10th Annual
Conference of Computer Society of Iran
February 15-17, 2005

Parallel Subspace Clustering

Hamid Nazerzadeh Mohammad Ghodsi Saba Sadjadian

Sharif University of Technology
{nazerzadeh, ghodsi, sadjadian}@ce.sharif.ir

Abstract

Subspace clustering is an extension of traditional clustering that discovers clusters respected to the different subspaces within a data-set. The time complexity of the algorithms to explore the high dimensional spaces and find clusters in subspaces is exponential in the dimensionality of the data and is thus extremely computationally intensive. Therefore, employing parallel algorithm for this problem can reduce its time complexity. In this paper, we propose an architecture for parallel implementation of the *Locally Adaptive Clustering Algorithm.* We demonstrate the efficiency of our approach through analysis and experimental evaluation.

Keywords: parallel algorithms, data clustering, PRAM model

1 Introduction

The enlarging masses of information emerging by the progress of technology, makes clustering of very large masses of high dimensional data a challenging task. This is a data mining task that is widely used in many fields, such as pattern recognition, image processing and analysis of genes in bioinformatics [1], that concern with grouping similar data. Fundamentally, finding the homogeneous groups of data is the main point of clustering. The resemblance between the data in one cluster is based on different similarity measures. The data is mostly represented as vectors of measurements or points in multi-dimensional space. Calculating the distance measures over the various dimensions in a data-set can be considered as similarity measures which in some applications is a computationally intensive task.

Recently, many researches have shown interest in the problem of high dimensional data clustering and have proposed several solution approaches. But, the curse of dimensionality is what these methods suffer from. As the number of dimensions of the data increases, the distance measures calculated from the distances of attributes over dimensions become meaningless. In such spaces, for each pair of data, you can find at least one dimension where the points are far from each other on that dimension.

A potential solution to this problem is the use of *Subspace Clustering* which seeks to discover the set of objects in the data-set that are correlated along some of dimensions. CLIQUE [2] proposed as a subspace clustering finds the dense regions in each subspace through the bottom-up searching methods; however, the resulted clusters are not disjoint. Performing a top-down searching, PROCLUS [3], is another algorithm which partitions data in to disjoint groups.

Local Adaptive Clustering, or LAC [4], is another subspace clustering algorithm that works differently from the traditional clustering algorithms (e.g. CLIQUE, PROCLUS) which are based on the idea of feature selection. LAC, assigns weights to features according to the local similarities of the data along dimensions in such a way that features that correlate strongly with the data receive large weights while the loosely related dimensions are assigned small values. These characteristic of

LAC algorithm makes it more efficient in some areas, since considering all the dimensions, makes the algorithm less prone to loss of information.

Some characteristic of LAC, such as local weight estimating and independency of clusters, make it attractive for parallel implementation. and that is why we have focused on this algorithm. Our main contribution in this paper is to propose a parallel version of LAC, or *PLAC*, on *Parallel Random Access Machine* (PRAM) [5]. As we know, LAC suffers from the long running time in some applications. Our parallel approach, has achieved a significant speed up in these cases. In applications such as Gene clustering for which the similarity are measured by computationally intensive methods such as *Transformation Distance* [6], we achieve a consequential efficiency of $O(1)$.

The reminder of the paper is organized as follows. We first describe the formal definition of clustering problem and the LAC algorithm. The structure of our solution is discussed in section 3 followed by the parallel version of LAC. In section 5, we describe our experimental results.

2 Locally Adaptive Clustering Algorithm

In this section, we give a formal definition of the stated clustering problem followed by the description of LAC. First, we define some significant terms: weighted cluster and centroid. Consider a set of points in some n-dimensional space. A *weighted cluster* C is a subset of data points including a point called *centroid* such that the points in C are clustered around its centroid according to the L_2 norm distance which is weighted using a vector of weights. In other words, the concept of cluster is not based only on points, but also involves a weighted distance factor along each dimension.

