Data stream clustering for low-cost machines

Christophe Cérin, Keiji Kimura, Mamadou Sow

Version of Record: https://www.sciencedirect.com/science/article/pii/S0743731522000831
Manuscript_57f5b8aba1f6ced52ddc4706f8b22c7c

Data Stream Clustering for Low-Cost Machines

Christophe Cérin[a], Keiji Kimura[b], Mamadou Sow[a]

[a] *Sorbonne Paris Nord university, LIPN UMR CNRS 7030, 99, avenue J.B. Clément, Villetaneuse, 93430, France*
[b] *Waseda university, Faculty of Science and Engineering, School of Fundamental Science and Engineering, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555, Japan*

Abstract

Nowadays, the operations performed by the Internet of Things (IoT) systems are no more trivial since they rely on more sophisticated devices than in the past. The IoT system is physically composed of connected computing, digital, mechanical devices such as sensors or actuators. Most of the time, each of them incorporates a logical arithmetic unit that can pre-compute or compute on the device. To extract value from the data produced at the edge, processing power offered by cloud computing is still utilized. However, streaming data to the cloud exposes some limitations related to the increased communication and data transfer, which introduces delays and consumes network bandwidth. Clustering data is one example of a treatment that can be executed in the cloud. In this paper, we propose a methodology for solving the data stream clustering problem at the edge. Data Stream clustering is defined as the clustering of data that arrive continuously, such as telephone records, multimedia data, sensors data, financial transactions, etc. Since we use low-cost and low-capacity devices, the objective is, given a sequence of points, to construct a good clustering of the stream using a small amount of memory and time. We propose a 'windowing' scheme, coupled with a sampling scheme to respect the objective. Under the experimental conditions, experiments show that the clustering solutions can be controlled, with difficulties for time-stamped data but not for random data or data with well-delimited clusters. The main advantage of our schema is that we are clustering data "on the fly" with no knowledge or assumption regarding the available data. We do not assume that all the data are known before a treatment batch by batch. Our schema also has the potential to be adapted to other classes of machine learning algorithms.

Keywords: Edge AI, Machine-Learning Algorithms, Online data stream clustering, Experiments on heterogeneous and low cost hardware.

1. Introduction

Edge computing is an emerging paradigm to meet the ever-increasing computation demands from pervasive devices such as sensors, actuators, and smart things. Though the edge devices can execute complex applications, some applications must migrate to centralized servers or clouds. But we would like to avoid this migration. In the case of smart buildings, if we decide to give more control over data privacy and security to residents, one may choose to avoid sending data produced by the residents of a building into any cloud (GAFAM - Google Amazon Facebook Apple Microsoft, or BATX - Baidu, Alibaba, Tencent, Xiaomi clouds). This new vision entails new engineering of the building data, processors, and overall controllers. This new positioning

raises several further research questions about 1- the efficient use of the computing, networking, and storage capabilities owned by the building residents and visitors; 2- the performances of the devices installed in the building. Both are central if we want to effectively learn from the data produced in-situ inside a building.

A Co2 sensor inside a building produces data, and we may be interested in clustering the data to determine some anomalies. Our motivation is to cluster data at the edge and on devices present at the edge for such a use case. By definition, such devices have low capacities in terms of memory and CPU speed, and they may be constituted of computing facilities of the residents (Home PC, tablets…).

For the reader convenience, we put at the end of the paper a table for notations and a table for acronymes used in the paper.

Data Stream clustering is defined as the clustering of data that arrive continuously. Since we require low-cost devices to solve the problem, we need to control the memory space used to compute the solutions. The first step is to show that clustering can effectively occur in a small memory space (not caring about the number of passes). Small-Space is a divide-and-conquer algorithm that divides the data, S, into l pieces, clusters each one of them, and then clusters the centers obtained. The underlying idea is as follows. So, if M is the size of memory, we need to partition $n = \|S\|$ into l subsets such that each subgroup fits in memory (n/l) and so that the weighted $l.k$ centers also fit in memory, $l.k < M$. In brief, we need to satisfy $(n/l) + l.k < M$. But, second, such an l may not always exist. Anyway, l and k need to be chosen carefully.

The paper proposes a new small-space methodology to solve the data stream clustering problem on resource-constrained devices, those present at the edge, in a building to continue with our initial motivation.

Thus, the contributions of the paper are twofold. First, we propose a small-space framework for the "on the fly" data stream problem, which only depends on one parameter $W = M/c$, where c is a constant factor. We mean that $W = \Omega(M/c)$ to keep the notation used in the analysis of algorithms. Moreover we do not assume that we know all the data before the treatment batch by batch. Second, we experimentally demonstrate that we can control the solutions in comparing them to the solutions of clustering algorithms when we know the whole data in advance. We conduct experiments with low-cost and low-end devices such as Raspberry Pi Zero and Pi 4, as well as with an FPGA circuit, in a realistic setting, based on an MQTT (Mosquito) server to publish the data.

The organization of the paper is as follows. Section 2 is about related work, and it introduces data stream clustering algorithms, the low-cost machine, and heterogeneous concepts. Section 3 states the problem and the algorithmic solution. Section 4 introduces the series of experiments and the lessons learned from the experiments. Section 5 concludes the paper.

2. Related work

2.1. Data Stream clustering algorithms

Clustering is a crucial data mining task, and it refers to the problem of partitioning a set of observations into clusters such that the intra-cluster observations are similar (or close) and the inter-cluster observations are dissimilar (or distant). The other objective of clustering is to reduce the complexity of the data by replacing a group of observations (cluster) with a representative observation (prototype/centroid).

When the data continue to arrive at a more or less rapid rate, we can not access them randomly to generate the clustering solution. We refer to these types of data as data streams. The

data stream clustering problem requires a process capable of partitioning observations continuously while considering restrictions of memory and time, particularly for IoT and edge computing. In the literature of data stream clustering methods, many algorithms use a two-phase scheme that consists of an online component that processes data stream points and produces summary statistics and an offline component that uses the summary data to generate the clusters. An alternative class is capable of generating the final clusters without the need for an offline phase.

One can also consider the following models, on the methodological plan, to build data streams algorithms:

- Data stream model: if the stream has length n and the domain has size m, algorithms are generally constrained to use space that is logarithmic in m and n. Algorithms make only some small constant number of passes over the stream, sometimes just one.
- Sliding window model: in this model, the function of interest is computed over a fixed-size window in the stream. As the stream progress, items from the end of the window are removed from consideration while new items from the stream take their place.

