorithm varying the number of processors from 2 to 20.

First, we have performed an experimental analysis of the parallel algorithm using synthetic data generated randomly in $\mathbb{R}^m$. The number of objects varies

in 10000, 50000 and 200000. The number of features of the objects varies in 2, 8, 16, 64 and 128 and $\beta_0 = 0.5$. The values of the features have been generated so that the average density of the synthetic sets is the same for all data sets. The similarity measure used was $Sim(o_i, o_j) = \frac{1}{1+D(o_i,o_j)}$, where $D(o_i, o_j)$ is the euclidean distance between $o_i$ and $o_j$.

Another experimental analysis was carried out using two document collections of articles published in the Spanish newspaper "El País" during June 1999. The first collection consist of 554 international articles. The second contains 2463 articles belonging to the sections Opinion, Health, National, International, Culture and Sports. The documents are represented using the traditional vectorial model. The terms of documents represent the lemmas of the words appearing in the texts. Stop words, such as articles, prepositions and adverbs are disregarded from the document vectors. Terms are statistically weighted using the normalized term frequency (TF). These data sets are of very high dimensionality. In our experiments the $\beta_0$ threshold used was 0.25 and to compare the documents we use the cosine measure.

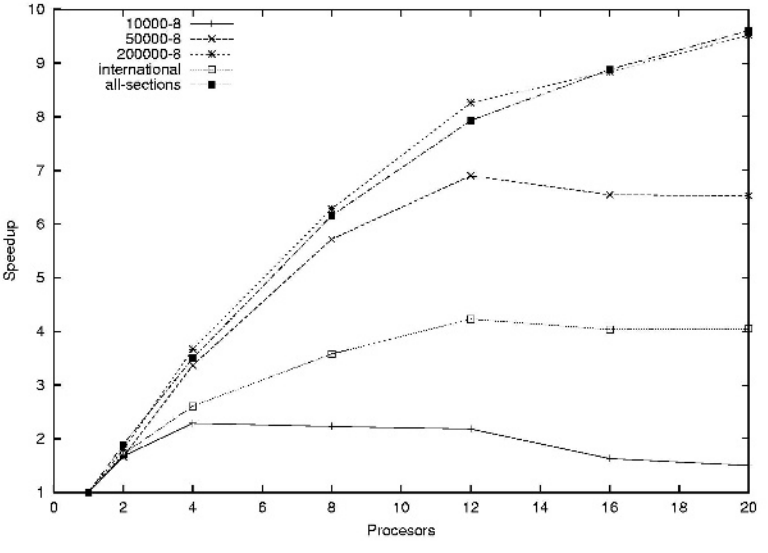


**Fig. 1.** Speedups of the synthetic and document data sets.

Figure 1 shows the speedup behavior of the document data sets and the synthetic data when its size is varied in a 8 dimensional data space. From the plot it can be seen that we have achieved near linear speedups for up to a certain number of processors depending on the data size. This is due to the algorithm being heavily data parallel and the computation time decreasing linearly with the increase in the number of processors. Parallelism reduces computation time for up to a certain number of processors depending on the data size, after which

communication overhead overshadows computational gain. The higher the data size, the greater the speedup for the same number of processors. Although the document data size is small compared to the our synthetic data sets, the obtained speedups are larger. We think that this behaviour is due to the computation time of the similarity measure between documents being much greater than the computation time of the similarity measure between objects.