Formally, given a set S of points x in the n-dimensional space, a set of k centroids $\{c_1, \cdots, c_k\}, c_j \in S, j = 1, \cdots, k$ coupled with a set of corresponding weight vectors $\{w_1, \cdots, w_k\}$, $w_j \in \mathcal{R}^n$, $j = 1, \cdots, k$, partition S into k sets $\{S_1, \cdots, S_k\}$:

$$S_j = \{x|(\sum_{i=1}^{n} w_{ji}(x_i - c_{ji})^2)^{1/2} \leq (\sum_{i=1}^{n} w_{li}(x_i - c_{li})^2)^{1/2}, \forall l \neq j\}$$

where w_{ji} and c_{ji} represent the ith components of vectors w_j and c_j respectively and the operator $-$, minus operator, is defined as the distance of the operand points in S.

The LAC algorithm is presented in Table 1. In this algorithm, a distributed set of points in S are chosen first as the k centroids. In the next step, the weight vectors are initialized by $1/\sqrt{n}$. Steps 3 to 7 presents the main iterating phase of the algorithm. For each centroid, in each iteration, the corresponding sets S_j is computed according to the definition above. Then, X_{ji}, the average distance from c_j to the points in S_j due to feature i, is calculated. The smaller X_{ji} the larger will be the correlation of points along dimension i. The values of weight vector are updated according to the formula $w_{ji} = \exp(-h \times X_{ji})/(\sum_{l=1}^{n}(\exp(2 \times -h \times X_{jl})))^{1/2}$, where h is a experimental parameter which determines the effect of X_{ji} on the new value w_{ji}. Bottou *et all* [8], shows that the exponential weighting is more sensitive to changes in local feature correlation [8]. This iteration continues until the S_j, for all values of j converge. It is proved in [9] that LAC will in fact converge. Also, the result is local minimum of the error function:

$$Err(\mathcal{C}, \mathcal{W}) = \sum_{j=1}^{k} \sum_{i=1}^{n} w_{ji} e^{-X_{ji}}$$

where $\sum_{i=1}^{n} w_{ji}^2 = 1 \forall j$

```
INPUT: $\mathcal{S}$: set of points $x$, $k$ and $h$
OUTPUT: *weighted clusters*, centroids $c_1, \cdots, c_k$ and weight vectors $w_1, \cdots, w_k$.
    1. Start with $k$ initial centroids $c_1, \cdots, c_k$;
    2. Set $w_{ji} = 1/\sqrt{n}$. for each centroid $c_j$, $j = 1, \cdots, k$ and each feature $i = 1, \cdots, n$,
    3. For each centroid $c_j$, and for each point $x$
        • Set $S_j = \{x | j = min_l L_w(c_l, x)\}$, $L_w(c_l, x) = (\sum_{i=1}^{n} w_{ji}(x_i - c_{ji})^2)^{1/2}$;
    4. COMPUTE NEW WEIGHTS. For each centroid $c_j$, and for each feature $i$:
        • Set $X_{ji} = \sum_{xinS_j} (c_{ji} - x_i)^2 / |S_j|$
        • Set $w_{ji} = \frac{\exp(-h \times X_{ji})}{\sum_{l=1}^{n} \exp(2 \times -h \times X_{jl}))^{1/2}}$;
    5. For each centroid $c_j$, and for each point $x$:
        • Recompute $S_j = \{x | j = min_l L_w(c_l, x)\}$;
    6. COMPUTE NEW CENTROID. Set $c_j = \frac{1}{|S_j|} \sum_{x \in S_j} x$
    7. Iterate 3.4.5,6 until convergence.
```

Table 1: The Locally Adaptive Clustering Algorithm

3 Design & Architecture of PLAC

In this section we describe the design principles of *PLAC: The Parallel Locally Adaptive Clustering* method. Plac structure is based on *Parallel Random Access Machine*(PRAM) architecture and its basic routines.

Suppose we are to find a set of k centroids for clustering objects in a n-dimensional space. PLAC architecture, for solving such instance of the clustering problem, consists of $n \times k$ processors which share a global memory (see Figure 1). We group the processors into k bunches of size n and denote them *C-Columns*. Each processor in a group corresponds to one of the attributes of the space, consequently. A C-Column represents a *centroid*. *The Leader* is a process of an C-Column which maintains the communicating operations among the C-Columns. The leaders produce a set which we name it *L-Row*.

In the following subsections, we will go thorough the design of an interface for implementing PLAC over PRAM. This interface is based on the basic PRAM's operations such as broadcasting and semi-grouping.