In this paper, we consider a sequence of potentially infinite, non-stationary data. We mean that we assume that the probability distribution of the unknown data generation process may change over time. For instance, we will experiment with time-stamped data in a 2D space, and also with 2D data, chosen randomly in the same range of values.

Due to the memory limitation of our computing devices we do not consider streaming platforms such as MOA [Bifet et al. (2011)] and distributed streaming platforms such as Spark Streaming [Zaharia et al. (2013)] and Flink [Friedman and Tzoumas (2016)].

2.1.1. Pionnering work

In this subsection, we review the pioneering work on data-stream clustering. BIRCH [Zhang et al. (1996)] algorithm incrementally and dynamically clusters multi-dimensional data points to try to produce the best quality clustering with the available resources (i.e., memory and time constraints) by making a single scan of the data and to improve the quality further with a few additional scans. One can note that the BIRCH method is not designed for clustering data streams and cannot address the concept drift problem. The essential characteristic of the BIRCH is to introduce a new data structure called a clustering feature (CF) as well as a CF-tree. The CF can be regarded as a concise summary of each cluster. The complexity of the algorithm is $O(N)$ since one pass suffices to get a good clustering (results can be improved by allowing several passes).

COBWEB [Fisher (1987)], [Fisher (1996)] is an incremental clustering technique that keeps a hierarchical clustering model in the form of a classification tree. COBWEB descends the tree for each new point, updates the nodes along the way, and searches for the best node to put the point on (using a category utility function). There are four basic operations COBWEB employs in building the classification tree. The operations are Merging two nodes, Splitting a node, Inserting a new node, and Passing an object down the hierarchy. As with IRSH, COBWEB is a hierarchical stream method.

CluStream [Aggarwal et al. (2003)] divides the clustering process in two phases: (a) Online, the data will be summarized; (b) Offline, the final clusters will be generated. CluStream uses micro-clusters that are temporal extensions of BIRCH [Zhang et al. (1996)] cluster feature vector, so that it can decide if a micro-cluster can be newly created, merged, or forgotten based

in the analysis of the squared and linear sum of the current micro-clusters data-points and timestamps, and then at any point in time, one can generate macro-clusters by clustering this micro-clustering using an offline clustering algorithm like k-means [Lloyd (1982)], thus producing a final clustering result.

2.1.2. A recent synthesis of data stream clustering methods

The reference [Ghesmoune et al. (2016b)] is a survey of state-of-the-art approaches for clustering data streams. The paper is, in part, related to methodologies to device algorithms, and this point constitutes one remarkable point. Figure 1, from [Ghesmoune et al. (2016b)], illustrates the different methodologies and algorithms. These algorithms are categorized according to the nature of their underlying clustering approach.

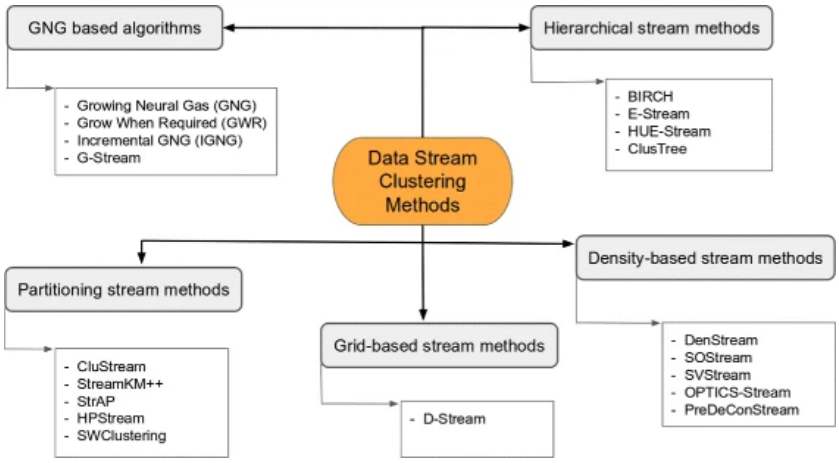


Figure 1: Data stream clustering methods [Ghesmoune et al. (2016b)]. The presented algorithms categorized according to the nature of their underlying clustering approach.

Neural gas is an artificial neural network inspired by the self-organizing map and introduced in 1991 by Thomas Martinetz and Klaus Schulten [Martinetz and Schulten (1991)]. The neural gas is a simple algorithm for finding optimal data representations based on feature vectors. Fritzke in [Fritzke (1994)] describes the growing neural gas (GNG) as an incremental network model that learns topological relations by using a "Hebb-like learning rule." We do not use an artificial neural network in our work. Typically, it is used for finding topological structures that closely reflect the structure of the input distribution, which is not the aim of our work.

We skip the hierarchical stream methods such as the BIRCH algorithm because it has been introduced above as well as CluStream. We also forget the density-based stream methods not because they are not adapted to our problem but because we have chosen to work with the nearest cluster center notion using Euclidean distance. Most of the grid-based stream methods consist of an online component that processes input data stream and produces summary statistics and an offline component that uses the summary data to generate clusters. The method implies that we need additional data structures, which is not our choice. Multiple stream structures introduce tedious evaluation of data stream algorithms.

A partitioning-based clustering algorithm organizes the objects into some number of partitions, where each partition represents a cluster. The clusters are formed based on a distance

function like the k-means [Lloyd (1982)] algorithm, which leads to finding only spherical clusters. It is in this category that we find our approach. The originality of our work is that it does not require additional complex data structures, contrary to the algorithms presented in Figure 1 for the partitioning stream methods. We only use an in-place sorting step, iteratively, after catching W data.

Moreover, we estimate that we do not need the optimization as the one presented in [Sculley (2010a)] because the publication rate of data, and the performance of the MQTT server that collects the data overlap the performance time of the treatment of one batch of data.

2.1.3. Advanced methods

In [Attaoui et al. (2021)], authors tackled the subspace clustering problem, which discovers clusters embedded in multiple, overlapping subspaces of high dimensional data. They proposed the S2G-Stream algorithm based on growing neural gas and soft subspace clustering. Experiments on public datasets demonstrated the ability of S2G-Stream to detect relevant features and blocks and provide the best partitioning of the data.

In [Attaoui et al. (2020)], authors investigated the clustering problem according to multiple objectives. Most of the clustering algorithms follow only one cluster validity measure. Given different data properties, a single validity measure does not work well for all datasets. In the paper, the authors introduced the MOC-Stream algorithm based on Multi-objective clustering and data stream concepts. The goal of MOC-Stream is to find clusters by applying several algorithms corresponding to several objective functions.

