ONLINE CLUSTERING OF EVOLVING DATASTREAMS INTO ARBITRARY SHAPED CLUSTERS (CEDAS) USING PARALLEL PROGRAMMING

Alqasimi Saddam Mohammed Saif Nasser, Elankovan A. Sundararajan

Faculty of Information Science and Technology
Universiti Kebangsaan Malaysia
43600 UKM Bangi, Selangor Malaysia.

Gp06082@siswa.ukm.edu.my, elan@ukm.edu.my

ABSTRACT

Recently, stream data mining technologies such as data stream clustering algorithms became highly demanded like K-Means, DBSCAN, K-medoids, CURE, etc. Various applications such as multimedia data, financial transactions, telecommunication, and planetary remote sensing require real-time clustering due to the evolving of data flow continuously. As the advance of networks and the amount of data transmitted in real-time, it becomes harder even though some of the current sequential algorithms keep up with the evolving of data stream in real-time. Some suffer from some weaknesses such as lower processing time in the data stream like clustering algorithm called CEDAS. We proposed to develop a multi-core CPU-accelerated CEDAS using Parallel Programming in MATLAB. The new technique aims to achieve a higher speed while maintaining accurate and pure clusters. The proposed technique has three phases. First, Initialization where all Parameters' and variables are set. The second phase is dataset partitioning, using a decomposition model to break the data set stream into smaller data streams where it can be executed independently as independent threads by the cores using their own smaller data component of the data set. Then, parallel clustering and data gathering, using Parallel for-loop (Parfor) of MATLAB Parallel Computing Toolbox (PCT), we distribute the dataset streams between CPUs' cores/workers which perform CEDAS parallelly before we gather all workers results and save them. We use TicToc statements and ParTicToc tool to get processing time of current and proposed algorithms. The experimental results show P-CEDAS processing time improved by 3.5 to 14.5 times faster than sequential CEDAS. The proposed Parallel CEDAS algorithm (P-CEDAS) quality is assessed using two quality metrics, the Mean-Purity and Mean-Accuracy on synthetic and real datasets each of which with various characteristics, P-CEDAS outperform CEDAS clusters purity and accuracy for most of the cases. Furthermore P-CEDAS achieved higher speed up on four cores of multicore CPU.

1. INTRODUCTION

A data stream is a set of series of instances that can be read once or limited times using finite memory storage and finite computing capability as well (Zimányi & Kutsche 2015). Datastream is generated by various sources continuously in such a way that the data mining process becomes very challenging and requires a particular process of extracting structures of knowledge from evolving and rapid data records where this process is known as data stream mining (Kokate et al. 2018). Datastream clustering is conducted as a preprocessing of data stream mining. Additionally,

data stream clustering is the process of assigning similar data points into groups called a cluster as it arrives online in the data stream without prior information.

Clustering is one of the most important tasks in the data mining process as many fields and applications rely on it, including biology, physics, and marketing. Datastream mining has to follow the necessity of real-time response, limitation of memory, single-pass, and concept-drift detection (Kokate et al. 2018)The data is processed in an incremental manner so that the technique used does not access the entire data. On the other side, it focuses on the most beneficial way to predict the value or class of the new instance/datapoint using previous knowledge from the previous data point. Some examples of data streams are ATM transactions, sensors data, and network traffic (Zimányi & Kutsche 2015)However, clustering of a high multi-dimensional data stream in a real-time became a very challenging process due to data characteristics such as clustering in limited memory and time with single pass over the evolving data streams as well as handling noisy data (Amini et al. 2014), with plenty of applications like network intrusion detection, monitoring environmental sensors, telecommunication, social networks, and web site analysis, etc. (Amini et al. 2014; Amini & Wah 2012).

The wide sources of data streams result in various data formats, and the advance of real-time data streaming makes large volume data in such a way that the traditional clustering algorithm cannot handle the clustering process as required. Researchers have developed new algorithms that can facilitate and accelerate the clustering process, such as k-means, k-medoids, and DBSCAN, etc. (Amini & Wah 2012). In some studies, these algorithms were also modified to fit the data which was being clustered, for instance when the clustering process causes a delay of data streaming, running the algorithms on multi-core CPU in parallel rather than serial was one of the best solutions.

CPUs multicore is an associate computer circuit chip that uses two or additional procedure engines (cores) placed in a single processor. This new approach has proposed to separate the big task of work of applications into small tasks/threads and unfold them over multiple execution cores to make the pc system take advantage of the higher performance and better responsiveness of the system. Computers with more than one processor provide the potential to accelerate application executions (Fotohi et al. 2019). Multi-Threading is an effective method for application developers to take advantage of parallelism in hardware. Completely different threads will run on different processors at the same time utilizing standard APIs and system calls; the programmatic method in which a program requests a service from the kernel of the operating system it's executed on (Rinku & Asha Rani 2017). Through Multicore based Multithreaded Programming (MCMTP), the throughput of an embedded computing system will be increased.

In multicore design, every core contains its own processing resources as a single CPU, except the Global Memory that is shared between the cores. Tasks can be split into smaller tasks to produce simply operable components/threads (Rinku & Asha Rani 2017). A thread could be a unit of execution within a method that's created and maintained to execute a group of threads. Alternatively, threads will be executed from an associate software system to a different one, however, the software system is in most cases accountable to schedule the execution of various threads. Multi-threading increases the efficiency of processor performance with a low-cost memory system (Fotohi et al. 2019). Performance gets increased by gathering the outputs of the individual threads that have been executed on individual cores (Fotohi et al. 2019; Rinku & Asha Rani 2017).

CPUs are dedicated hardware for carrying out the instructions of computer programs. Due to some limitations in processors' heat, architecture complexity, the gap between processing speed and speed of memory access times, etc., CPUs have evolved into multicore CPUs and are being used as highly parallel multi-core processors. General-purpose computing can also benefit from the computing power of multicore CPUs (Al-Ayyoub et al. 2016). General-purpose computing on multicore CPUs became popular with this advance; it was obvious the problems related to vectors and matrices with multi-dimensions vectors become effortless to translate to parallel processors

(Du et al. 2012) because multicore CPUs are capable to deal with this type of tasks. The first multicore CPU was developed by IBM and had two cores on a single chip. Parallel programming languages like Sh/RapidMind and Brook were not easy to use because of the need for understanding the underlying graphical concepts, however, later MPI and OpenMP appeared and allowed programmers to focus on high-performance computing concepts rather than multitask concepts (Du et al. 2012).

