Parallel Algorithm for Extended Star 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 present a new parallel clustering algorithm based on the extended star clustering method. This algorithm can be used for example to cluster massive data sets of documents on distributed memory multiprocessors. The algorithm exploits the inherent data-parallelism in the extended star clustering algorithm. We implemented our algorithm on a cluster of personal computers connected through a Myrinet network. The code is portable to different architectures and it uses the MPI message-passing library. The experimental results show that the parallel algorithm clearly improves its sequential version with large data sets. We show that the speedup of our algorithm approaches the optimal as the number of objects increases.

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 involve three main elements, namely: the representation space, the similarity measure and the clustering criterion.

One of the most important problems in recent years is the enormous increase in the amount of unorganized data. Consider, for example, the web or the flow of news in newspapers. We need methods for organizing information in order to highlight the topic content of a collection, detect new topics and track them. The star clustering algorithm [1] was proposed for these tasks, and three scalable extensions of this algorithm can be found in [2]. The star method outperforms existing clustering algorithms such as single link, average link and k-means in the organizing information task, as it can be seen in [1]. However, the clusters obtained by this algorithm depend on the data order. In [3] we proposed a

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

new clustering algorithm that outperforms the Aslam's algorithm and it is also independent of the data order.

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 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 spent by the algorithms. Parallel versions of some clustering algorithms have been developed, such as *K-Means* [4], *MAFIA* [5], *GLC* [6] and the Incremental Compact Algorithm [7].

In this paper we present a new parallel algorithm that finds the clusters obtained by the Extended Star Clustering Algorithm [3]. This algorithm distributes the same number of objects to each processor and balances the workloads in average. The parallel algorithm was tested in a cluster of personal computers connected through a Myrinet network. Experimental results show a good behavior 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 sequential algorithm. Section 3 shows the parallel algorithm. Section 4 includes the experimental results. Finally, conclusions are presented in Section 5.

2 Extended Star Clustering Algorithm

Two objects are β_0-similar if their similarity is greater or equal to β_0, where β_0 is a user-defined parameter. We call β_0-similarity graph the undirected 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 β_0-similar to o_i. Finding the minimum vertex cover of a graph is a NP complete problem. This algorithm is based on a greedy cover of the β_0-similarity graph by star-shaped subgraphs. A star-shaped subgraph of $l+1$ vertices consists of a single *star* and l *satellite vertices*, where there exist edges between the star and each of the satellite vertices.

On the other hand, we define the complement degree of an object o is the degree of o taking into account its neighbors not included yet in any cluster, namely:

$$CD(o) = |N(o) \setminus Clu|$$

where Clu is the set of objects already clustered and $N(o)$ is the set of neighbors of the object o in the β_0-similarity graph. As we can see, the complement degree of an object decreases during the clustering process as more objects are included in clusters.

In the extended star clustering algorithm the stars are the objects with highest complement degree. The isolated objects in the β_0-similarity graph are also stars. The algorithm guarantees a pairwise similarity of at least β_0 between the star and each of the satellite vertices, but such similarity is not guaranteed

Algorithm 1 Extended star clustering algorithm.

```
1. Calculate all the similarities between each pair of objects to build the $\beta_0$-similarity graph
2. Let $N(o)$ be the neighbors of each object $o$ in the $\beta_0$-similarity graph
3. For each isolated object $o$ ($|N(o)| = 0$): Create the singleton cluster $\{o\}$
4. Let $L$ be the set of non-isolated objects
5. Calculate the complement degree of each object in $L$
6. While a non-clustered object exists:
   (a) Let $M_0$ be the subset of objects of $L$ with maximum complement degree
   (b) Let $M$ be the subset of objects of $M_0$ with maximum degree
   (c) For each object $o$ in $M$:
       i. If $\{o\} \cup N(o)$ does not exist, create a cluster $\{o\} \cup N(o)$
   (d) $L = L \setminus M$
   (e) Update the complement degree of the objects in $L$
```

between satellite vertices. The main steps of our algorithm are shown in the Algorithm 1.

The complexity time of the algorithm is $O(n^2m)$ [3]. This algorithm creates overlapped clusters. Unlike the original star clustering algorithm, the obtained clusters are independent of the data order. Besides, the selection of stars using the complement degree allows the algorithm to cover quickly the data and it reduces the overlapping among the clusters.