Finally, in [Ghesmoune et al. (2016a)], authors presented a new algorithm, called G-Stream, for clustering data streams by making one pass over the data. G-Stream is based on the growing neural gas method that allows us to discover clusters of arbitrary shapes without any assumptions on the number of clusters. By using a reservoir and applying a fading function, the quality of clustering is improved. Experiments on public datasets validated the improvements.

These three papers illustrate the variety of approaches and research questions in the field of clustering algorithms. In our case, we investigate the "classical" problem with the objectives to control the memory used by the algorithm, and such that the algorithm is implementable with companion IoT technologies, i.e., easy to implement because the underlying idea is kept simple.

The reference [Carnein and Trautmann (2019)] is yet another seminal synthesis on data stream clustering. Matthias Carnein also maintains a dedicated Web site[1] with pointers to codes.

Scikit-learn [Pedregosa et al. (2011)], a simple and efficient Python tools for predictive data analysis, contains many data stream clustering algorithms such as BIRCH [Zhang et al. (1996)] and MiniBatchK-Means [Sculley (2010b)]. The Scikit-learn Web page for clustering[2] also introduces some other clustering methods not related to data-stream clustering.

The MiniBatchKMeans is a variant of the KMeans algorithm, which uses mini-batches to reduce the computation time while still attempting to optimize the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

In [Béjar Alonso (2013)] Béjar Alonso Javier digs into the advantage of the Mini batch k-means algorithm to reduce the computational cost by not using all the datasets each iteration

[1] https://www.matthias-carnein.de/streamclustering

[2] https://scikit-learn.org/stable/modules/clustering.html

but a subsample of a fixed size. The purpose of the paper is to perform empirical experiments using artificial datasets with controlled characteristics to assess how much cluster quality is lost when applying this algorithm.

2.2. Low-cost machines

The low-cost paradigm considers the following principles, which in our case correspond to the following non-exhaustive list:

1. Radical simplification in a good or service reduced to its basic functionality (thus devoid of generality, subtlety).
2. Drastically reduced input costs (cheap materials and components, minimized labor costs for operation and maintenance).

2.3. Heterogeous computing

Heterogeneous computing [Zahran (2019), and Hwu (2016), and Terzo et al. (2019)] refers to systems that use more than one kind of processor or core. These systems gain performance or energy efficiency not by adding the same type of processors but by adding different types of coprocessors.

Nowadays, many new processors now include built-in logic for interfacing with other devices (SATA, PCI, and memory controllers), as well as programmable functional units and hardware accelerators (GPUs, cryptography coprocessors, neural network processors, programmable network processors, etc.).

Heterogeneous computing systems present new challenges not found in typical homogeneous systems. For instance, the memory interface [Howes et al. (2016)] and hierarchy issues are essential. Since compute elements may have different cache structures, cache coherency protocols, and memory access may be uniform or non-uniform memory access (NUMA), the coordination is challenging. The ability to read arbitrary data lengths as some processors/units can only perform byte-, word-, or burst accesses also introduce differences. Thus, the problem is an interconnect problem.

3. Problem statement and algorithmic solution

3.1. Problem statement

The problem we are dealing with is schematized in Figure 2. Sensors continuously send data to a server governed by an Internet of Things protocol (MQTT). The machine performing the clustering, namely "Processor for clustering" in Figure 2, subscribes to the messages received by the server and performs the clustering. The publisher (MQTT server) and the processor that executes the clustering are low-cost machines. A single machine can host both functionalities, and, in this case, one can locate the service for collecting data from sensors and the service to cluster data on the same device. The main advantage of this co-location is to keep the production and the treatment as close as possible to limit communication.

Finally, note that for reasons of clarity, we have chosen to use only one processing machine (doing the work of clustering), but several sources can feed the processing machine. Thus, the issue of scaling and dimensioning the architecture is not considered in our paper. Also, we believe that the proposed architecture is adequate for using phones or tablets as processing machines because the MQTT protocol, for example, has already been implemented on these devices.

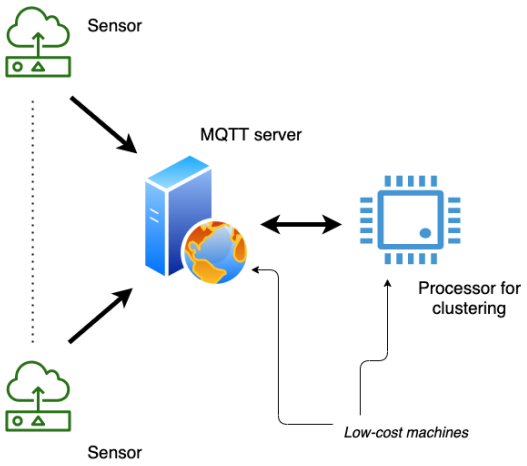


Figure 2: Diagram of interactions and problem statement illustrated.

3.2. *Our solution*

Our generic framework is presented in Algorithm 1. We first consider a window of W points from the data stream, and we cluster it (lines 1 and 3). Notice that we have experimented with k-means [Lloyd (1982)] and k-means++ [Arthur and Vassilvitskii (2007)], hence the explicit mention in the algorithm. We sort the data (windowing of size W points) according to values of the coordinates (line 4). Notice that the sorting step is in place and it does not require supplementary memory. Then we cancel p values, chosen regularly to preserve the diversity, the distribution of data (line 5). It remains to read p fresh points and go to step 2 (line 6). Note that our algorithm has two input parameters: the window size W and the number of points to eliminate at each iteration step. This point makes the difference compared to other algorithms. The choices for the corresponding values are of first importance to ensure a "good" convergence, meaning that the centroids adapt continuously.

In Algorithm 1 we follow the sliding window model, and we repeat infinitely first a clustering step and second a sorting step to capture the streaming nature of the arrival of data. Both k-means (and their variants) and sorting have been well studied in the past. Thus, there are multithreaded versions of k-means and sorting that we can use if the low-cost machine has few cores and accelerates the execution times. This point is one competitive advantage of working with proven algorithms. Note that our codes, for instance sorting (see the Github link below), are ready for multithreading on an SMP (Shared-memory Multiprocessing) machine since they use OMP (OpenMP) directives.