Processors (cores) in multicore CPUs are run with high frequencies, using the number of cores that can guarantee high performance with a low cost of energy (Al-Ayyoub et al. 2016). Moreover, the modern CPU pipeline uses the speed of a multi-core CPU to maximum advantage by executing a series of instructions in parallel per clock. Essentially, the pipelined multi-core CPU has various arithmetic units in a sequential and simultaneous manner to perform a chain of complex equations simultaneously in one execution cycle. These pipelines were matched effectively appropriate to scientific computing requirements, and since then they have been developed for this purpose. CPUs can run up to 32 threads. Therefore, many researchers utilize this advantage to improve the performance of many algorithms (Loh & Yu 2014).

Parallel programming detaches the code into sub-blocks and implements them at the same time, which provides a fast processing time in the absence of dependencies between executed applications capable to run up to 32 threads in parallel. Parallel programing has speeded up many algorithms especially those which coped with evolving or high dimension data stream.

The main objective of this project is to accelerate and evaluate a data stream clustering algorithm “Fully Online Clustering of Evolving Data Streams into Arbitrarily Shaped Clusters” (Hyde et al. 2017). Based on (Hyde et al. 2017), the old algorithm named (CODAS) has some limitations during the clustering process of not updating the removed micro-clusters which makes it non-fully online as it does not support clusters evolving during data stream clustering process. CEDAS was a development to CODAS as it becomes fully online, solving those limitations. CEDAS code will be optimized in some logic parts and then, using the power of parallel processing on a multi-core CPU, the clustering process will run in a parallel manner. Further, CEDAS will be evaluated and its productivity will be shown by comparing it with current serial CEDAS.

2. THE CURRENT ALGORITHM CEDAS

Accelerating the clustering of data-streams is considered the most important factor to get the best results when the data stream is evolving rapidly. Many techniques have been successfully applied to solve the clustering of evolving data-streams and this includes (Hyde et al. 2017) which successfully enhances data stream clustering by offering main processes of clustering like joining and separating macro-clusters as they evolve in a fully online manner using CPU. Where the study aimed to implement clustering data into arbitrarily shaped clusters using a fully online clustering technique. the study claimed that an old technique called "clustering of continuous data streams into arbitrary shaped" (CODAS) has some limitations during the clustering process which makes it non-fully online. Moreover, Hyde et al introduce a developed algorithm named (CEDAS) as a shortcut to "Clustering of Evolving Data-streams into Arbitrary Shapes", which proved from the experiments and comparisons with similar its capability of accurate detecting the anomaly in a defined period of time which in order give a great view of possible applications in network security and atmospheric science research as well as its efficiency and ability to automate detection across multiple dimensions that cannot be easily visualized or to present a visualization for primary interpretation by the user. CEDAS algorithm has four main parts, first the Initialization part to establish the structure where the related information of all micro-clusters will be stored. The second part is Assign and Update Cluster, to assign data points and update the

micro-clusters information with the arriving data sample, and that to decide when the data point will be added to a Micro Cluster (MC) or Outlier. The third is Kill MC when no data points are fall near a previous data point this course decreases its energy till to be deleted by this part. The last part is Update MCs Graph and this happens recursively due to the evolving of the data stream, the clustering of data points will be updated till all MCs grouped with each other by Edges according to their intersecting data points in order to perform MCs Graph (Macro-Clusters). CEDAS algorithm four main parts are listed below

CEDAS: initialization.

Input : x , r_0
Create micro-cluster structure containing:
C 1 (*Centre*) = x
C 1 (*Count*) = 1
C 1 (*Macro*) = 1
C 1 (*Energy*) = 1
C 1 (*Edge*) = 1
Set number of micro-clusters to 1
Set modified micro-cluster number, for use updating the graph structure.

CEDAS: update micro-cluster

Input : x , r_0
find distance to nearest micro-cluster center, d_{min}
if $d_{min} < r\ 0$ **then**
    reset micro-cluster *Energy* to 1
    increment number of samples contained in micro-cluster
    if data is within micro-cluster shell **then**
        recursively update micro-cluster center
    end
else
    Create new micro-cluster
end

CEDAS: : kill micro-cluster.

Input C, *Decay*
Reduce all C(*Energy*) by *Decay*
if Any C(*Energy*) < *0* **then**
    Remove micro-cluster
    Remove all edges containing the micro-cluster
    Decrement the number of micro-clusters
end

CEDAS: update graph.

if A micro-cluster has been modified **then**
  if the micro-cluster edge list has changed **then**
    Set a new macro-cluster number throughout the graph
  end
end
if Any micro-clusters have died **then**
  Set new macro-numbers for the sub-graphs of its previous edges
end

CEDAS Algorithm

3 THE PROPOSED APPROACH P-CEDAS

The first practical step in developing CEDAS algorithm is to determine the weaknesses and the parts that can be paralleled by MATLAB Parallel Computing Toolbox (PTC). MATLAB Profiling is a way to identify functions are consuming the most time and parallelizable parts in CEDAS The weakness of the current version lies in its long-time period and low speed that limits the algorithm's ability to keep up with the increase in sample speed, which means increase processing time in general. Also, the way that CPU executes threads sequentially may use 15% of CPU or in other words, only one core is used and that is not efficiently capable with evolving data stream any more as the data stream rate of transfer has increased nowadays. Figure 2.1 shows the side of MATLAB Profiling