3.1 C-Column

Cooperating with each other, the processors of a *C-Column* perform the operations invoking the attributes of the corresponding centroid in the clustering algorithm. In other words, the processors act as one *weighted cluster*. The term C_i is used to refer to the C-Column corresponding to the ith centroid.

Every C-Column has its own memory in the globally-shared memory. In fact a C-Column can be viewed as a PRAM. To implement PLAC, some simple operations are defined on C-Columns. In the following, these operations are described based on the simple routines of PRAM and are corresponding

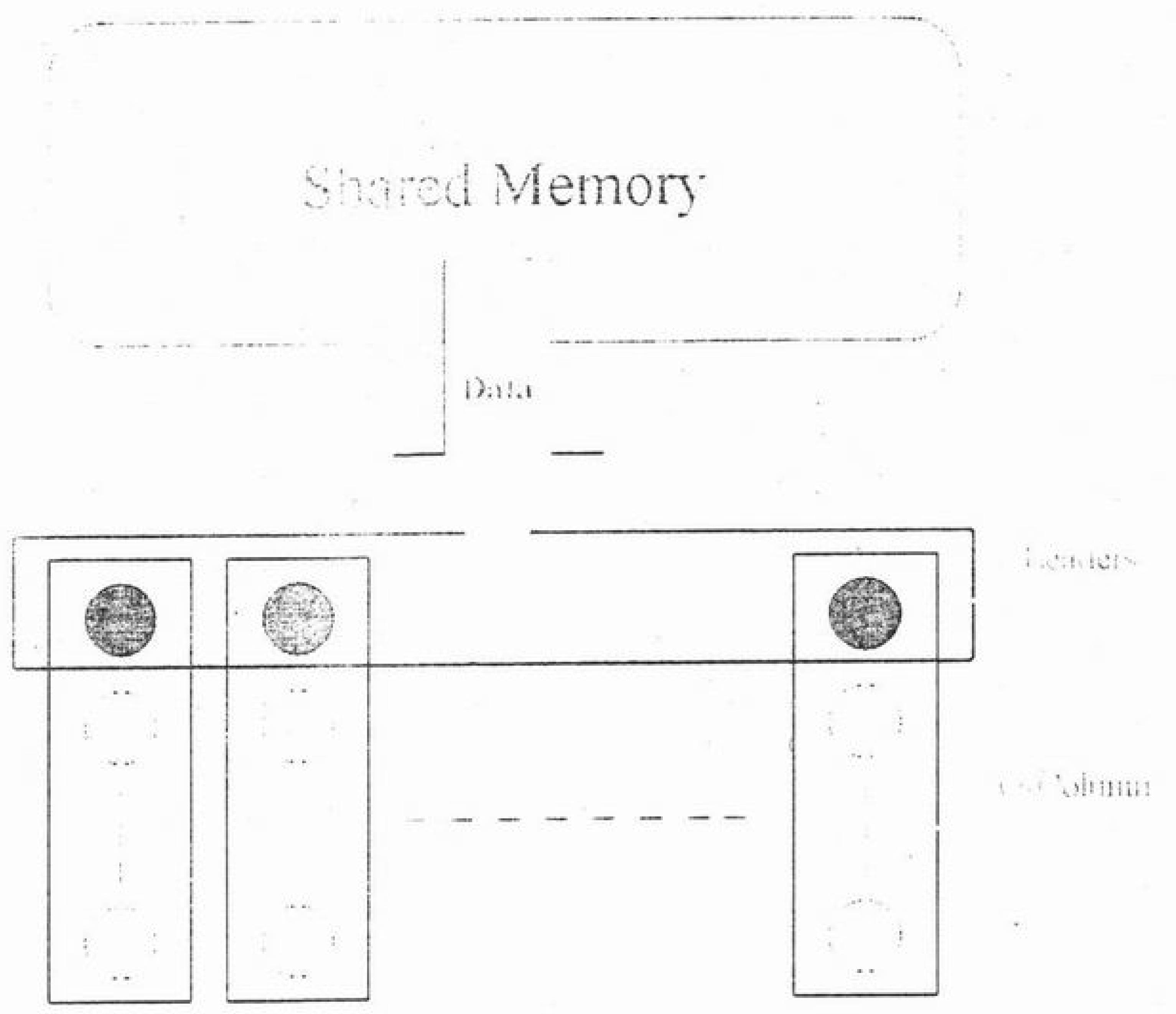


Figure 1: The structure of PRAM processors

to $C_i (1 \leq i \leq k)$. We denote the representing processor of dimension j (the jth attribute) of C_i by $P_{ji} (1 \leq j \leq n)$. The c_{ji}, w_{ji}, $|S_j|$ and x_{ji} values are maintained by P_{ji}.