Notice that streaming clustering algorithms (see the related work section) cannot serve as baseline algorithms because, to the best of our knowledge, they are not designed with the requirement of using low-cost machines in mind nor in using memory-constrained devices. We are guessing that the comparison would not be fair in this case. In brief, the properties of our generic framework are:

- The algorithm is based on offline clustering (traditional Kmeans or kmeans++ or KNN can be used);

- When we repeat, round after round, the clustering of step 3, it is only $p << W$ data that must be inserted; hence control of the memory space;
- Each sort in step 4 can be done in parallel (if the hardware does permit it); Sorting, coupling with the next step, helps in preserving diversity in the data;
- The idea of eliminating p data by taking them in a regular way corresponds to the idea of preserving diversity in the input.

Algorithm 1 Main algorithm of our generic framework

Require: W: window size in points' number that fit in memory;
Require: p: number of points to remove at each iteration; $p \ll W$;
1: n = Read(W inputs from the data stream);
2: **loop**
3: Cluster the n input data (for instance with k-means or k-means++);
4: Sort the n data;
5: Evict, in the sorted data, p data (for example by considering regular intervals);
6: Read p new data from the data stream to form a new window of size W;
7: **end loop**

To summarize, the work's main contribution is in clustering sensor data given in a streaming manner with multiple devices, each of which has limited computation power and memory resource. In contrast to most of the work mentioned in the Related work section, we devise an online algorithm in the sense that data arrives "on the fly." If we take, for instance, the paper [Sculley (2010a)] on the mini-batch algorithm, authors assumed that data are all known in advance and are exploited batch b by batch b randomly. This view permits authors to select b examples picked randomly from input X and used per-center learning rates for fast convergence searching for the best centroid attached to a given point. Convergence properties follow closely from a prior known result. The fact that is not fair between this algorithm and our algorithm is that we can pick data as far as possible in the input. Of course, this manner limits the amount of memory, but it may impact the convergence. In our case, we do not assume that it is possible to check "the horizon." Our horizon is limited to b inputs, and again, we treat data "on the fly."

Note also that the schema available for Algorithm 1 has the potential to be used for other machine learning algorithms. Indeed, step 3 could be replaced by an offline supervised algorithm or an offline unsupervised algorithm, Logistic regression or Outlier Detection to cite few examples.

4. Experiments

4.1. Settings and experimental plan

This subsection introduces the devices we use for testing, and it explains the experimental plan. Potentially we can utilize either k-means or k-means++ as the core clustering algorithm. This choice gives two dimensions of our research plan. We can also wonder if we need to reinitialize the centers at each iteration or if only one initialization step is enough to provide

```
pi@raspberrypi:~/mosquitto $ time /bin/bash test.sh
real 96m30,030s
user 12m33,585s
sys 53m7,764s
```

Figure 3: Execution time of the script on a Raspberry PI-4

good results regarding the final centroids. This choice introduces two other dimensions in the research plan.

The research plan also compares the centroids obtained through our algorithm and the centroids obtained by k-means (k-means++) if all the data are known in advance. Of course, we compare k-means with k-means algorithms or k-means++ with k-means++ algorithms.

At last, we use 2D points that are randomly generated for the x and y dimensions and 2D points with the x dimension being a timestamp and the y being a random integer. This choice is justified by the need to test our algorithms under "orthogonal" experimental conditions. We know that the distribution of data can impact the quality of the results.

4.1.1. Low-cost machines

The low-cost machines we use in the experiments are:

- Raspberry Pi Zero W Rev 1.1: 1GHz single-core CPU, 512MB RAM, 16GB of micro SD. The architecture is based on ARMv6-compatible processor rev 7 (v6l) which is a 32bits architecture; Hardware: Broadcom BCM2835. The mosquitto server is an MQTT v3.1.1 broker;
- Raspberry Pi 4: high-performance 64-bit quad-core processor at 1.5Ghz; 8GB of RAM; 16 GB of micro SD; The architecture is based on ARMv7 Processor rev 3 (v7l); Hardware: Broadcom BCM2711. The mosquitto server is an MQTT v3.1.1 broker (version 1.5.7).
- Asus Sonicmaster Notebook with Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz; 4 cores; 4GB of memory; 368GB of disk. The board is dated from Q2 of 2012 and is running Linux version 5.11.0-18-generic. The mosquitto server is an MQTT v5.0/v3.1.1/v3.1 broker (version 2.0.10).

4.1.2. Performance of low-cost machines

For all experiments, we decided to run our implementation and the MQTT server on the same machine. We noticed that the publication of data (one by one) through the `mosquitto_pub` command line introduced a significant overhead. On the Raspberry Pi Zero W, firing no more than 256 2D points lasted about 12s on average. On a Raspberry Pi 4, firing no more than 32248 2D points gave the information depicted on Figure 3 regarding resource usage.

We notice in Figure 3 that the testing has lasted more than 96 minutes, the execution spent more than 53 minutes in kernel space and only 12 minutes in userspace. These figures imply that the transfer rate to the MQTT server cannot be significant on the type of hardware considered. We are therefore not dealing with a deluge of information per second.

Consequently, in all the experiments, we set $W = 256$ or $W = 2048$ and $p = \lfloor \sqrt{W} \rfloor$ to get experiments that last no more than 90 minutes according to the production of 12256 or 37248 2D points. We justify these choices as follows.

First, notice that the strategy to get performance is to overlap data transmission with computation. We mean that, in the permanent regime corresponding to the loop in Algorithm 1, we need to overlap step 6 of Algorithm 1 (Read p new data from the data stream to form a new window of size W) with steps 3, 4 and 5.

On one side, and according to the last numbers, the transmission rate on a Raspberry pi zero is approximatively given by $32248/3180 = 10$ 2D points per second. This number is deficient, as pointed out, and it constitutes the limiting factor for performance.

On the other side, it is known that the time complexity for steps 3, 4 and 5 is in $\mathcal{O}(nkdi)$ where n is the number of d-dimensional vectors (to be clustered), k the number of clusters, i the number of iterations needed until convergence of clustering, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n)$ respectively. For the experimental work $k = 5, d = 2$, and we have measured, for instance, a value of i, which is equal to 5 most of the time for our data. Let $T(p)$ be the time to deliver p 2D points to our algorithm with a windowing of size W, then the fundamental relation which depicts the overlapping is approximated as with Equation 1.

$$T(p) < 50.W_p + W_p \log W_p + W_p \tag{1}$$

The relation expresses that we want the data reception times to be shorter than the calculation times (clustering on the window of size W). In this way, we can overlap communications with computations. At step n of the algorithm, we compute data received during iteration $n-1$, and at the same time, we receive the data for the following iteration. The terms in W express the (approximate) complexities of the three steps of the algorithm.

