PPreDeConStream: A Parallel Version of PreDeConStream Algorithm

Reza Tashvighi
Department of Computer Engineering,
Tehran North Branch,
Islamic Azad University,
Tehran, Iran

Alireza Bagheri*
Department of Computer Engineering, Tehran North Branch, Islamic Azad University,
Tehran, Iran
Department of Computer Engineering and Information Technology, Amirkabir University of Technology,
Tehran, Iran

ABSTRACT
Clustering is one of the major techniques in data mining. Clustering data streams have drawn attentions in the past few years because of their ever-growing presence. Data streams add more challenges to clustering such as limited time, limited memory and one pass clustering. Further, discovering clusters with arbitrary shapes is important in data stream applications.

Now a few clustering techniques for data streams exist in multidimensional spaces and the technique of "clustering projected or subspace" is used. Therefore, the task of projected clustering (or subspace clustering) has to be defined.

PreDeConStream is a density-based data stream clustering algorithm for clustering high-dimensional data streams. In this paper, PPreDeConStream is present as a parallel version of PreDeConStream algorithm in the shared memory model. The theoretical and experimental results show that PPreDeConStream offers nearly linear speedup while keeps other advantages of PreDeConStream.

General Terms
Data mining, data stream, parallel algorithms, clustering, micro cluster

Keywords
Clustering data stream algorithms, parallel algorithms, microcluster, density-based clustering, shared memory model.

1. INTRODUCTION
Every day, a huge amount of data will be made and the high percentage of them has been created in the recent years making us not able to store such massive amounts of data. Therefore, much attention has been made into mining data streams, but a few algorithms introduced into this area. Clustering is an important way to extract data streams. In clustering, data is classified into some groups.

Data streaming applications such as [1] are as follows: financial, network monitoring, security, telecommunication data management, web applications, manufacturing, sensor networks, and others. Algorithms clustering data streams are categorized into these main types: Partitioning, Hierarchical, Density-based, Grid-based and Model-based algorithms.

Comprehensive methods of the data stream algorithms are based on data density. In a clustering survey [2], algorithms based on density in the data stream are classified into two general groups of microclustering, and grid-based algorithms. With this approach, the clusters of different shapes can be identified.

Clustering data streams must handle the following challenges:

- Handling noisy data
- Handling evolving data
- Limited time
- Limited memory
- Handling high-dimensional data

There are different algorithms for Density-based clustering data stream approach. DenStream [3] is a well-known Density-based microcluster of data stream's algorithms.

High dimensional data has challenged, known as the curse of dimensionality [4]. HDDStream [5] algorithm is intended to handle the high dimensions by extending DenStream. This algorithm keeps the online phase summary from both points and dimensions of clusters' final offline phase based on a PreDeCon [6] algorithm.

HDDStream algorithm has introduced to prefer vector for each microcluster, which is related to preferring dimension in high-dimensional data. At the time of pruning such as Denstream, microcluster weight periodically is controlled.

Nowadays, since a large amount of data must be analyzed, the algorithms should be performed effectively in terms of time complexity. To improve time complexity of existing algorithms, parallelization is considered. Parallel and distributed computing have important roles in decreasing the response time. With parallel computing, the performance of clustering can be improved. For example, the G-DenStream [7] used in heterogeneous environments for diverse computing with OpenCL [8] implementation has been proposed.

In phase offline [7], an effective strategy is provided on the GPU. Calculating the distances between the data points is executed in parallel on GPU. Algorithm ensures a very good work and parallel GPU with low divergence is achieved. Similar calculations in the G-DenStream from any point with the center microcluster are performed effectively in parallel.

PreDeConStream [9] is similar to HDDStream; however, the algorithm improves the efficiency of HDDStream on the offline phase. Nevertheless, searching the impact neighboring clusters and upgraded cluster are a time-consuming process.

In this paper, PPreDeConStream algorithm is presented, a parallel version of PreDeConStream in the shared memory model. The complexity of parallel version PPreDeConStream algorithm is 1/P sequential PreDeConStream algorithm, where the number of processors is denoted by P.

The remaining of the paper is organized as follows. In section 2, the preliminaries is given. In section 3, PreDeConStream algorithm is described and section 4 presents the parallel algorithm, PPreDeConStream. Section 5 shows the experimental results. Section 6 lists the conclusions and highlights the future works.