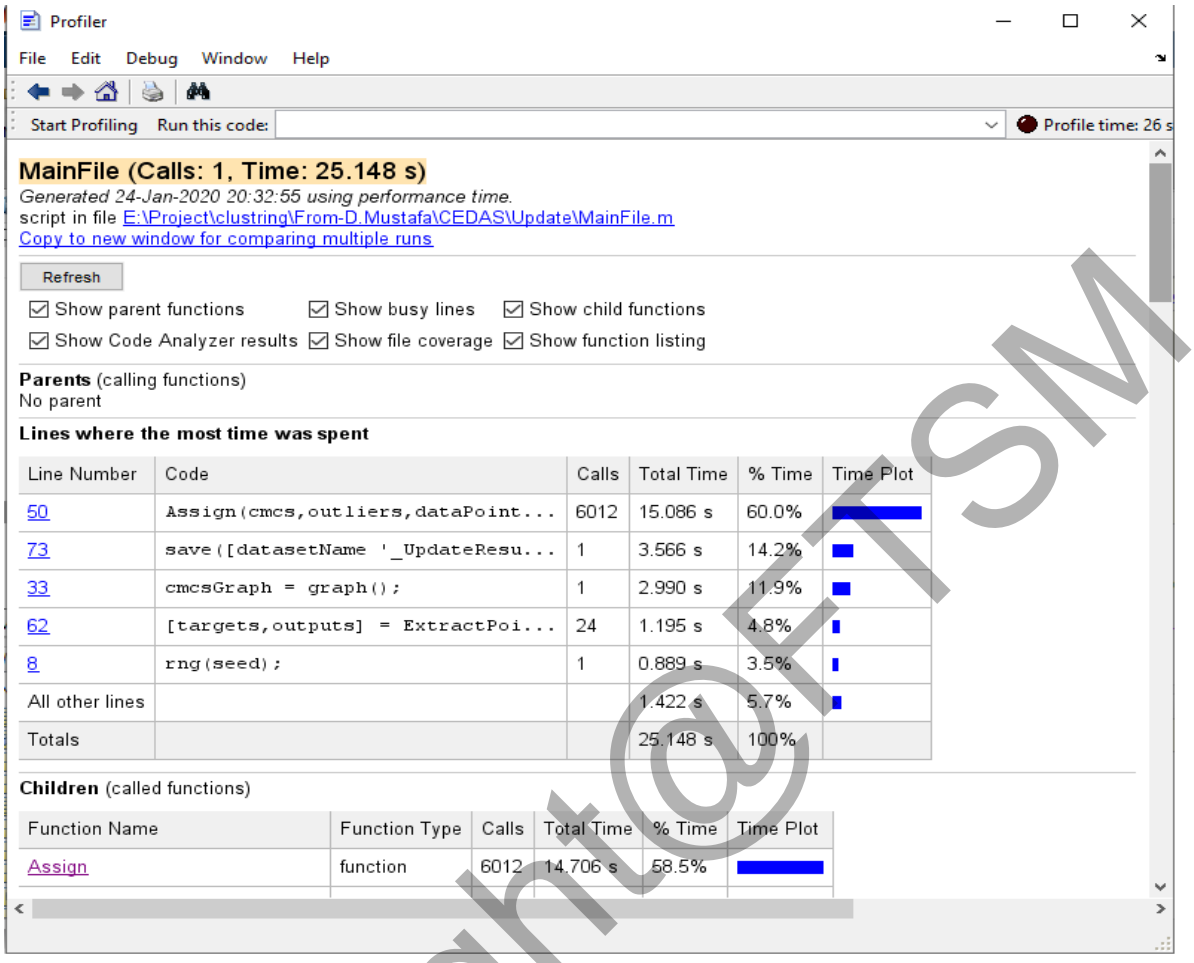


Figure 2.1 Profiling (Run and Time CEDAS Algorithm)

CEDAS algorithm phases are very dependent on each other and have no complex calculations that can be particularly parallelized but we realized from studying some previous researches, to overcome this problem in some studies the serial processes that can not be parallelized were broken down into smaller parts and performed them in parallel while they run in sequential manner locally and independently by multiple threads on multiple cores/processors. CEDAS has three dependent steps that can gain more speed if they improved to run using the same concept. First is Assign and Update Cluster, to assign data points and update the micro-clusters information with the arriving data sample. The second that should come after 'Assign and Update Cluster' is 'Kill MC'. The third part is 'Update MCs Graph' which is happens recursively due to the evolving data stream. The power of MATLAB PTC was used to enhance the detected workload of the algorithm where the compute-intensive portion of the algorithm utilizes the 100% of CPU cores and run in parallel.

There are different programming models to perform parallel multi-threading. In this study, a data decomposition model is used. To analyze the data set, the clustering process needs to process it and break it down into smaller sets. Small sets with tasks can be executed independently as independent threads by the cores but using a separate data component of the data set.

First, an optimization applied on some logic parts in the current serial algorithm to implement it on Multi-Cores CPU and to match MATLAB Parfor conditions as well. Then, we add Parfor to work with the clustering process; we break down the dataset into smaller components as many as workers we have, then the partitioned datasets will be ready to be executed independently on one worker/block in parallel. Each worker will run in a sequential manner but all workers will work on their independent data simultaneously. P-CEDAS terminology is defined in the following:

1. Sample: is referring to whichever one of a data point in the data set.
2. Local density: is the number of samples of each micro cluster.
3. Micro-cluster: a micro cluster formulated only when local density is higher than the threshold.
4. Threshold: describes the minimal of samples within the radius of a micro cluster to formulate a micro cluster otherwise considered an Outlier.
5. Outlier-micro-cluster: when the local density of the micro cluster is less than threshold it's considered an outlier.
6. Macro cluster: containing a number of intersecting micro-clusters
7. Graph structure: the intersecting micro cluster formulate a macro cluster and all the details are recorded in a Cluster.

3.1 P-CEDAS ALGORITHM DESCRIPTION AND STAGES

A description of each stage is illustrated in the following. Algorithm1 is running sequentially however algorithm 2 and 3 are running in parallel and sequential at the same time.

3.1.1 Initialization and parameters setting

This sets P-CEDAS Parameters' values and creates a structure to save micro-clusters' (MCs) information, which happens with the first data sample. Pseudo-code is shown in algorithm1 below.

Algorithm 1: P-CEDAS: Initialization and parameters setting

Parameters setting
Set P-CEDAS parameters
Set MC *radius* to *0.05*
Set Number of data samples to consider 'recent data' to *decay* ∈ {*500,1000*}
Set the min density *Threshold* of MC to *4*

Initialization
Create MC structure = {*Center, Count, Energy, X ∈ (1,2,3….,n)*}
Create Outliers structure = { *Center, Count, Energy, X ∈ (1,2,3….,n)*}
INTI *Counter* to *0*
INIT MCs-Graph {*Nodes, Edges*}

P-CEDAS parameters values are as same as CEDAS Decay, radius and Minimum Density Threshold since they should be set based on expert knowledge (Hyde et al. 2017). 'radius' is neighborhood radius so that any data points located on micro-clusters' radius joins that particular micro-cluster. 'Number of data samples' value is dependent on dataset type where we set its value to 500 in case of artificial datasets and 1000 in case of the real high dimensional dataset. '$Fade = 1/decay$' is the time when the micro cluster entirely disappear due to its energy decrement. 'Threshold' is the minimum neighbors within the ε-neighborhood of a point to be considered a core point.