In [3] we compare the extended star clustering algorithm with the original star algorithm in several subcollections of TREC data[1] using the F1 measure [8]. It obtains a better cluster quality in these subcollections in most cases.

3 Parallel Algorithm

Our parallel algorithm is based on the Single Program Multiple Data (SPMD) model using message passing, which is currently the most prevalent model on distributed memory multiprocessors. We assume that we have p processors each with a local memory. These processors are connected using a communication network. We do not assume a specific interconnection topology for the communication network, but the access time to the local memory of each processor must be cheaper than time to communicate the same data with other processor.

This algorithm uses a master-slaves model, where one of the processors acts as the master during some phases of the algorithm. The data is cyclically distributed among the processors, so that processor i owns the object j if $i = j \ mod \ p$. This data partition tries to balance the workload among the processors in order to improve the efficiency of the parallel algorithm. Each processor stores the indexes of the objects connected to its $\frac{n}{p}$ objects in the β_0-similarity graph.

Initially, the algorithm builds the β_0-similarity graph, and each processor creates the singleton clusters formed by its isolated objects. Then it spends most

[1] `http://trec.nist.gov`

Algorithm 2 Parallel algorithm.

```
1. Build_$\beta_0$-similarity_graph()
2. On each processor:
   (a) For each isolated object $o$: Create the singleton cluster $\{o\}$
   (b) Let $L$ be the set of non isolated objects in this processor
3. Processor 0 gathers the singleton clusters
4. Processor 0 broadcasts the number of non-clustered objects
5. While there exist non-clustered objects
   (a) Find_stars()
   (b) Build_clusters()
   (c) Update_complement_degree()
   (d) On each processor: $L = L \setminus M$
```

Algorithm 3 Build_β_0-similarity_graph()

```
1. Processor 0 broadcasts the number of objects $n$
2. On each processor:
   (a) For $i = 1, ..., n$:
       i. If processor 0: read the object $o_i$ and broadcast it
       ii. If processor owns $o_i$: store its description
       iii. Calculate the similarities of its objects with $o_i$ to build the $\beta_0$-similarity
            subgraph
   (b) Let $N(o)$ be the neighbors of each object $o$ in the $\beta_0$-similarity graph. The
       complement degree of $o$ is $|N(o)|$
```

of the time of the algorithm to find the star shaped clusters. The algorithm terminates when all objects are clustered. The main steps of our parallel algorithm are shown in the Algorithm 2.

The parallel algorithm involves four major steps: building the β_0-similarity graph, finding of stars, constructing the clusters, and updating the complement degree. Each of the above four steps is carried out in parallel. The algorithms *Build_β_0-similarity_graph*, *Find_stars*, *Build_clusters* and *Update_complement_degree* describe each of these steps.

The *Build_β_0-similarity_graph*() algorithm is embarrassingly parallel. Observe that the similarity calculations are inherently data parallel, that is, they can be executed asynchronously and in parallel for each object. Therefore a perfect workload balance is achieved.

In the *Find_stars*() algorithm each processor starts by finding the candidate stars (step 1). Given the local maximum complement degree and the local maximum degree on each processor, the *Reduce* communication operation computes the global maxima and broadcasts them to all processors. For example, if $(2, 6)$, $(3, 4)$ and $(3, 5)$ are the maximum complement degrees and the maximum degrees calculated in the processors 0, 1 and 2 respectively, the obtained global maxima are $(3, 5)$. Notice that the global maximum degree is the maximum de-

Algorithm 4 Find_stars()

```
1. On each processor:
   (a) Let $M_0$ be the subset of objects of $L$ with maximum complement degree
   (b) Let $M$ be the subset of objects of $M_0$ with maximum degree
2. Reduce communication to get the global maximum complement degree and its
   corresponding global maximum degree
3. On each processor:
   (a) If its local maximum complement degree and its local maximum degree coincide
       with the global maxima, the stars are the objects of $M$. Else, $M = \emptyset$
```

Algorithm 5 Build_clusters()

```
1. On each processor:
   (a) For each star object $o$ of $M$, build the cluster with $o$ and its neighbors $N(o)$
   (b) Determine the number of non-clustered objects
2. Reduce communication to obtain the global clusters and the global number of
   non-clustered objects
```

gree corresponding to the global maximum complement degree. Finally, the stars are found by the processors whose maxima coincide with the global ones.