2. PRELIMINARIES

PreDeConStream [9] uses the notions of density-based clusters, mentioned in PreDeCon [6] and DenStream [3]. The algorithm applies the ideas of the subspace and micro cluster. To control dimensions, which is relevant to a cluster, the notion of subspace preference for each point is used.

This algorithm, like other data stream clustering algorithms, uses online and offline famous models. In offline phase, the effective localization of the resulting cluster by changing the stream was affected at a particular time and then keeps only the final cluster. The algorithm defines the interval time that ensures no changes will be done as a result of clustering. Online phase also provides several lists for organizing the speed up to update.

Ideas and notions of this section come from [9]. The following definitions are used to clarify PreDeConStream.

PreDeConStream has seven input parameters; four density parameters $\varepsilon_N, \varepsilon_F$,μ_N, μ_F and fading parameter λ and preference parameter τ and sensitivity to outliers parameter β. Since the algorithm uses a density-based clustering over its online and offline phases, similar symbols that appear in both phases are differentiated with a F subscript for the offline phase and N for the online phase [9]. They should be chosen as suggested in [3].

Let DS be a database of d-dimensional points ($DS \subset R^d$), where the set of attributes is denoted by $A_i = \{A_1, A_2, \cdots, A_d\}$ and Dist: $R_d \times R_d \rightarrow R_d$ is a metric distance function between points in DS.

According to [10] Note that weight of a given point of time arrival is calculated based on fade function weight. The adopted decay role follows the exponential function given by $f(t) = 2^{-\lambda t}$, where the $\lambda > 0$ limit decides the decay rate, and t is the current time. The higher the value of λ, the lower the importance of the past data regarding the most recent data [10].

Based on these ideas [9], the classic definitions of density-based data stream clustering is derived:

Definition 1 [9] **Core Microcluster**. At time t, the core microcluster is defined as close points $p_1, \cdots, p_n$ with time stamps $t_1, \cdots, t_n$. It is depicted by a tuple $CMC(w, c, r)$ with:

1. Weight,$w = \sum_{j=1}^{n} f(t - t_j), w \geq \mu_N$
2. Center, $C = \frac{\sum_{j=0}^{n} f(t-t_j)p_j}{w}$
3. Radius, $r = \frac{\sum_{j=0}^{n} f(t-t_j)dist(p_j,c)}{w}, r \leq \varepsilon_N$

The types of microclusters are also given, the potential micro cluster and the outlier microcluster, to allow quickly recognize changes in the data stream.

Definition 2 [9] **Potential and Outlier microcluster**. A potential microcluster $PMC = (\overline{CF^1}, \overline{CF^2}, w, c, r)$ is denoted as follows:

1. Weight, $w = \sum_{j=1}^{n} f(t - T_j), w \geq \beta\mu_N$
2. Linear weighted sum of the points, $\overline{CF^1} = \sum_{j=0}^{n} f(t - Tjpj$
3. linear weighted squared sum of the points, $\overline{CF^2} = \sum_{j=0}^{n} f(t - T_j)p_j^2$
4. Center, $C = \frac{\overline{CF^1}}{w}$
5. Radius, $r = \sqrt{\frac{|\overline{CF^2}|}{w} - \left(\frac{\overline{CF^1}}{w}\right)^2}$

An outlier microcluster $OMC = (\overline{CF^1}, \overline{CF^2}, w, c, r, t_0)$ indicates similar PMC with the following modifications:

1. Weight, $w = \sum_{j=1}^{n} f(t - T_j), w < \beta\mu_N$
2. In addition, the entry point is decided to going to outlier microcluster, either it evolves or fades.

Definition 3 [9] **Microclusters Maintenance.** Any core, potential, or outlier microclusters at time t $MC_t = (\overline{CF^1}, \overline{CF^2}, w)$ is maintained as follows:

If a point impacts MC at time $t + 1$ then its statistics become: $MC_{t+1} = (2^{-\lambda}.\overline{CF^1} + p, 2^{-\lambda}.\overline{CF^2} + p^2, 2^{-\lambda}.w + 1)$. In a different state, if no point was appended to MC for any time interval δt , the microcluster can be made after any time interval δt as follows:

$MC_{t+\delta t} = (2^{-\lambda\delta t}.\overline{CF^1} + p,\ 2^{-\lambda\delta t}.\overline{CF^2} + p^2, 2^{-\lambda\delta t}.w)$.