bcast(msg): Broadcasting the value of msg to all of the other processors.

load(x): P_{ji}s read x_i from the memory.

dist(x): *load*(x) is performed then P_{ji}s compute the distance between x_i and c_{ji} in parallel and store the result on the memory.

sum: Compute the sum of the values written on the shared memory.

cum_c: P_{ji} maintains the new value of c_{ji} by cumulating the value of $x_i \in S_j$.

cum_x: For maintaining the value of X_{ji}, P_{ij} performs the following operation: cumulating the value of $(c_{ji} - x_i)^2$ and increasing $|S_j|$ by.

$update_c$: Updating the c_{ji} by dividing the corresponding cumulated value by $|S_j|$.

$update_x$: Updating the x_{ji} by dividing the representing cumulated value by $|S_j|$.

$update_w$: Using basic operations *bcast* and *sum*, the new value of w_{ji} is computed in parallel according to the following formula:

$$\exp(-h \times X_{ji}) / \sum_{l=1}^{n} \exp(2 \times -h \times X_{jl}))^{1/2}$$

In Table 2. the complexities of these operation on the different PRAM architecture are listed.

ARCHITECTURE	C-COLUMN							L-ROW	
	bcast	*load*	*dist*	*sum*	*cum*	*update*		*min*	*converge*
					$\{x, c\}$	$\{x, c\}$	w		
EREW	$\log n$	1	d	$\log n$	1	1	$\log n$	$\log k$	$\log k$
CRCW-SUM	1	1	d	1	1	1	1	$\log k$	$\log k$
CRCW-MIN	1	1	d	$\log n$	1	1	$\log n$	1	1
SEQUENTIAL	n	n	$n \times d$	n	n	n	n	k	k

Table 2: The time complexity of the operations over different architecture. d is the cost of computing the similarity of to features.

3.2 L-Row

L-Row is the set of the leaders of the C-Columns. These processors collaborate together for communicating the data and perform aggregation operation among the C-Columns. Similar to C-Columns, L-Row can be viewed as a PRAM which has its own part in the shared memory. Two main operations are defined on L-Row: *min* and *converge*.

In some parts of the algorithm we need to find the minimum value among the leader processors. Finding the closest centroid to an object or *converge* operation are instances of such operations. *min* is the operation defined to compute such minimum value.

Clusters are converging when the $S_j (1 \leq j \leq k)$s do not change. We estimate the convergence by considering the size of S_js. We assume that the clusters are converging when the size of S_js remains constant after v iterations where v is a constant which will be evaluated through the experiments; we find 7 a reliable value for v in our evaluation. *converge* operation inspects whether the sets are converging or not. This operation can be easily performed by observing the maximum difference of old and new value of $|S_j|$. Note that the maximum of some positive numbers can be found by searching for the minimum in negatives of these numbers.We list the complexities of these operations in Table 2.

4 Implementation of PLAC

In this section, using the interface defined in the previous section, we describe the parallel implementation PLAC. The computationally complex part of the algorithm is the iteration part while the other parts executed only once. In the following, we present the implementation of iteration phase of LAC. The ▷ mark, indicates that the operation is done inside one processor.

The analysis of PLAC follows. Our main contribution is reducing the time complexity of iteration of parts 3 to 7 where the number of iterations depends on the characteristics of input data. It is clear that most of the time of the program spend on iteration 3 and 5 and we can neglect the other part to compute the time complexity.