Creating MCs and Outlier structures, 'Centre' is locating the position of the micro cluster in the data space. 'Count' stores and add up the count of data samples that are allotted to the MC. The 'Count' value is also used to enable updates of micro-cluster centers recursively. 'Energy' used to specify the length of time since the last data has arrived in a micro cluster. Decay algorithm reduces 'Energy' when no data has been received by the micro cluster. Low 'Energy' allow unused MC entirely disappear; X presences one datapoint each iteration of the for loop from 1 till end of dataset. ' Edge' lists the intersecting MCs. 'Counter' is defined as a modified MC number and will be used within the clustering process for updating graph structure. Initializing an MCs graph with two fields 'Nodes' will be the MCs and 'Edges' are the pairs with intersecting Nodes where points fall within intersect part of nodes. the graph is used to define whichever MC belongs to Macro-clusters (Macro), MCs are linked with each other by edges inside the Macro to perform the arbitrary shape clusters.

3.1.2 Dataset partitioning

Keeping in mind that when data is partitioned and stream proportional portions of the data into the workers if all chunks fit into the cache on a core. some of the serial codes' overhead caused by single-core execution will be decreased. However, too small chunks will result a false performance improvement (Hershgal 2010).

P-CEDAS partitions the dataset into small datasets so that each core of the CPU can have its single part of data and run the code independently as it has its data. Number of the produced data sets is depending on the number of CPUs' multiple cores we have. for example, if we have a CPU with 4 multi-core then the data set will be partitioned into 4 datasets. A simple Pseudo-code for this part is shown in algorithm 2 below.

Algorithm 2: P-CEDAS: Dataset Partitioning

Load data set
NumSets = ***Get*** *NumWorkers*
SmallSetSize = (*DataSize* / *NumSets*)
Parfor *i= 1* to *NumSets* **%** Partition Dataset into *NumSets* of *SmallSetSize*
set(1)= data(1, *SmallSetSize*)
set(2)= data(*SmallSetSize, SmallSetSize*2*)
set(3)= data(*SmallSetSize*2, SmallSetSize*3*)

Set(i)= data(*SmallSetSize*(i-1), SmallSetSize*(i+1)*)

End

Using MATLAB object of 'cv' partition class. Our data set has been partitioned and seated to small datasets for workers. After loading the data set, MATLAB 'Random Number Generator' is set to 'default' to avoid random parts of dataset and produce the same part of data every time P-CEDAS running with same NumWorkers. '*NumSets'* is the number of sets will be partitioned from the whole dataset, and as mentioned before *NumSets* will be as NumWorkers we have. '*SmallSetSize'* is the amount of data in each set, and last is the partitioning part where each set will get same amount of data using *SmallSetSize*.

3.1.3 Parallel Clustering and Data Gathering

In this stage, three main parts are 'Assign datapoints and update cluster graph', 'Kill micro-cluster' and 'update clusters relation table'. The three parts are executed by the multiple cores simultaneously as each one of the cores will work independently on its data that were previously prepared in algorithm 2.

Algorithm 3: P-CEDAS: Parallel Clustering and Data Gathering

```
Par (NumWorkers)
Parfor  n= 1 to NumWorkers
Par.tic
Switch (NumWorkers)
Case 1:          % Worker(1) do Clustering Process on set(1)
for Counter= 1 to size (Set(1)) / sampleSpeed
   time = time +1
        for i=1 to sampleSpeed
          Counter = Counter +1
         if not end of stream Then
           Read datapoint X ∈ (1,2,3….,n)
           Assign (CM)
        End
        Kill (CM)
        UpdateClustersRelationTable
        If time equal to datalogSpeed Then
          Create  DataLogged structure of CMs and Outliers
        End
      End
End
Case 2: :          % Worker(2)
   %Repeat case1 on set(2)
Case n: :          % Worker(n)
```

```
    %Repeat case1 on set(n)
Endswitch

result{n} = DataLogged      %Gather and Save all Workers Results
Par.toc
End Pafor
Plot (p)     %plot timing of each Worker

Save clustering result set(i)= data(SmallSetSize*2, SmallSetSize*3)
Set(i)= data(SmallSetSize*(i-1), SmallSetSize*(i+1))

End
```

Clustering process will not be explained in detail since it is as same as in CEDAS (Hyde et al. 2017). P-CEDAS uses ParTicToc tool for timing Parallel for Loops. What makes it special is the ability to observe various running costs that maybe occur when using Parfor loops, the use of each worker as well. 'Par(NumWorker)' creates an object of Par class with a number of iterations (NumWorker). 'Par.tic' is to log the iterations start time and directly called after parfor. The next call is before 'END' Parallelfor, the reason for that is to inform the compiler that the variable is sliced so that it will allocate the end time of each worker separately. 'Par.toc' stops the clustering timing and then plots the result by using Par tool of all workers' overheads, the timing of each worker also saved in a structure.

'Parfor 1 To NumWorkers' starting a par pool with the given number of workers. The statement 'Switch' used to guarantee independent data access. While multi-threads are synchronized and not mixed to execute their part of the code. Since some of variables used inside Parfor are temporary variables, it means they will be released after parfor execution. Therefore, data gathering must be before 'EndParfor' using Sliced structure 'result{n}' where n refers to a worker number. DataLogged stores local clustering results that controlled by the inner for loop condition using the variable 'SampleSpeed' the number of samples to be obtained. Finally, combine the clustering result of all workers and save the clustering result of the whole resulted clusters, outliers, graph, parameters, processing time and counters.

3.2 EXPEREMENNTAL SETUP

Using a free tool in MATLAB (MathWorks) in a PC with Intel® Core™ i5-4572U CPU @ 3.20GHz and 24GB memory running on Windows10.

Two synthetic data sets and one real are selected to evaluate P-CEDAS. Spiral contains 6012 data records; DS2-Class has 9919 of data records as well. However, KDD99ÇUP CUP'99 has around 5 million, only 10% of it used in researches, the same will be used with P-CEDAS as well , which contains 490,000 data records (Qian et al. 2017).

3.3 P-CEDAS PERFORMANCE EVALUATION

In addition to accelerating data clustering processing time, the developed algorithm will pass through an evaluation test to ensure the quality of the output's accuracy and purity of the clustering of the data stream. Accuracy and purity are an evaluation method that uses quality metrics classified in internal and external indices. Moreover, speedup scalability over scaling the number of cores will be justified. These are used in order to guarantee the proposed P-CEDAS algorithm has a better performance compared with the benchmark CEDAS.