The maximum weight w_{max} of any core, potential or outlier microcluster MC is $\frac{1}{1-2^{-\lambda}}$.The minimum duration time for a newly produced micro cluster to get bigger into a potential microcluster is$T_p = \left\lceil \frac{1}{\lambda} log_2 \left(\frac{1}{1-\beta\mu_N(1-2^{-\lambda})} \right) \right\rceil$. The minimum duration time demanded a potential microcluster to decay into an outlier microcluster is $T_d = \left\lceil \frac{1}{\lambda} log_2(\beta\mu_N) \right\rceil$.

Definition 4 [9] **Minimum Offline Clustering Legality Interval**. The minimum legality interval of an offline clustering T_v defines the time within which PreDeConStream does not need to update the offline clustering since it is still valid because no change of the status of any microcluster status happened. It is defined as $T_v = min\{T_p, T_d\}$.

Definition 5 [9] **Subspace Preference Vector W_c** . For each dimension ith, if the variance of the microclusters c of the Euclidean $\varepsilon - neighborhood\ N_{EF}(c)$ is below a user-defined threshold, then the ith entry of the preference subspace vector w_c is set to a constant ≥ 1 , otherwise, the entry is set to 1.

A data structure manages the updated and non-updated microclusters at each timestamp in an efficient and effective way (see

Figure **1**). The algorithm grouped the microclusters into multiple lists according to their weight.

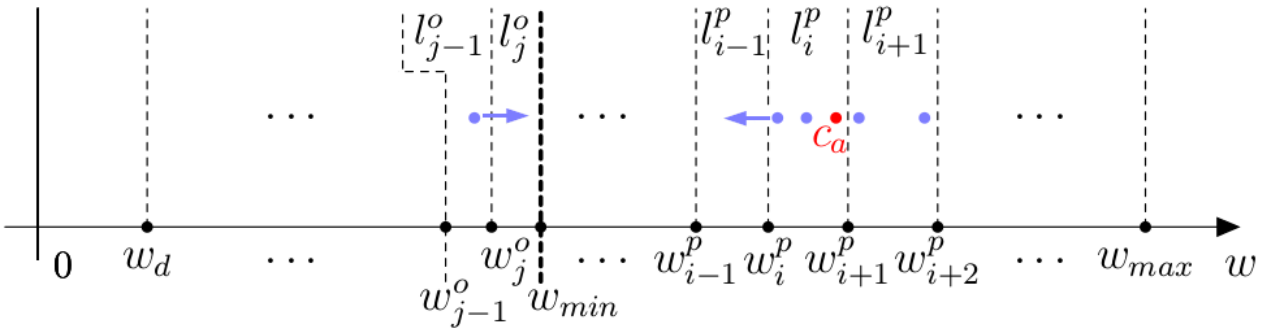

Figure 1: A sample of outlier and potential lists [9]

There are two types of lists: outlier lists l_j^o, and potential lists l_i^p. The borders of the lists are $W_d = 1$, $W_{min} = \beta\mu_N$, $W_{max} = \frac{1}{1-2^{-\lambda}}$. The internal borders are selected as $w_i^p = \frac{w_{i-1}^p}{2^{-\lambda}}$ for the potential lists, and $w_i^o = 2^{-\lambda} w_{i-1}^o + 1$ for the outlier lists [9]. All microclusters in these two types of lists that are not hit in the previous timestamp disappear to a lower-weighted list.

As DenStream, PreDeConStream discovers a cluster uniquely by any of its preference-weighted core microclusters.

3. THE PreDeConStream ALGORITHM

To find data stream clusters, the PreDeConStream [9] algorithm merely runs one pass over the database according to limit setting.

In the initialization phase of PreDeConStream algorithm, the minimum time span Tv is calculated and uses the PreDeCon [6] algorithm on the first early points, and forms the preliminary lists potential microclusters.