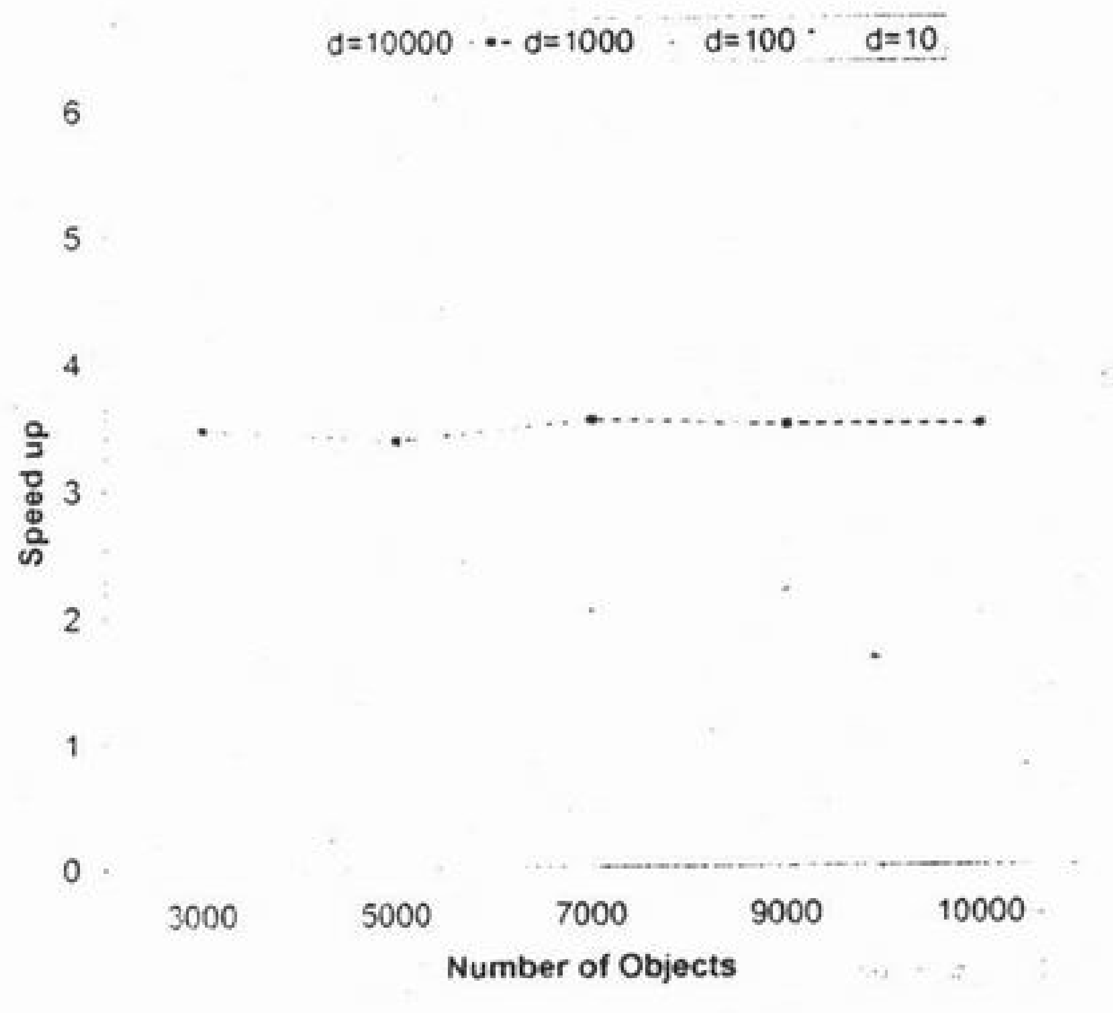
Parts 3 and 5 consist of *dist*, *sum*, *min* and $cum_{\{x,c\}}$ operations. In Table 4, we list the time complexity of these operations on different PRAM's architectures. The term d is the cost of measuring the similarity of to features. The complexity of d varies gravely due to the different applications. We have considered two cases:, $d \in O(1)$ and $d \in \Omega(\log n + \log k)$. Former for the data-sets are simple vector of real numbers and the later, according to applications such as Gene clustering when the similarity are measured by, are computationally intensive methods. As it become clear from the Table 3. for large value of d we will have consequential speed-up and efficiency of $O(1)$.

```
3. For each point $x$ do
      For each C-Column $C_{i_{(1\leq i\leq k)}}$ do:
            $C_i.dist(x)$
            $C_i.sum$
      L-Row.$min$
      ▷ If $min$ is equal to value computed by $C_i$ then:
            $C_i.cum_x$
4. For each C-Column $C_{i_{(1\leq i\leq k)}}$ do:
      $C_i.update_x$,
      $C_i.update_w$,
5. For each point $x$
      For each C-Column $C_{i_{(1\leq i\leq k)}}$ do:
            $C_i.dist(x)$
            $C_i.sum$
      L-Row.$min$
      ▷ If $min$ is equal to value computed by $C_i$ then:
            $C_i.cum_c$
6. For each C-Column $C_{i_{(1\leq i\leq k)}}$ do:
      $C_i.update_c$,
7. If L-Row.$converge$
      Terminate.
```