The algorithm 5 builds the clusters from the obtained stars. This process may lead to identical clusters being formed. We need to identify the repeated clusters and retain only one of them. Elimination of identical clusters is carried out on each processor and also on the *Reduce* communication operation. The elimination on each processor is not needed, but it decreases greatly the time during the *Reduce* operation. Finally, both the global clusters and the global number of non-clustered objects are broadcast to all processors.

The clusters built in this iteration, incorporate some objects that were not even clustered, that is, the objects that did not belong to any cluster built in previous iterations. The algorithm 6 firstly constructs the set of these objects.

Algorithm 6 Update_complement_degree()

```
1. On each processor:
   (a) Let $C$ be the set of its objects clustered in this iteration
   (b) Build the set of pairs $(o, v)$, where $o$ is a neighbor of an object of $C$ and $v$ is
       the value in which its complement degree must be decreased. $v$ is the number
       of objects of $C$ that are neighbors of $o$.
2. Reduce communication to get the global set of pairs $(o, v)$
3. On each processor:
   (a) For each object $o$ of the global set of pairs $(o, v)$
       i. If this processor owns $o$, decrease its complement degree in $v$
```

The neighbors of these objects change its complement degree and therefore we need update them. For this purpose, we build a data structure that contains these neighbors and the value in which its complement degree must be decreased. Since each processor only knows the complement degree decreases due to its objects clustered in this iteration, a *Reduce* communication operation is performed so that all processors have the global decreases. Finally, each processor updates the complement degree of its objects.

As we can see, the tasks are divided among the processors such that each processor gets approximately an equal amount of work when the number of objects is large. The description of objects and the β_0-similarity graph are fairly distributed among the processors. Thus the memory used by the parallel algorithm is similar in each processor. On the other hand, the overhead of communication is reduced by packing several informations in a single message.

4 Performance Evaluation

The target platform for our experimental analysis is a cluster of personal computer connected through a Myrinet network. The cluster consists of 34 Intel Pentium IV-2GHz processors, with 1 Gbyte of RAM each one. The algorithm has been implemented on a Linux operating system, and we have used a specific implementation of the MPI message-passing library that offers small latencies and high bandwidths on the Myrinet network. We have executed the parallel algorithm varying the number of processors from 1 to 32.

We used data (in Spanish) from the TREC-4 and TREC-5 conferences as our testing medium. The TREC-4 collection contains a set of "El Norte" newspaper articles in 1994. This collection has 5828 articles classified in 50 topics. The TREC-5 consists of articles from AFP agency in 1994-1996 years, classified in 25 topics. We only used the data from 1994, for a total of 695 classified articles. The TREC-4 document collection were partitioned in three subcollections to evaluate the performance of the parallel algorithm using data sets with different sizes. The documents are represented using the traditional vectorial model. Terms are statistically weighted using the normalized term frequency. Moreover, we use the traditional cosine measure to compare the documents. The obtained results are shown in table 1.

The clusters obtained with the parallel algorithm are independent of the number of processors involved. As we can see, the parallel algorithm clearly

Table 1. Experimental results.

Time (sec.)		Number of processors									
Collection	Size	1	2	4	8	12	16	20	24	28	32
afp	695	9.22	4.74	2.83	1.50	1.47	1.22	1.30	1.60	1.58	1.14
eln-1	1380	56.27	29.16	15.85	8.90	6.78	5.82	5.35	5.10	4.72	4.66
eln-2	2776	232.83	119.43	62.15	32.48	23.59	18.93	16.62	14.48	13.88	13.01
eln-3	5552	972.37	494.67	255.20	129.89	90.06	70.74	60.85	51.32	46.23	41.82

reduces the sequential time in all collections. The time reductions are larger as we increase the size of the data sets.

Figure 1 shows the speedups obtained with the parallel algorithm using different data sets. 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. The computation time decreases almost linearly with the number of processors except in the smaller data (afp). Besides, when we deal with few objects, the number of stars and neighbors per processor could be quite different, and so the workload per processor could be unbalanced. However, with large data sets, all processor should have a similar number of stars, thus balancing the workload. On the other hand, the effect of the communication time is smaller as we increase the size of the data sets. Therefore, the higher the data size, the greater the speedup for the same number of processors.