The algorithm is needed to an initial potential of microclustering to get started. During the initial setup algorithm, the points of a data stream are buffered and call PreDeCon algorithm with least $\beta\mu N$ points in its εN −neighborhood.

In online process, for each data point, if the aggregate of the weights of the data points in the neighborhood $\varepsilon(N)$ radius is above the weight $\beta\mu N$ threshold, then a potential microcluster is generated. When a new data point arrives, it is added to either the nearest existing potential microcluster or outlier microcluster. A microcluster is chosen with the distance less than or equal to the radius threshold [2]. If it does not belong to any of lists then, a new outlier microcluster is created, and it is placed in the outlier lists.

In PreDeConStream, offline processing of the data stream, the neighbors of newly inserted potential microclusters as well as deleted p-microclusters is checked. The subspace chose vectors of these neighboring microclusters are updated and put in a list as updated microclusters. The affected microcluster list is used in the offline phase as expanding clusters to improve the efficiency of the offline phase [2]. Finally, a list of potential microclusters is needed to be reinserted into the clustering. Starting from a microclustered in the list UPDSEED, the algorithm PreDeCon [6] is called with considering the old existing clustering.

4. THE PPreDeConStream ALGORITHM

In this proposed parallel algorithm, all the points and microclusters are available to each of P processors by the shared memory. Using parallel techniques can enhance processing speed. The both techniques data and task division are applied.

In initialization phase of the proposed parallel algorithm, the initial clustering is computed with an adapted version of PPreDeCon [11](see Figure 2).

```
Algorithm initPPreDeCon(D_buf, d, ε, μ, λ, δ)
/* assumption: each point in D in marked as unclassified
and D_j is set of point of each Processor */
Processor j, 0 ≤ j < P        do
  for each unclassified   o ∈ D_buf    do
  if  C_ORE^pref_den(o) then  /*expand a new cluster*/
    create new microCluster  p-micro
    generate new clusterID
        insert all x ∈ N_ε^w̄o(o) into queue Q;
        while Q ≠ ∅ do
        q = first point in Q;
        compute  R = {x ∈ D_IR R_EACH^pref_den(q, x)};
        for each x ∈ R do
          if  x is unclassified  then
              insert x in Q;
              if   x is unclassified or noise  then
                 assign current clusterID to x
                 insert x into microCluster
                 remove   q   from Q;
              else
                   mark o as noise;
          end.
       End.
          Add p-micro into p-microCluster lists
   end.
// Merge step
Processor  j, 0 ≤ j < P        do
  for each classified  m ∈ D_j  do
     if   |clusterID| > 1   then
     // number of clusterID that point gets
         set Min of them as clusterID;
      remove  m point from microcluster in the list l_j^p
   end.
```

Figure 2: The pseudo code of the initPPreDeCon

In PPreDeCon, data is randomly divided among P processors. Each processor checkpoint that is in the neighborhood creates

clusters. A new cluster identification (CID) is created, and a fresh cluster is expanded.

After all the processors finish their work, some points may get more than one CID [11]. This problem should be resolved. To resolving, in merge step, the algorithm sets the minimum CID for each point with more than one CID and removes from microcluster list l_i^p.

The sequential algorithm finds the nearest point on the list of potential clusters and outlier micro spends a lot of time. Also, in procedure fade algorithm spends a lot of time. In update clusters' procedure, the algorithm spends a lot of time. Pseudo-code in Figure 3 is provided to search for the nearest microclusters of a point. The structure maintenance of microclusters contains several lists. In parallel, a list of the nearest microclusters is found and the closest microcluster in the list is selected and also the fade task can be executed for all microclusters by means of a single one-dimensional list (see Figure 4).

```
Algorithm search Nearest MicroClusterPoint (list Lm, data point p)

Processor  j, 0 ≤ j < P      do
  for each  m ∈ Lm     do
      compute distance p from m
       if distance ≤ list_min_distance[j] then
       z     list min distance [j] = distance
             list min micro [j] = m
        end-if.
    End for.
//merge for find minimum of minimum list
  for each 0 ≤ i < P do
      if list_min_distance[i] ≤ min_distance then
            min distance = list_min_distance[i]
           min micro = list_min_micro[i]
      end-if.
   End for.

return min_micro;
```

Figure 3: Search the nearest point from a list of microclusters

In the update clustering procedure, the neighbors of newly inserted potential microclusters, as well as deleted possible microclusters, are checked and put in a list as affected microclusters. Finally, the actual microclusters of UPDSEED need to be reinserted into the clustering and call PPreDeCon algorithm.