3.3.1 Speedup Scalability

The amount of performance increment or speedup is the ratio of time of sequential algorithm to the time of the parallel algorithm. Below is the simple formula to estimate the maximum speedup of a parallel algorithm

This research focuses on the development of new techniques for

$$\text{Speedup} = \frac{T_{serial}}{T_{parallel}} \quad \ldots(3.1)$$

'T_{serial}' is presence the time taken without parallelism, and '$T_{parallel}$' is the computed time with parallelism. speedup depends on problem size as well, which is affected by the size and type of memory from the first place. There are other factors that affect speed up negatively such as inherently sequential part of the algorithm (Buell 2011).

3.3.2 Purity and Accuracy

According to (Hyde et al. 2017), the measurements of accuracy and purity of CEDAS clustering have improved to reach 100% in some parts of the clustering period and to above 90% in general. In this research, the mean measurement of both will be calculated, to guarantee P-CEDAS output quality was not affected by the updates that applied to accelerate the clustering process. To determine the percentage of correct cluster samples number have been assigned to the dominant class, the equations below of both accuracy and purity measurements are used.

$$mean\ purity\ = \frac{\sum_{i=1}^{N} \frac{|C_i^d|}{|C_i|}}{N} \times\ 100\% \quad \ldots(3.2)$$

$$mean\ accuracy = \frac{\sum_{i=1}^{N} |C_i^d|}{\sum_{i=1}^{N} |C_i|} \times\ 100\% \quad \ldots(3.3)$$

where the variable 'Ci' depicts the number of samples in a cluster; 'C_d' represents the number of assigned samples that added up to the main class, and 'N' refers to the number of clusters.

4 RESULT AND DISCUSSION

To asses the alternative P-CEDAS algorithm in different environments. using the high-performance language MATLAB, we run the algorithms illustrated in chapter 3. P-CEDAS parameters are set based on dataset type.With Spiral and DS2_Class; *Decay* = 500; *radius* = 0.05; *Threshold* =4; Sample speed set to one of {5,10,15,20,25} each experiment. And with 10%KDD99ÇUP: *Decay* = 1,000; *radius* = 0.05; *Threshold* =4; Sample speed set to one of {100,125,150,175,200} in each experiment.

Both artificial and real data are being streamed as a data flow. The dataset streamed sequentially on CEDAS, as the clustering process runs in a serial manner. While in P-CEDAS each dataset was partitioned before being streamed to fit the number of workers that P-CEDAS is running, thus, enabling data stream clustering parallelism.

The execution period is timed by 'MATLAB tic/toc' with CEDAS, while P-CEDAS uses 'ParTicToc' for timing Parfor loop. As the experiment has employed the different values of Sample speeds, each of which we mesured the processing time average of performing 'CEDAS' and 'P-CEDAS' ten times using the same Sample speed value with the streamed dataset.

This experiments aims to verify the average of purity, accuracy and processing time. Real or artificial datasets are used to assess the proposed algorithm.

4.1 Spiral Dataset

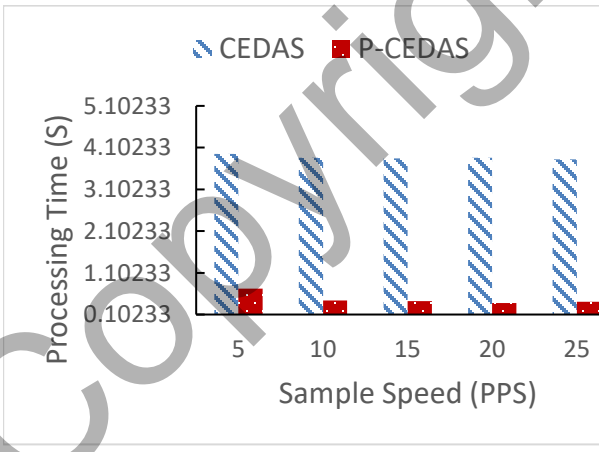


Figure 4.1 Spiral processing time (CEDAS, P-CEDAS)

Figure 4.1 of Spiral dataset illustrates the processing time in seconds of P-CEDAS and CEDAS. When the sample speed various from 5 to 25 point per second (pps), it can be seen that the processing time of the P-CEDAS algorithm values less than 0.8203s, while CEDAS algorithm processing time is relatively high above 3.8182s. Looking from an overall perspective it is readily apparent that P-CEDAS algorithm outperforms CEDAS algorithm regarding the processing time due to the decrement in CPU overhead by partitioning the process into four smaller processes and executing them in parallel using Parfor(PCT) of four workers.

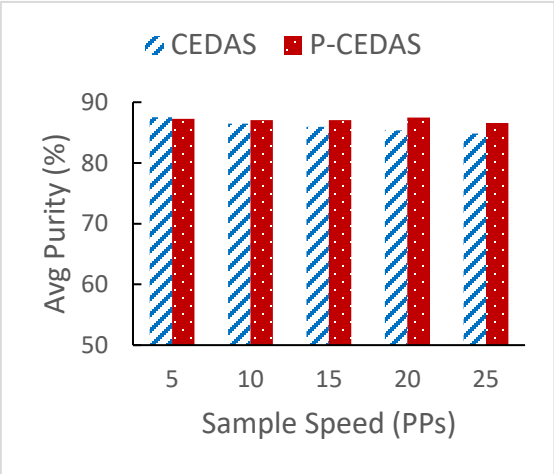


Figure 4.2 Spiral purity

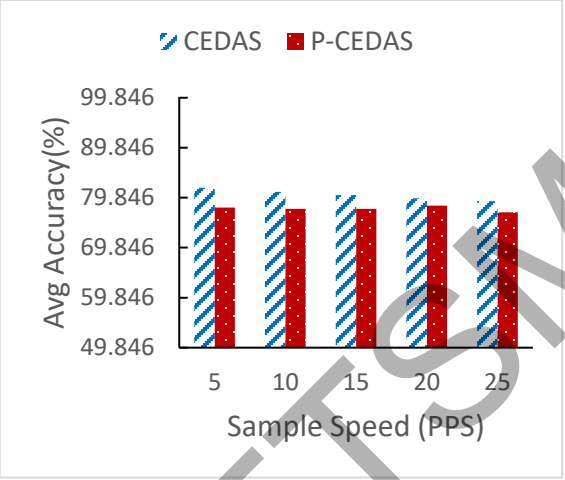


Figure 4.3 Spiral Accuracy

Spiral purity and Acuracy of (CEDAS, P-CEDAS) arbitrary shape clusters

Figure 4.2 illustrates the average purity test of resulted clusters of P-CEDAS and CEDAS using the Spiral dataset. As it shows below P-CEDAS average purity ranges between 87.2% and 86.6% not affected by the number of sample speeds. While CEADS records an average range between 84.84% and 87.5% inversely proportional to the number of sample speeds. Overall, P-CEDASs' average purity has been maintained as CEDAS's, the reason is that a new CM can capture the characteristics of a mixed data object, and is accurately distributed, is introduced. This feature makes the cluster purity of the P-CEDAS algorithm increasingly accurate.