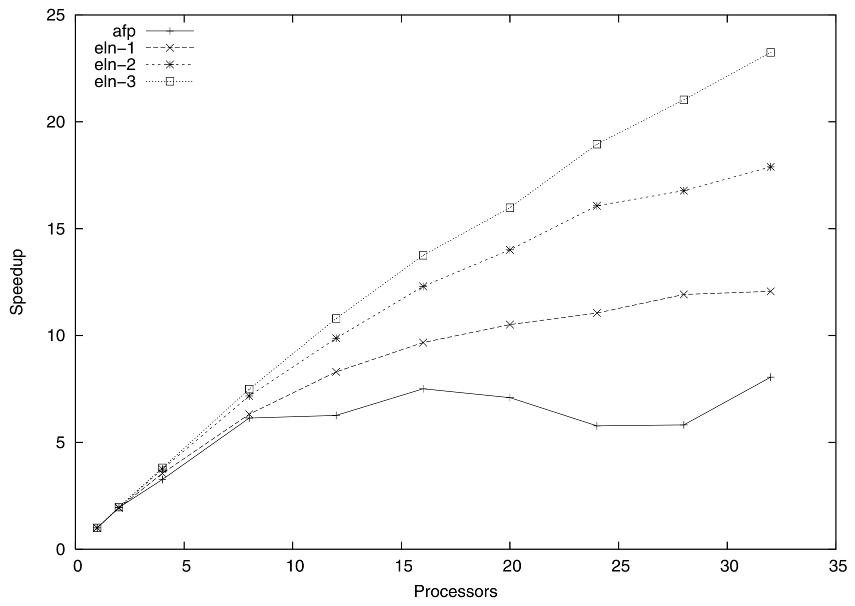

Fig. 1. Speedups on TREC data.

5 Conclusions

In this paper we present a parallel extended star clustering algorithm. The generated set of clusters is unique, independently of the arrival order of the objects. Another advantage of this algorithm is that it can deal with mixed incomplete object descriptions and it obtains overlapped clusters. 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 applications such as information organization, browsing, filtering, routing and topic detection and tracking. Besides, the resulting parallel algorithm is portable, because it is based on standard tools, including the MPI message-passing library.

We have implemented and tested the parallel code in a cluster of personal computers. The experimental evaluations on TREC data show the gains in performance. The obtained results show a good behavior of the parallel algorithm that clearly reduces the sequential time. Moreover, we have achieved near linear speedups with large data sets. The main reason for this behavior 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. Aslam, J.; Pelekhov, K. and Rus, D.: Static and Dynamic Information Organization with Star Clusters. In *Proceedings of the 1998 Conference on Information Knowledge Management*, Baltimore, MD, 1998.
2. Aslam, J.; Pelekhov, K. and Rus, D.: Scalable Information Organization. In *Proceedings of RIAO*, 2000.
3. Gil-García, R. J.; Badía-Contelles, J. M. and Pons-Porrata, A.: Extended Star Clustering Algorithm. *In Proceedings of the 8th Iberoamerican Congress on Pattern Recognition,* LNCS 2905, Springer Verlag, pp. 480-487, 2003.
4. Dhillon, I. and Modha, B. A.: Data Clustering Algorithm on Distributed Memory Multiprocessor. *Workshop on Large-scale Parallel KDD Systems*, pp. 245-260, 2000.
5. Nagesh, H.; Goil, S. and Choudhary, A.: A Scalable Parallel Subspace Clustering Algorithm for Massive Data Sets. *International Conference on Parallel Processing*, pp. 447-454, 2000.
6. Gil-García, R. and Badía-Contelles, J.M.: GLC Parallel Clustering Algorithm. In *Pattern Recognition. Advances and Perspectives. Research on Computing Science* (In Spanish), pp. 383-394, 2002.
7. Gil-García, R. J.; Badía-Contelles, J. M. and Pons-Porrata, A.: A Parallel Algorithm for Incremental Compact Clustering. *In Proceedings of the Europar2003*, LNCS 2790, Springer-Verlag, pp. 310-317, 2003.
8. Larsen, B. and Aone, C.: Fast and Effective Text Mining Using Linear-time Document Clustering. In *KDD'99*, San Diego, California, pp. 16-22, 1999.