Equation 1 serves to guess the W parameter of our algorithm, given a certain value for p. Operationally, you can monitor the transmission time, as we did with the script in Figure 3. This method gives a unit number, in seconds, for example, and then you can check that the right-hand expression of the equation, for a well-chosen W and after monitoring the code, passes above the value of the expression on the left.

Then we can practically say that with $W = 256$ or $W = 2048$, the time required for steps 3, 4, and 5 is far below the value of $T(p)$ for our chosen low-cost machine. Remind that $p = \lfloor \sqrt{W} \rfloor$. We could even increase the value of W for two reasons. First, we have enough memory on our boards for that purpose. Second, the time to deliver p data to our application, with a rate of only ten 2D points per second, authorizes to overlap more data, depending on the clock cycle of the board we utilize.

However, the performance metric we study in the paper is more related to the quality of the result rather than to the speed we obtain the result. Moreover, the quantitative analysis made in this section helps the expert to decide a value for W and p.

4.1.3. Heterogeneous devices and compilation toolchain

In this paper, we experiment with the reconfigurable computing class of heterogeneous computing systems, especially the Field-programmable gate array (FPGA) class. A field-programmable gate array is an integrated circuit designed to be configured by a customer or a designer after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL). The most common HDLs are VHDL and Verilog, as well as extensions such as SystemVerilog. Many tools, as those commented below, have a C-like syntax or better, accept as inputs C/C++ codes, unlike VHDL.

In [Cérin et al. (2017), and Cérin et al.] we did experiments with the low-cost Xilinx Zynq-Z2[3] board to explore thehttps://fr.overleaf.com/project/61fcfcde1d0312a1d9ce9714 optimization potential of machine learning kernels. We studied the k-means Lloyd's algorithm [Forgy (1965)], which is not a data stream algorithm and written in Python, that systematically calls the FPGA from the central processing unit to compute the distance between two points. More ambitiously, in this paper, we explore the cost, in terms of the number of gates and other resources for executing our datastream clustering algorithm fully inside an FPGA circuit.

From a technical point of view, the objective is to install an MQTT server under the control of the CPU of the Zynq board (see Figure 4), to publish or pipeline the stream from the DRAM to the internal memory of the FPGA part (the board has 630 KB of fast block RAM), then to compute the clustering on a custom block of programmable logic.

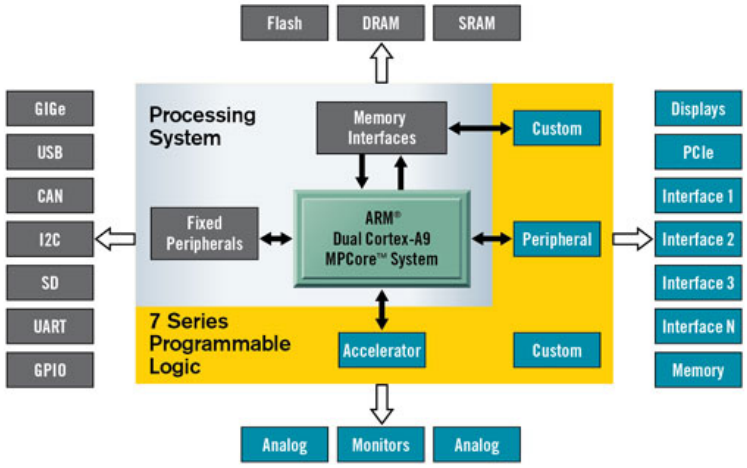


Figure 4: A functional view of a Xilinx Zynq All Programmable System on a Chip.

The three synthesis system that we use is Xilinx Vivado HLS[4], Intel/Altera Quartus[5] and Bambu[6]. Vivado introduces high-level synthesis with a toolchain that converts C/C++ code into programmable logic. Bambu is also able to synthesis C/C++ code. Quartus is used with the outputs (.vhd files) that result from the synthesis of the C/C++ code by Vivado HLS. We have chosen these three tools because they offer complementary perspectives on synthesis at the cost of writing the algorithms in C/C++.

4.2. Experimental testing

4.2.1. Metrics of performance

The total inertia I is measured as the sum of the squares of the distances of the points from the center of gravity. In the case where all the points are not assigned the same weight, the squares of the distances are, of course, weighted. If we note d the distance of a point from the

[3]See: https://www.tul.com.tw/productspynq-z2.html
[4]https://www.xilinx.com/products/design-tools/vivado.html
[5]https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html
[6]https://panda.dei.polimi.it/?page_id=31

center of gravity of the cloud, we have for n units assigned a weight p, the relation of Equation 2 for the definition of I.

$$I = \sum_{i=1}^{n} p_i.d_i^2 \tag{2}$$

It follows that the more points we add, the more inertia increases. If all the observations are assigned the same weight, the inertia is confused with a dispersion indicator. In our case, the final result of clustering is only of size $w \ll n$, n being the number of 2D points we have covered. Moreover, inertia assumes that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters or manifolds with irregular shapes. At last, inertia is not a normalized metric: we know that lower values are better and zero is optimal.

In this paper, we do prefer to use the scattering metric as introduced in [Gauvrit and Delahaye (2006)]. The metric refers to the order r diameter, and it is defined as with Equation 3 for n points.

$$D_r = \left(\frac{1}{n(n-1)} \sum_{i \neq j}^{n} d_{i,j}^r \right)^{1/r} \tag{3}$$

In the general case, specific indices are recognized. Thus, D_2 is the Greenwood index. The limit for r tending to infinity, D_∞ is the "diameter" in the usual sense of mathematics of graphs. Finally, D_1 is called the Gini index. This arithmetic mean of the distances is one of the most frequent spreading measures by far for the treatment of two-dimensional data, as it is reminded in [Gauvrit and Delahaye (2006)].

So, in the paper, we consider the Gini index over the cloud of centroids. In this way, we can compare two sets of centroids, eventually obtained by two different methods and even if we do not have access to all the data covered during the clustering process.

Moreover, the Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient is defined as the size of the intersection divided by the size of the union of the sample sets, as with Equation 4.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4}$$

The Jaccard distance, which measures dissimilarity between the sample sets is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union. The Jaccard distance equation is Equation 5.

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{5}$$

In our context, we use the Jaccard index to evaluate the similarity (or dissimilarity) between two sets of centroids.