Figure 4.3 illustrates the average accuracy. On the Spiral dataset as shown below P-CEDAS reach an average accuracy above 77.5% on various numbers of sample speed. On the other side, as the number of sample speeds increases, CEDAS fell slightly from 81.84% in 5 pps sample speed to 79.14% in 25 pps sample speed. Unlike CEDAS, P-CEDAS shows relative accuracy decay but maintained during the increases of numbers of sample speed, because, the data points cannot be treated as noise points, and this has maintained cluster accuracy.

4.2 DS2_Class Dataset

Figure 4.4 shows the result of DS2_Class dataset, illustrates the processing time in seconds of P-CEDAS and CEDAS. When the sample speed various from 5 to 25 pps, it can be seen that the processing time period of the P-CEDAS less than 0.7568s, while CEDAS algorithm processing time period is very high above than 6.3568s. Moreover, P-CEDAS gives faster results as it responds positively to the increment of the number of samples. Looking from an overall perspective it is readily apparent that P-CEDAS algorithm outperforms the CEDAS algorithm regarding the processing time due to the decrement in CPU overhead by distributing the process into four smaller processes and executing them in parallel using Parfor (PCT) of four workers.

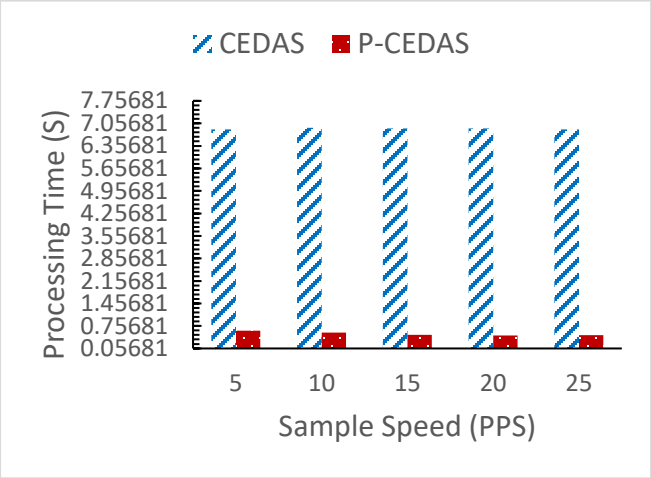
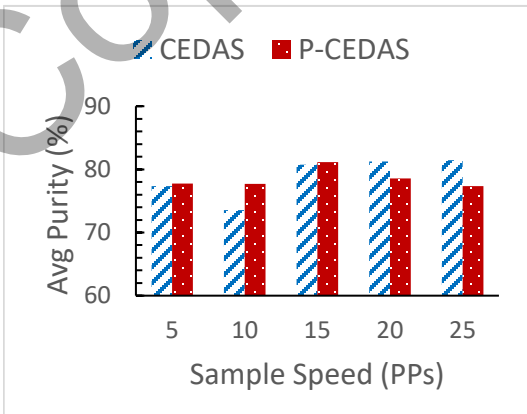


Figure 4.4 DS2_Class processing time (CEDAS, P-CEDAS)

Figure 4.5 illustrates CEDAS and P-CEDAS clusters' purity average on the DS2_Class dataset using sample speeds various between 5 to 25 pps. Looking at the chart there is a slight upward trend in P-CEDAS purity average from 77.78% in 5 until 81.2% in 15 sample speed. Additionally, the bar chart shows a relative increase in purity average of CEDAS instable proportional to the sample speed increment from 77.3% in 5 sample speed to reach the highest average 81.75% when sample speed number is 15. Both, CEDAS and P-CEDAS give a slight fell down in 20 and 25 sample speed numbers but CEDAS maintained an average purity better by 81.46% when sample speed is 25, unlike P-CEDAS which fell down till 77.3% again.

Figure 4.5 illustrates CEDAS and P-CEDAS clusters' purity average on the DS2_Class dataset using sample speeds various between 5 to 25 pps. Looking at the chart there is a slight upward trend in P-CEDAS purity average from 77.78% in 5 until 81.2% in 15 sample speed. Additionally, the bar chart shows a relative increase in purity average of CEDAS instable proportional to the sample speed increment from 77.3% in 5 sample speed to reach the highest average 81.75% when sample speed number is 15. Both, CEDAS and P-CEDAS give a slight fell down in 20 and 25 sample speed numbers but CEDAS maintained an average purity better by 81.46% when sample speed is 25, unlike P-CEDAS which fell down till 77.3% again.

Overall, Although it seems P-CEDAS data partitioning might result in some clustering efficiency loss, P-CEDASs' purity average overperformed CEDAS's three times, and that because of a new CM that can capture the characteristics of a mixed data object, and is accurately distributed, is introduced. This feature makes the cluster purity of the P-CEDAS algorithm sometimes accurate.



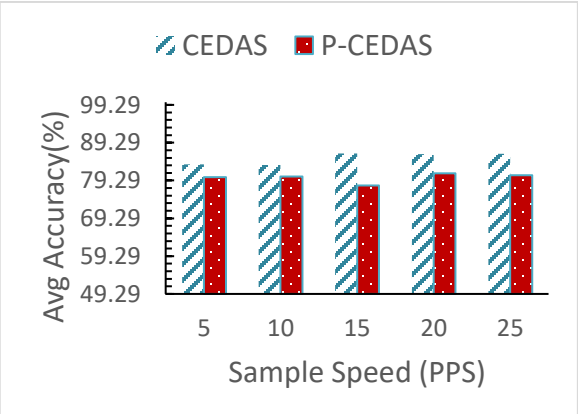


Figure 4.5 DS2_Class Purity……. Figure 4.6 Ds2_Class Accuracy

DS2_Class purity and Accuracy of (CEDAS, P-CEDAS) arbitrary shape clusters

Figure 4.6 illustrates the average accuracy. On the S2_Class dataset as shown below P-CEDAS and CEDAS are being asses during a various of 5 pps until 25 pps of sample speeds.

It's obvious in the charts below, P_CEDAS reaches an accuracy average above 77.5% and below 81.2% on various numbers of sample speed pps. On the other side, as the number of sample speed increases, CEDAS upward trends slightly from 83.84% in 5 sample speed pps to 86.14% in 25 sample speed. P-CEDAS shows relative accuracy improvement as the sample speed number is increased because, the data points cannot be treated as noise points, and this has improved cluster accuracy. Generally, P-CEDAS shows relative accuracy decrement as data is too partitioned into small sets and that causes efficiency to lose.