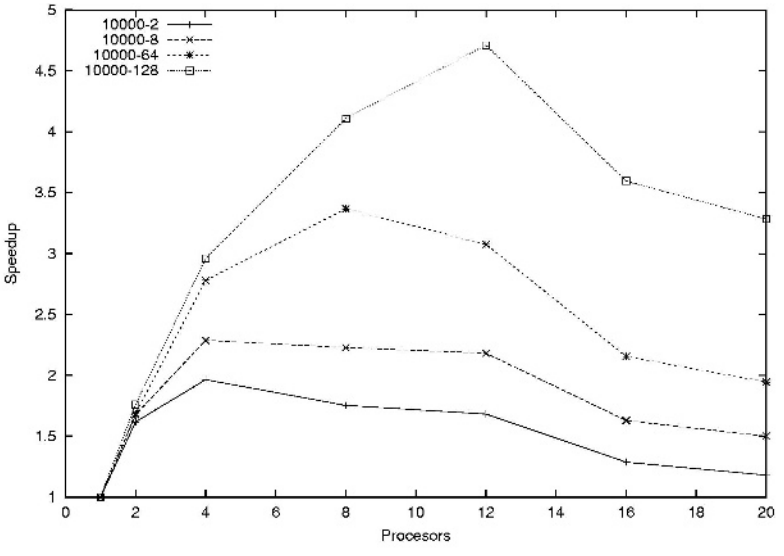


**Fig. 2.** Speedups of the synthetic data with a fixed data size.

Further, we set the data size and vary the dimensionality of the data. As it can be noticed in Figure 2, the speedup increases with the growth of the data dimensionality for the same number of processors. The data size taken in this experiment was small. If a similar experiment with larger data size is carried out, greater speedups are obtained and the maximum speedup for a fixed data is reached in a bigger number of processors.

## 5 Conclusions

In this paper we presented an efficient parallel algorithm that implements an incremental method to determine the compact sets in a collection of objects. The generated set of clusters is unique, independently of the objects arrival order of the objects. Another advantage of this algorithm is that it can deal with mixed incomplete object descriptions. On the other hand, the algorithm is not restricted to the use of metrics to compare the objects.The proposed parallel algorithm can be used in many problems such as the event detection

tasks, in the knowledge discovery problems and others. Besides, the resulting parallel algorithm is portable, because it is based on standard tools, including the message-passing library MPI.

We have implemented and tested the parallel code in a cluster of personal computers. The experimental evaluations on a variety of synthetics and real data sets with varying dimensionality and data sizes show the gains in performance. The obtained results show a good behaviour of the parallel algorithm that clearly reduces the sequential time. Moreover, we have achieved near linear speedups when the collection of objects and the number of features are large. The main reason for this behaviour is that we have tried to minimize the communications and to balance the load on the processors by carefully distributing the objects and the tasks that each processor performs during each step of the algorithm.

## References

1. Dhillon, I. and Modha, B. A.: Data Clustering Algorithm on Distributed Memory Multiprocessor. Workshop on Large-scale Parallel KDD Systems (2000) 245–260.
2. Nagesh, H., Goil, S. and Choudhary, A.: A Scalable Parallel Subspace Clustering Algorithm for Massive Data Sets. International Conference on Parallel Processing (2000) 447–454.
3. Gil-García, R. and Badía-Contelles, J.M.: GLC Parallel Clustering Algorithm. In Pattern Recognition. Advances and Perspectives. Research on Computing Science CIARP'2002 (In Spanish), México D.F., November (2002) 383–394.
4. Hill, D. R.: A vector clustering technique. Samuelson (ed.), Mechanized Information Storage, Retrieval and Dissemination, North-Holland, Amsterdam (1968).
5. Walls, F., Jin, H., Sista, S. and Schwartz, R.: Topic Detection in Broadcast news. Proceedings of the DARPA Broadcast News Workshop (1999) 193–198.
6. Aslam, J., Pelekhov, K. and Rus, D.: Static and Dynamic Information Organization with Star Clusters. Proceedings of the 1998 Conference on Information Knowledge Management CIKM 98, Baltimore, MD (1998).
7. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R. and Santiesteban-Alganza, Y.: An Incremental Clustering Algorithm to find partitions with mixed data. In Pattern Recognition. Advances and Perspectives. Research on Computing Science, CIARP'2002 (In Spanish). México D.F., November (2002) 265–276.
8. Pons-Porrata, A., Berlanga-Llavori, R. and Ruiz-Shulcloper, J.: Detecting events and topics by using temporal references. Lecture Notes in Artificial Intelligence 2527, Springer Verlag (2002) 11–20.