4.2.2. On low-cost machines

This section selected what we consider the most exciting points underlying our design. First, we have chosen a non-favorable case in playing with the nature of the input data. We also

measured the dispersion of the centroids, and the results are given in Tables 1 and 2. Second, we made a series of experiments with a more mild case than the previous one with a random data generation for both two dimensions. Table 3 synthesized the main results regarding the dispersion of centroids. The third series of experiments consider even a more favorable case in explicitly generating data spread in five clusters. Table 4 gives the results for the considered performance metrics. The fourth series of experiments consider a real data distribution from the Smart Building sector. We have timestamps and CO_2 measures related to the building in the data set. Again, since we have time series, we are in the presence of potentially non-trivial clusters.

Experiment 1. The first set of experiments is not favorable to our algorithm since-https://fr.overleaf.com/project/61fcfcde1d0312a1d9ce9714 we strictly increase time-stamped data for one dimension while the other dimension takes random values. Indeed in our case, we will calculate the centroids from the data of the last window and history. In contrast, by knowing the totality of the data in advance, we manage less uncertainty.

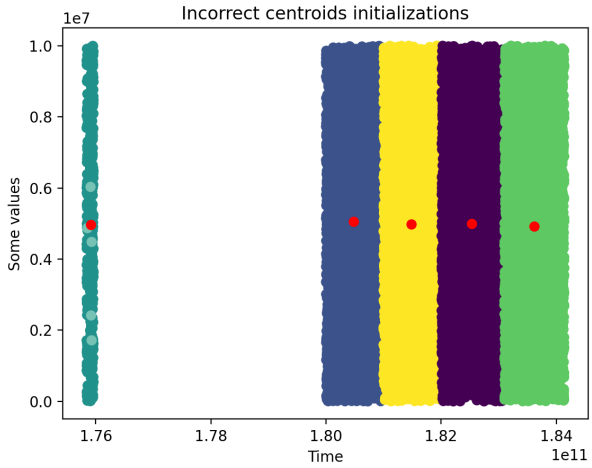


Figure 5: Illustration of the scattering.

Figure 5 illustrates the problem of dispersion of centroids with our settings. This picture corresponds to the results on line 1 of Table 1. Here, the five circles in the left part of the image and inside the left block are our initialization from the first 256 values read. This initialization sets the centroids forever. Since the values on the x-axis are time-stamped data, this initialization is wrong since we do not take into account the temporal evolution of the data. The horizontal circles are the centroids computed by Scikit learn's Kmeans, i.e., over the whole data. The centroids are much better than with our data stream. It is, therefore, preferable to initialize the centroids at each iteration of our algorithm.

Thus, we compare the obtained centroids with our algorithm and with the Scikit-learn kmeans algorithms, assuming that in this case, we know all the data in advance.

Table 1 presents the results when we do not re-initialize the centroids at each iteration of our algorithm, and Table 2 introduces the results when we reinitialize the centroids at each iteration

```
#!/bin/bash
name=‘date +%d%m%Y%H%M%N.csv‘
for i in 1..37248; do
    foo=‘date +%H%M%S%N‘;
    i=‘expr $foo / 1000‘;
    j=‘shuf -i 1-10000000 -n 1‘;
    # publish data
    /usr/bin/mosquitto_pub -h localhost -t date/celsius -m "$i,$j"
    echo "$i,$j" ≫ ${name}
done
# publish the signal to terminate
/usr/bin/mosquitto_pub -h  localhost -t date/celsius -m "0.0,0.0"
# publish the file name of generated data
/usr/bin/mosquitto_pub -h localhost -t final/final -m "$name,$name"
```

Figure 6: How-to start the production of data

of our algorithm. The test checks the importance of the re-initialization step in our algorithm. The main observation is the ratio of Gini indexes. For Table 1, the ratios vary from 42.5% to 99.31%, meaning that the clustering obtained with our data stream algorithm and in using the Scikit-learn k-means algorithm are very different. On the contrary, the ratios of Gini indexes for Table 2 vary from 26.32% and 72.55% with many values close to the ideal 0%. This quantitative result, meaning that we control dispersion, demonstrates the importance of reinitializing the centroids at each iteration step of our algorithm.

Summarizing, Tables 1 and 2 present experiments with $W = 256$, the number of 2D points is set to 12256, and the number of clusters is $k = 5$. Our data stream algorithm is configured with kmeans as the offline algorithm. The low-cost machine used in these experiments is a Raspberry Pi 4, executing the MQTT server and our algorithm. The publication of data to the MQTT server is done according to the Bash script shown on Figure 6.

We notice in Tables 1 and 2 that the Jaccard indexes are always 0, meaning that we have never found common centroids. This result is not dramatic in itself. It simply shows that the Jaccard index is not an exciting metric for our problem. We could not infer this at the beginning of our study.

Finally, note that in the tables, we give the centroids obtained through QR codes rather than listing, in extension, the multiple values. This presentation reduces the size of the tables.

Experiment 2. The second series of experiments is more favorable than the previous one. First, we do not use time-stamped data but data generated at random for both the x and the y dimensions. Moreover, we now configure our clustering algorithm with kmeans++ (and no more kmeans), and we compare our results with kmeans++ from Scikit learn. This last option means that the clustering is accomplished knowing all the data. Table 3 shows the results with $W = 2048$ and the number of 2D points set to 37248. By experience, we know that kmeans++ does a better initialization than k-means. Second, the intuition is that the more we have data in the window, the more accurate the clustering will be, hence our choice to increase the W value. These ideas motivate the experiment for Table 3. We observe that the ratio of Gini indexes is much better than the ones in the first series of experiments. The ratios of Gini indexes vary only between -0.48% and 5.71%, which is excellent regarding the dispersion. The practical

Table 1: Quality metrics with no recalculation of centroids between iterations.

W	n	k	**Centroids with data stream algorithm**	**Centroids with Scikit learn kmeans**	**Jaccard index**	**Gini index for data stream algorithm**	**Gini index for Scikit learn kmeans**	**Ratio Gini indexes (%)**
256	12256	5			0	18165972.03	1746156257.56	-98.95
256	12256	5			0	11216278.13	1647259205.64	-99.31
256	12256	5			0	19471853.63	1696275260.49	-98.85
256	12256	5			0	8301303.31	878905310.71	-99.05
256	12256	5			0	17447215.28	1680150040.83	-98.96
256	12256	5			0	15332205.36	863360772.02	-98.22
256	12256	5			0	21623806.32	2429909814.65	-99.11
256	12256	5			0	19259137.69	864481821.94	-97.77
256	12256	5			0	1248623980.75	876209679.72	42.50

Table 2: Quality metrics with recalculation of centroids between iterations.