4.3 KDD99ÇUP Dataset

Figure 4.7 showing the 10%of the KDD99ÇUP dataset below illustrates the processing time in seconds of P-CEDAS and CEDAS. When the sample speed various from 100 to 200 pps. CEDAS, the processing time period increases dramatically from 2,457s in 100 sample speed to 5,298s in 200 sample speed, a high sample speed number means more samples carried per second. This means increase in the period of real-time processing CEDAS due to the limit in CPU memory bandwidth. However, P-CEDAS shows stable processing time, less than the 1,600s during the changeable sample speeds in the experiment and that because P-CEDAS uses an appropriate data-parallel model helps to decrease the memory bandwidth limitation by resulting more work done through 4 workers at the same time.

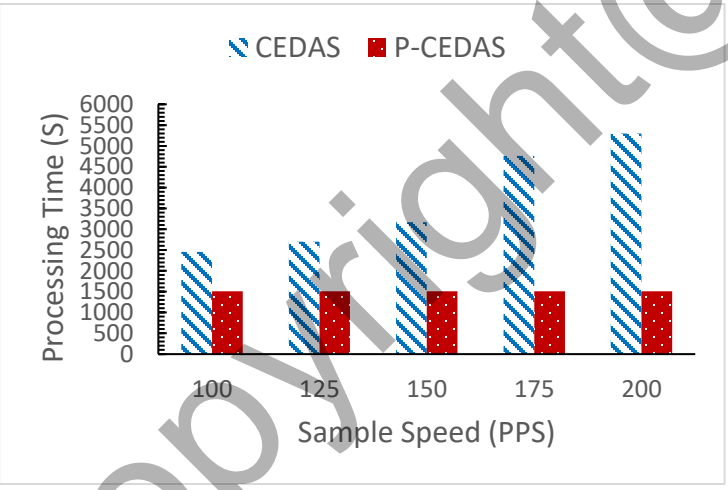


Figure 4.7 KDD99ÇUP processing time (CEDAS, P-CEDAS)

To sum-up, the Parallel CEDAS overcome sequential CEDAS, which suffers from additional overhead as far as the number of sample speed becomes greater. However, P-CEDAS does not affected by the changeable number of sample speed as it did with the synthetic data sets due to the high dimensions found in KDD99ÇUP.

Figure 4.8 illustrates CEDAS and P-CEDAS clusters' purity average on the 10%KDD99ÇUP dataset using various numbers of sample speed. Looking at the chart although P-CEDAS average purity increases proportionally from 87.44% in 100 sample speed pps to 97.57% in 175 samples speed pps, there is a slight downward trend in P_CEDAS average purity 93.0% when the sample speed number is 200.While CEDAS purity average increases proportionally from 9.29% in 100 sample speed to above 97% in 200 sample speed. We conclude that, P-CEDAS reaches a total purity percentage of 94.6 overperforming CEDAS average 92.5% purity, due to the feature that

makes cluster purity more accurate. Because of a new P-CEDAS CM that can capture the characteristics of a mixed data object and is accurately distributed.

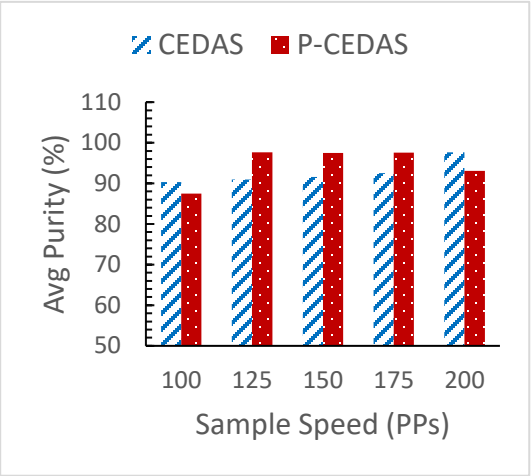


Figure 4.8 KDD99ÇUP-Purity

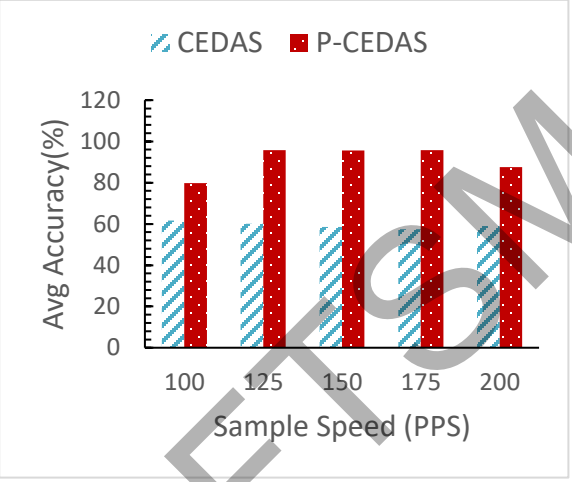


Figure 4.9 KDD99ÇUP-Accuracy

KDD99ÇUP- Purity and Accuracy average of (CEDAS, P-CEDAS) arbitrary shape clusters

Figure 4.9 illustrates CEDAS and P-CEDAS clusters' average accuracy on the 10%KDD99ÇUP dataset using various numbers of sample speed. As shown in the chart P-CEDAS average purity range between 79.77% in 100 sample speed pps to 95.74% in 175 samples speed pps, there is a slight downward trend in P_CEDAS purity average 87.6% when the sample speed number is 200. While CEDAS purity averages show an unstable range of 61.58% in 100 sample speed to 58.9% in 200 sample speed.

Overall, P-CEDAS reaches a remarkable total purity percentage of 90% overperforming CEDAS with an average of 59.3% accuracy, P-CEDAS shows relative accuracy improvement because, the data points cannot be treated as noise points, and this has improved cluster accuracy.

4.4 SPEED UP AND SCALABILITY EVALUATION OF P-CEDAS EXECUTION TIME

The following experiment has been intended to examine P-CEDAS speedups scalability. The idea is performing P-CEDAS on an ascending number of available cores and measuring the gained speedup achieved.