A task for all elements is affected by repeating the list of microclusters. This is the task of the proposed algorithm is done in parallel. The influence microcluster list is used for PPreDeCon [11] algorithm. The pseudo-code parallel updated clustering in the proposed algorithm is shown in Figure 5.

```
//The weight w is βμN for potential and wd for outlier micro-cluster .
Algorithm updateFadeMicroClusters
      (list Lm, weight w,time δt)

Processor        j, 0 ≤ j < P        do
  for each Microcluster mc ∈ Lm do
   if  the weight of mc < w then
     delete mc;
    if (m-1) not zero then
       add mc into Lm−1;
    end if;
   end if;
 end for;
end.
```

Figure 4: Update fade microcluster

5. EXPERIMENTAL EVALUATION

In this section, accuracy and time complexity of PPreDeConStream are evaluated. PPreDeConStream, as well as comparative PreDeConStream algorithm, were implemented in C and OpenMP [12] library. All the experiments were done on a CentOS Linux operating system with two and four processors.

Datasets: For the evaluation of PPreDeConStream, two data sets were used:

1. Network Intrusion Detection data set KDD CUP'99 (KDD-cup) [13] used to evaluate several stream clustering algorithms [3] with 494021 TCP connections; each represents either a normal connection or any of 22 different types of attacks. Each connection consists of 42 dimensions.
2. Physiological data set [14] is a multivariate data set recorded from a patient in the sleep with 3400 objects. Each connection consists of three dimensions, heart rate, the respiration rate, and the blood oxygen.

Evaluation measure and limit settings to evaluate the quality of the clustering results, the cluster purity [3] measure is used. Purity of clusters is defined as follows:

$$\text{purity} = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K} \times 100\% \qquad (1)$$

Where K denotes the number of clusters. $|C_i^d|$ denotes the number of points with the dominant class label in cluster i th. $|C_i|$ denotes the number of points in cluster i th [3].

For efficiency, runtime was tested in seconds. Experiments Purity and runtime were tested for both PreDeConStream and PPreDeConStream algorithms.

Evaluation of Clustering Quality: Unless mentioned, the limits were set similar to [9] as follows: decay factor $\lambda = 0.25$, initial data object $Init = 2000$, and horizon $H = 5$, and $\varepsilon_F = 2 \times \varepsilon_N$ and speed $v = 1000$.

Using the Network Intrusion Detection data set for both algorithms, the limits are set to $\mu_N = 10$, $\mu_F = 5$, $\beta = 0.23$, $\tau = 32$.

It can be seen from Figure 6 the cluster purity of PreDeConStream and PPreDeConStream for KDD Cup99 is equal.

Algorithm P_updateClustering (C)
Processor j, $0 \le j < P$ do
for all $\boldsymbol{c_p} \in \boldsymbol{Insert\_PMC}$ *do*
compute the subspace preference vector w_{c_p};
for all $\boldsymbol{c_q} \in \boldsymbol{N_{\varepsilon F}(c_p)}$ *do*
update the subspace preference vector of c_q;
if core member property of c_q has changed then
add c_q to AFFECTED_CORESi
end if;
end for;
compute UPDSEEDi based on AFFECTED_CORESi;
end for;
Processor j, $0 \le j < P$ do
for all $\boldsymbol{c_d} \in \boldsymbol{Delete\_PMC}$ *do*
for all $c_q \in N_{\varepsilon F}(c_d)$ do
pdate the subspace preference vector of c_q;
if core member property of c_q has changed then
add c_q to AFFECTED_CORESd;
end if;
end for;
compute UPDSEEDd based on AFFECTED_CORESd;
end for;
Processor j, $0 \le j < P$ do
$UPDSEED \leftarrow UPDSEED_i \cup UPDSEED_d$
Call PPreDeCon($UPDSEED$) with considering the old cluster structure.