Table 3: Parallel LAC

architecture	complexity	$d = O(1)$		$d = \Omega(\log n + \log k)$	
		speed-up	efficiency	speed-up	efficiency
SEQUENTIAL	$n \times k \times d + n + k$	-	-	-	-
EREW	$d + \log n + logk + O(1)$	$(kn)/\log kn$	$1/\log kn$	$O(kn)$	O(1)
CRCW-SUM	$d + logk + O(1)$	$(kn)/\log k$	$1/\log k$	$O(kn)$	O(1)
CRCW-MIN	$d + \log n + O(1)$	$(kn)/\log n$	$1/\log n$	$O(kn)$	O(1)

Table 4: The time complexity of the part 3 of the LAC algorithm over different architecture. d is the cost of computing the similarity of to features.



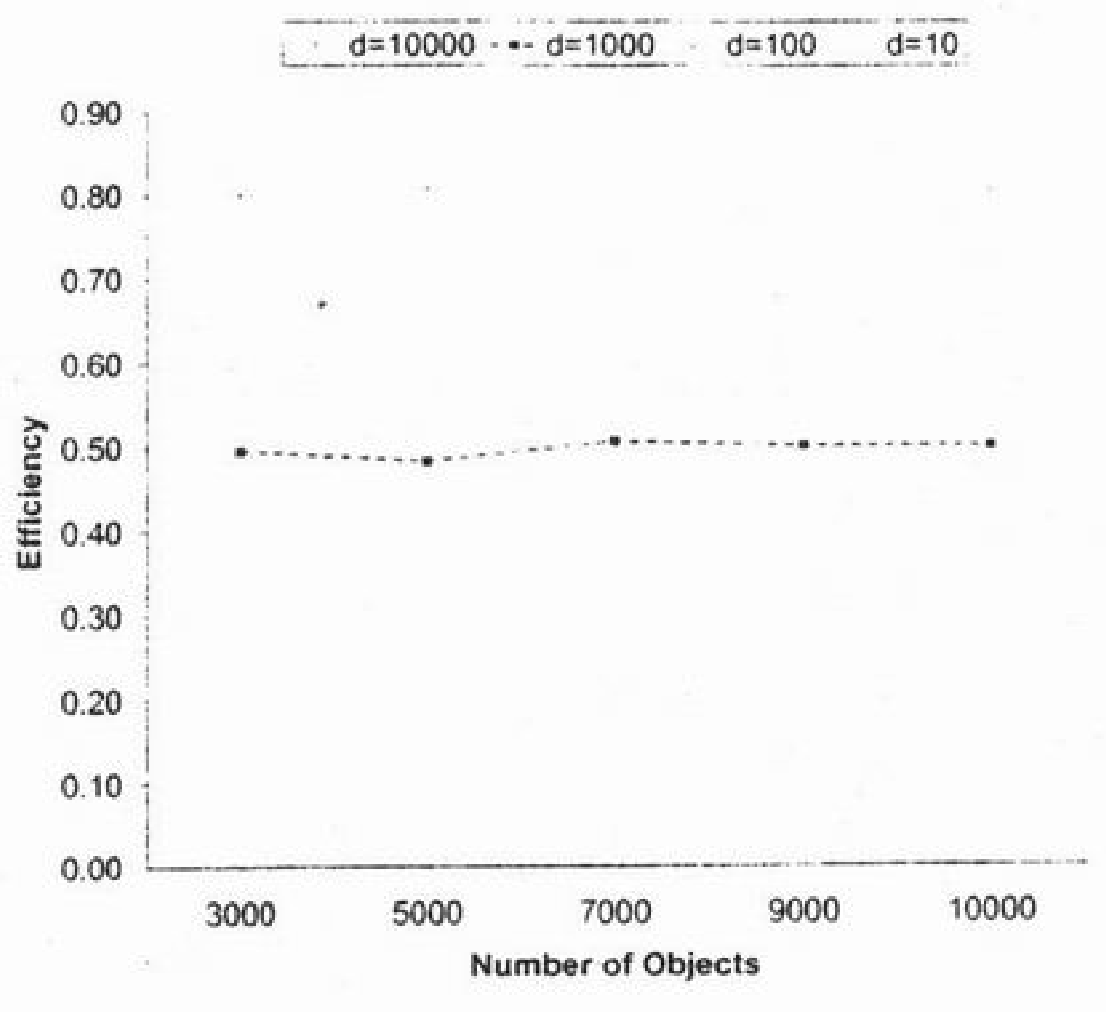


Figure 2: The graph of speed up and efficiency for different set of objects.

5 Experimental Evaluation

We have performed simulation to evaluate our design. The algorithm is implemented using LAM/MPI [7] technology. There is some limitation to propose a general method for simulating PRAM by MPI (Message Passing Interface) but the nature of our problem and algorithms permit us to emulate PRAM with intangible overhead. In our setup one of the computer of the cluster act as the share memory. Reading and writing is simulated by receiving and sending the message from(to) this computer.

In our experiment we have tested the algorithm with seven centroids each of which is assigned to one of the computers of the system emulating the n corresponding processor in PRAM model. Test-cases consist of 3000 up to 10000 objects in 30-dimensional space. The attributes of the space are synthesized by the Gaussian random variable with various parameter.

To evaluate the speed up and the efficiency of our approach, we execute the parallel clustering algorithm for different values of d which specifies the average number of float operation needed to estimate the similarity of two features. The results are presented in Figure 2. In the former plots, the achieved speed up for d equals to 10^1 up to 10^4. The later, exhibits the corresponding values of efficiency. The results validate our claim in Table 4. For large values of d, the speed up is about 5.6 and the efficiency is 80% which is considered as salient achievements.