Table 4.1 P-CEDAS Speedups

N	1	2	3	4
S	0.9442	1.7363	2.4073	2.6401

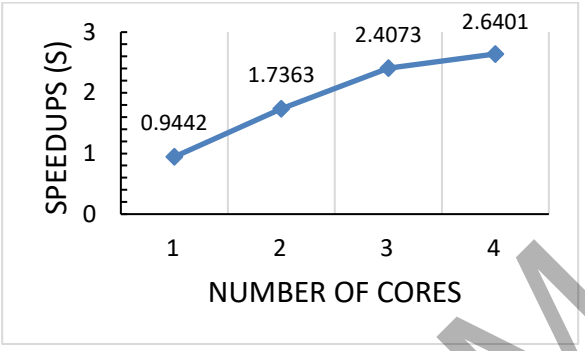


Figure 4.10 P-CEDAS speedups over 4 cores

By involving more cores the number of threads increasing as well, which means speed up supposed to keep up, and that would prove the scalability of P-CEDAS, otherwise, if the algorithm fails to keep up would means it's not scalable enough. The following table provides speedups of P-CEDAS.

Where *N* is the number of workers/cores and *S* is the computed speed up via the equation above. From the speedup's escalation gotten we can claim that P-CEDAS is scalable and speedup increase as more worker/cores are involved.

The graph below shows the speed up at which P-CEDAS gains more acceleration over 4 workers from 1 to 4. The graph shows P-CEDAS processing time reach 0.9442s on one core and 2.6401s on four cores. Overall, there is adoption on the number of workers/cores involves more computational threads that cooperate on carrying processing overhead. Thus, shows a typical success of speedup P-CEDAS as more cores involved in the process result in speedup boost up.

5 CONCLUSION

Many conventional data mining algorithms including data stream clustering algorithms are suffering from the limit memory bandwidth and increase in the period of real-time processing of the continuous data flow. The gap between big data growth and processors improvement causes CPUs to suffer more, indeed the capability of sequential processing is not the best solution to fully use the power of processors. The parallel processing approach is an extremely good solution when it comes to acceleration and full uses of processors' power. Therefore, this research aimed to use parallel processing to accelerate and evaluate an evolving data stream clustering algorithm CEDAS. In which it performs clustering process over multiple threads on multi-core CPU reducing processing period. The chosen algorithm has been studied and developed it to appropriate the new method of processing has done successfully and parallelization achieved through MATLAB (PCT) and Multi-core CPU. The data stream is mapped to a number of workers that run the clustering process in completely independent divisions, the parallel-for loop of MATLAB (PCT) carried out the small processes in each worker simultaneously, thus, processing time period and processor overhead have notably decreased.

Using synthetic and real data flows to examine the proposed parallel CEDAS validity, both sequential CEDAS and P-CEDAS performed on the same machines over various cases of sample speed values, to avoid false acceleration or less accurate evaluations. The result clusters are evaluated and compared with the serial version algorithm in terms of the quality purity and accuracy of clusters. The proposed algorithm P-CEDAS overcome CEDAS in the experiments and shows a remarkable speedup about 5 to 9 times faster using spiral, 11 to 14 times faster using DS2_Class, and 1.6 t0 3.5 times faster with KDDCUP99. Despite that, clusters' purity and accuracy have varied between 77.3% and 97.6% purity as well as 76.9% and 95.8% accuracy; various percentages illustrate the lowering and improvement in various cases of the experiment.

Even though the objectives were achieved, the proposed algorithm has some limitations that are worthy to be mentioned. We observed that P-CEDAS speedup falls down when sample speed is small, especially with high dimensional data. P- CEDAS recorded a modest speedup on single-core, which calls into question that the development was not good enough despite the speedup results on 4 cores. Although Parfor loop of MATLAB (PCT) is easy to use when comparing it with parallel programming languages, it has more limitations in control and conditions of use.

REFERENCES

Al-Ayyoub, M., Yaseen, Q., Shehab, M. A., Jararweh, Y., Albalas, F. & Benkhelifa, E. 2016. Exploiting GPUs to Accelerate Clustering Algorithms. *Computer Systems and Applications (AICCSA), 2016 IEEE/ACS 13th International Conference of* 1–6.

Amini, A., Saboohi, H., Ying Wah, T. & Herawan, T. 2014. A fast density-based clustering algorithm for real-time internet of things stream. *The Scientific World Journal* 1. doi:10.1155/2014/926020

Amini, A. & Wah, T. Y. 2012. DENGRIS-Stream: A Density-Grid based Clustering Algorithm for Evolving Data Streams over Sliding Window. *International Conference on Data Mining and Computer Engineerging* 206–210.

Buell, D. 2011. In Praise of An Introduction to Parallel Programming. *Aims.Me.Cycu.Edu.Tw*. doi:10.1007/978-1-4471-2736-9

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G. & Dongarra, J. 2012. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing* 38(8): 391–407.

Fotohi, R., Effatparvar, M., Sarkohaki, F. & Behzad, S. 2019. An improvement over threads communications on multi-core processors. *arXiv preprint arXiv:1909.11644*.

Hershgal, D. 2010. Intel Guide for Developing Multithreaded Applications. *Intel Technology Journal* 12(01): 27–38. doi:10.1535/itj.1201.03

Hyde, R., Angelov, P. & Mackenzie, A. R. 2017. Fully online clustering of evolving data streams into arbitrarily shaped clusters 383: 96–114. doi:10.1016/j.ins.2016.12.004

Kokate, U., Deshpande, A., Mahalle, P. & Patil, P. 2018. Data Stream Clustering Techniques, Applications, and Models: Comparative Analysis and Discussion. *Big Data and Cognitive Computing* 2(4): 32. doi:10.3390/bdcc2040032

Loh, W. K. & Yu, H. 2014. Fast density-based clustering through dataset partition using graphics processing units q , qq. *INFORMATION SCIENCES* 308: 94–112. doi:10.1016/j.ins.2014.10.023

Qian, Q., Zhao, S., Xiao, C. & Hung, C. 2017. Multi-level Grid Based Clustering and GPU Parallel Implementations (4). doi:10.1109/ISPAN-FCST-ISCC.2017.75

Rinku, D. R. & Asha Rani, M. 2017. Analysis of multi-threading time metric on single and multi-core CPUs with Matrix Multiplication. *Proceedings of the 3rd IEEE International Conference on Advances in Electrical and Electronics, Information, Communication and Bio-Informatics, AEEICB 2017* 152–155. doi:10.1109/AEEICB.2017.7972402

Zimányi, E. & Kutsche, R. D. 2015. Business Intelligence: 4th european summer school, eBISS 2014 Berlin, Germany, July 6-11, 2014 tutorial lectures. *Lecture Notes in Business Information Processing*, hlm. Vol. 205. doi:10.1007/978-3-319-17551-5