Figure 5: Parallel update clustering procedure in the proposed algorithm

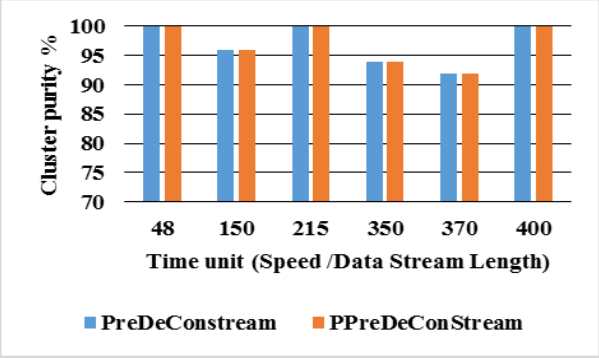

Figure 6: Clustering purity for KDDcup99 dataset

Using the patient in the sleep data set for both algorithms, the limits are set to $\mu_N = 5, \mu_F = 3, \varepsilon_N = 4, \beta = 0.2, \tau = 3$ and speed data stream $v = 100$. It can be seen from Figure 7 the cluster purity of PreDeConStream and PPreDeConStream is equal.

In all experiments, PPreDeConStream discovered all the clusters and detected noises as well as PreDeConStream.
The experimental results are summarized in Figure 8 and Figure 9. In these experiments, the number of processor P is 2 and 4.

The result shows the run time of PPreDeConStream is lower than PreDeConStream, which is compared in Figure 8 and Figure 9.

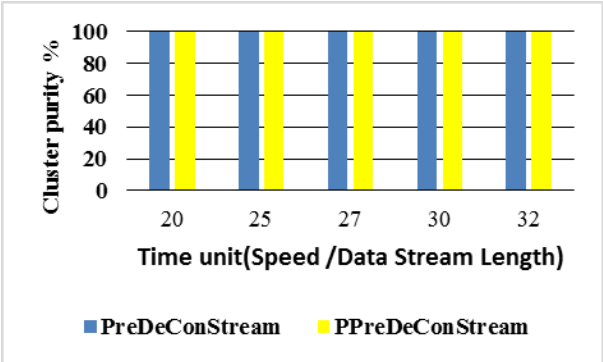

Figure 7: Clustering purity for a patient in the sleep dataset

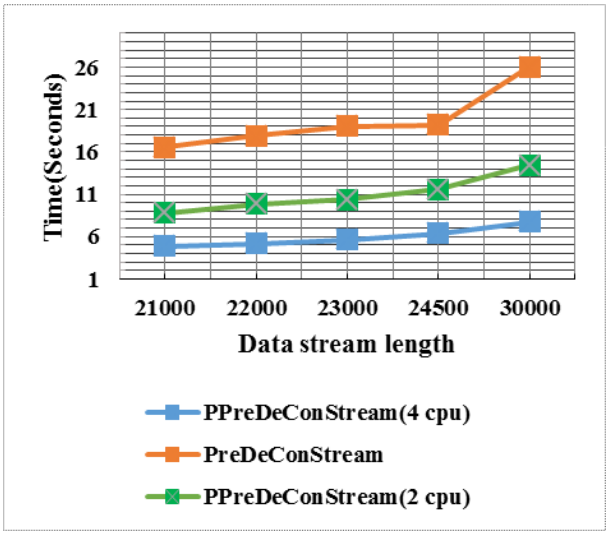

Figure 8 : Runtime comparison between PreDeConStream and PPreDeConStream for KKCup99

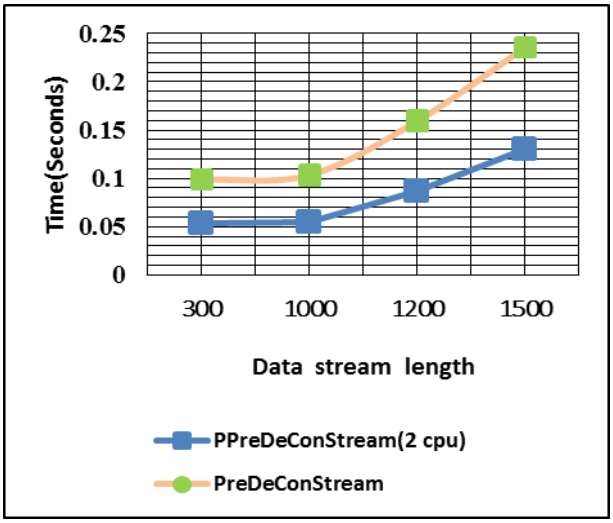

Figure 9: Runtime comparison between PreDeConStream and PPreDeConStream a patient in the sleep dataset

6. CONCLUSIONS

In this paper, the parallel clustering algorithm PPreDeConStream was presented for mining large and high dimensional data stream clustering. Lack of time can be solved by parallel computing. The parallel implementation uses the shared memory model. The experiments showed that the actual clustering could be performed with excellent speedup and good response time.