W	n	k	Centroids with data-stream algorithm	Centroids with Scikit learn kmeans	Jaccard index	Gini index for data stream algorithm	Gini index for Scikit learn kmeans++	Ratio Gini indexes (%)
256	12256	5			0	3122037561.71	3158922384.24	-1.16
256	12256	5			0	1463506031.54	1478086649.88	-0.98
256	12256	5			0	1286810869.93	4688680283.72	-72.55
256	12256	5			0	1468338014.16	1478659259.98	-0.69
256	12256	5			0	47160900799.98	47726023229.72	-1.18
256	12256	5			0	4540029344.29	3593890145.82	26.32
256	12256	5			0	1466775924.92	1858705978.39	-21.08
256	12256	5			0	3712835175.95	3566299932.92	4.10
256	12256	5			0	657737648.42	652562726.60	0.79
256	12256	5			0	3041363584.56	3160196060.50	-3.76

choice of the value of W and the use of kmeans++ on data that are no longer time-stamped is undoubtedly the cause of these good ratios.

As another illustration of the results for data generated at random for the x and y dimensions, Figure 7 corresponds to the sample on line 7 of Table 3. The crosses materialize the centroids obtained by our algorithm and the circles obtained by kmeans++. The markers seem to be confused, as expected from the results in Table 3.

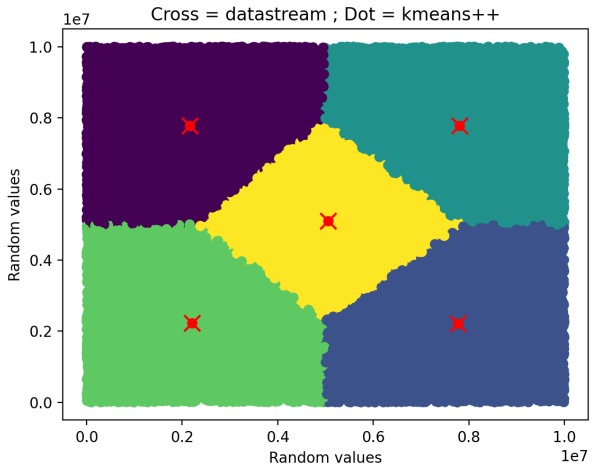


Figure 7: Illustration of the scattering.

Experiment 3. The third set of experiments is more favorable because we search for 5 clusters in data that contain 5 clusters, and with $k = 5$ as the parameter of kmeans++. An example of an input image to our algorithm is given in Figure 8. We notice that the delineation of clusters is clear. Thus, the configuration and the settings leading to the results presented in Table 4 are the following. We set up a process to obtain the five clusters based on randomly generating points inside five circles of radius 10000. Thus, each test set we present is unique, meaning that it contains different 2D points. Each test set is composed of 37248 2D points.

Moreover, since the value $W = 2048$ gave good results in the second set of experiments, and this value is reasonable for a low-cost machine, we use it again. The sensitivity of W and p have been discussed empirically in the first two experimental parts. Of course, these parameters affect the quality of the clustering result. But the nature of the data is also essential. It seems to us that we have previously given valuable information to the readers.

Table 4 shows the results over 8 test sets. We notice that the ratios of the Gini indexes are excellent because no value is greater than 0.789%. So, the scattering between the centroids obtained by our algorithm and the kmeans++ algorithm, i.e., when we know all the data in advance, is very poor. Our algorithm provides centroids similar to those we can obtain if one has used the offline kmeans. In other words, the uncertainty on data has been reduced drastically.

The experiments in Table 4 were performed on the Asus notebook in the characteristics given earlier in the text. Each experiment performs 801 iterations of the algorithm 1 for an average cost of 0.0023 seconds. Recall that an iteration consists of launching a clustering, sorting the

Table 3: Quality metrics with recalculation of centroids between iterations, using kmeans++ and random data for both dimensions.

W	n	k	Centroids with data stream algorithm	Centroids with Scikit learn kmeans	Jaccard index	Gini index for data stream algorithm	Gini index for Scikit learn kmeans++	Ratio Gini indexes (%)
2048	37248	5			0	2815391.61	2693971.93	4.50
2048	37248	5			0	2794727.55	2679762.32	4.29
2048	37248	5			0	2848730.58	2700650.29	5.48
2048	37248	5			0	2793567.11	2687583.25	3.94
2048	37248	5			0	2751330.78	2764695.72	-0.48
2048	37248	5			0	2715312.63	2690239.39	0.93
2048	37248	5			0	2850149.24	2695244.71	5.74
2048	37248	5			0	2838615.15	2700738.53	5.10

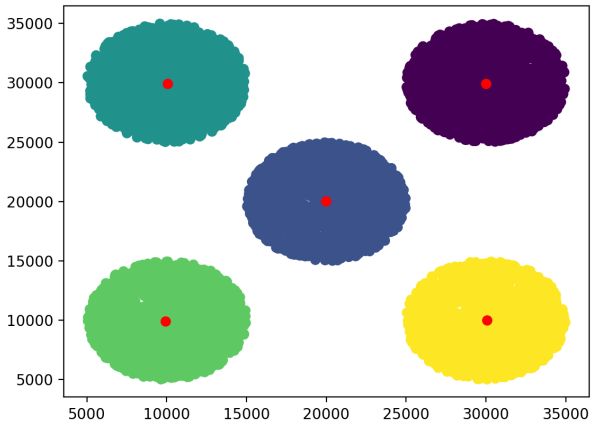


Figure 8: A favorable use case with five clusters.

result, and eliminating some data. These 0.0023 seconds are much lower and by far than the time spent by the system to deliver the data to our algorithm. Thus our algorithm, under the assumptions of this series of experiments, can process data of arbitrary size at a constant time cost and much less than the time it takes to supply the data from the sensors to our algorithm. Given these results, it does not seem helpful to us to study the impact of W on the quality and the time to obtain results in a more advanced way. In the absolute, since feeding data is the bottleneck, it would be nice to evaluate cases where data is fed faster from another machine. We did experiments on another low-cost device (an Asus Notebook from 2012 but with SpecInt better than the Raspberry), and we have observed the same trends as those presented above.

Table 4: Quality metrics, using kmeans++ and data with five clusters.