6 Conclusion

In this paper, we proposed a parallel approach to overwhelm the clustering problem in high dimensional spaces. We designed an environment on Parallel Random Access Machine. Also, we modified the algorithm to make it parallel. We analyzed the algorithm and showed that for computationally massive problems, we can achieve an $O(1)$ efficiency. We have demonstrated our claims through a series of experiments.

We plan to work on implementing parallel version of the others well-known clustering algorithms, such as CLIQUE, to evaluate and compare their performance. Also, we intend to experiment real data-sets on larger cluster systems.

References

[1] Jiang. D. Pei. J. and Zhang, A. DHC: A Density-Based Hierarchical Clustering Method for Time Series Gene Expression Data. *BIBE* 2003: 393-400

[2] Agrawal. R. Gehrke, J. Gunopulos, D. Raghavan, P. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, Proc. *ACM SIGMOD*, 1999

[3] Aggarwal. C., Procopiuc, C., Wolf, J. L., Yu, P. S., and Park, J. S. Fast Algorithms for Projected Clustering. *SIGMOD*, 1999

[4] Domeniconi. C. Papadopoulos, D. Gunopulos, D. Ma, S. Subspace Clustering of High Dimensonal Data. Proc. *SIAM International Conference on Data Mining*, April, 2004

[5] Leighton. F. *Introduction to Parallel Algorithms and Architectures : Arrays, Trees. Hypercubes* . 1991

[6] Behzadi. B and Steyaert, J.-M. On Transformation Distance Problem, proc, *SPIRE*, 2004

[7] http://www.lam-mpi.org

[8] Bottou. L.. and Vapnik, V. Local learning algorithms. *Neural computation*, 4(6):888-900, 1992.

[9] Domeniconi. C. Locally Adaptive Techniques for Pat- tern Classification, PhD dissertation, UC Riverside, Computer Science Dept., August 2002.