Speedup the algorithm has been nearly linear with the number of processors. The presented algorithm showed that the quality of the clusters also keeps in the parallel version.

Other advantages of this algorithm were accuracy and supporting high dimensional data sets comparing to other density-based clustering algorithms; it also discovers arbitrary shapes and is effective in noise detecting.

The future scope of this idea can be followed in below functions: financial, network monitoring, security, telecommunication data management, web applications, manufacturing, sensor networks, and others.

In the future, a parallel version of the message passing model of the algorithm is considered. Focusing on reducing time spent in all the step of the algorithm may be taken as future works. Focus on methods based on density. A parallel version of other data streams mining methods may be considered in the future. However, one may consider the message passing model, map-reduce model, distributed memory and heterogeneous platforms.

7. REFERENCES

[1] E. Ikonomovska, S. Loskovska and D. Gjorgjevik, "A Survey of Stream Data Mining," In Proc. the 8th National Conference, pp. 9-25, 2007.

[2] A.Amini, T. Y. Wah and H. Saboohi, "On Density-Based Data Streams Clustering Algorithms: A Survey," Journal of Computer Science And Technology, vol. 29, no. 1, pp. 116-141, 2014.

[3] F. Cao, M. Ester, W. Qian and A. Zhou, "Density-Based Clustering over an Evolving Data Stream with Noise," In Proc. the 2006 SIAM Conference on Data Mining, pp. 328-339, 2006.

[4] C. C. Aggarwal and C. K. Reddy, Data Clustering Algorithms and Applications, Chapman & Hall, 2014.

[5] A.Ntoutsi, A. Zimek, T. Palpanas, P. Kroger and H.-P. Kriegel, "Density-based Projected Clustering over High Dimensional Data Streams," Proceedings of the 2012 SIAM International Conference on Data Mining, pp. 987-998, 2012.

[6] C. Bohm, K. Kailing, H.-P. Kriegel and P. Kroger, "Density Connected Clustering with Local Subspace Preferences," Data Mining, 2004. ICDM '04. Fourth IEEE International Conference, pp. 27-34, 2004.

[7] M. Hassani, A. Tarakji, L. Georgiev and T. Seidl, "Parallel Implementation of a Density-Based Stream Clustering Algorithm Over a GPU Scheduling System," Trends and Applications in Knowledge Discovery and Data Mining, pp. 441-453, 2014.

[8] "The OpenCL Specification Version: 2.0 Document Revision: 26," Khronos OpenCL Working Group, 2014.

[9] M. Hassani, P. Spaus, M. M. Gaber and T. Seidl, "Density-Based Projected Clustering of Data Streams," Scalable Uncertainty Management, vol. 7520, pp. 311-324, 2012.

[10] J. A. Silva, E. R. Faria, R. C. Barros and J. P. Gama, "Data Stream Clustering: A Survey," ACM Computing Surveys (CSUR), vol. 46, no. 1, pp. 1-37, 2013.

[11] R. Biglari and A. Bagheri, "PPreDeCon: A Parallel version of Preference Density Connected Clustering Algorithm," International Journal of Computer Applications (IJCA), vol. 107, no. 1, pp. 22-26, 2014.

[12] "OpenMP Application ProgramInterface Version 4.0," July 2013. [Online]. Available: www.openmp.org. [Accessed 17 10 2015].

[13] "KDD Cup 1999 Data," The UCI KDD Archive, 28 October 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed 2 8 2015].

[14] N. Gershenfeld and A. Weigend, "The Santa Fe Time Series Competition Data," Addison-Wesley, 1994. [Online]. Available: http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html#setB. [Accessed 1 11 2015].