W	n	k	**Centroids with data stream algorithm**	**Centroids with Scikit learn kmeans**	**Jaccard index**	**Gini index for data stream algorithm**	**Gini index for Scikit learn kmeans++**	**Ratio Gini indexes (%)**
2048	37248	5			0	9742.04	9672.15	0.722
2048	37248	5			0	9759.56	9684.33	0.776
2048	37248	5			0	9687.30	9646.29	0.425
2048	37248	5			0	9682.17	9657.60	0.254
2048	37248	5			0	9666.88	9657.67	0.095
2048	37248	5			0	9724.13	9647.98	0.789
2048	37248	5			0	9651.84	9661.13	-0.096
2048	37248	5			0	9689.98	9668.39	0.223

Experiment 4. The data set we deal with in this experiment comes from a building located in France called the GreEn-ER building, located in the Presqu'ile of Grenoble, Isère, France. The building gathers the Grenoble-INP engineering school Ense3, the G2Elab laboratory, and training and research platforms. At the moment, the data set is not publicly available for privacy concerns, but anyone can use a public API[7] to extract data and build his own data set. The data

[7]http://mhi-srv.g2elab.grenoble-inp.fr/API/

set comprises timestamps plus CO_2 information (minimal observed values across the building). The date range for the data is from Dec 16, 2021, to Jan 7, 2022.

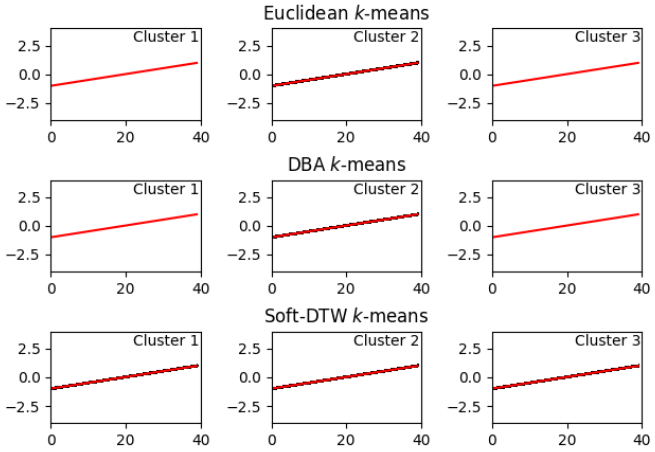
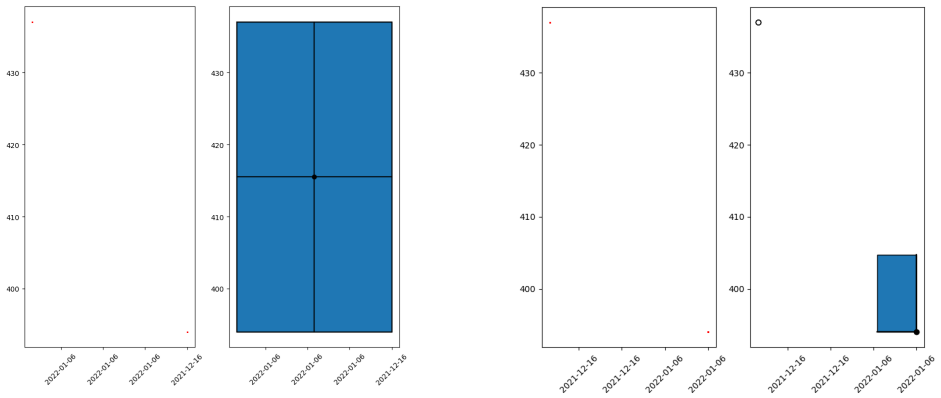


Figure 9: Characterization of the CO_2 data set.

Figure 9 presents a characterization of the CO_2 data set in terms of clustering, through the `TimeSeriesKMeans` method from the `tslearn` Python package. This method is dedicated to clustering for time series. Figure 9 has three rows; each row corresponds to a specific distance, among three distance metrics. Figure 9 has three columns; each column corresponds to a cluster because we asked to compute the clustering for three clusters. Indeed we called first the `TimeSeriesScalerMeanVariance(mu=0.0, std=1.0)` class that scales time series so that their mean (resp. standard deviation) in each dimension is mu (resp. std). This scaler is such that each output time series has zero mean and unit variance. The assumption here is that the range of a given time series is uninformative and one only wants to compare shapes in an amplitude-invariant manner. Second, we called the `TimeSeriesResampler(sz)` class that resamples time series to reach the target size (sz). In our case, we set $sz = 40$. Third, we fit k-means clustering using our CO_2 data set and then predict the closest cluster each time series in the data set belongs to. Thus, Figure 9 plots the regression between the prediction (in red) and the calculated values (in black). The x-axis gives the 40 (re)samples id for the initial time values, and the y-axis gives the resampling values for the CO_2. We observe that some clusters have no black prediction because the metric used was not discriminant enough to exhibit three clusters on the resampled data set. Anyway, we observe an excellent fit between the predictions and the observations, meaning that the values of CO_2 are not spread significantly. The linearity in the curves explains this conclusion.

In the Figure 10, for space reasons, we have given the centroids for a window of size $W = 2048$ after running our data-stream algorithm ten times, and with $k = 3$. Since there is a random initialization, we can expect different runs. However, the numerical values for centroid 2 are the 4 points $(1639661568000, 437)$ and the 4 points $(1641500377088, 394)$, while for centroid 3 we got the 5 points $(1639658160128, 438)$ and the 2 points $(1641544548352, 412)$. Some executions having returned $(0, 0)$ we have eliminated them. This fact is due, as explained before, to the distance calculation, which is not discriminating enough to show 3 clusters. In



(a) Centroid 2 (b) Centroid 3

Figure 10: Centroids obtained by our algorithm with 10 runs (W=2048)

the Figure 10, the x-axis gives the time, and the y-axis gives the CO_2 level. The shapes in the Figure 10 are different because in one case, we have an equal number of distinct values, and in the other case, we have an unstable situation (5 and 2 identical points), hence our intuition to represent it as a boxplot.

We can conclude that our algorithm is robust since it returns less distinct values for the centroids than Experiment 1, which also concerns timestamps, but the values were chosen at random. Since the CO_2 data are less sparse than for Experiment 1, we better control the production of centroids. These assertions are stil valid for $W = 128, 256, 512, 1024$.

4.2.3. On heterogeneous computing devices

Our first series of experiments is related to sorting since sorting is part of our main algorithm. Since synthesis tools are not able to compile recursive functions; we have not used the traditional Quicksort or 3-way Quicksort but a variant with no recursion, written in C++ over 2D points. Table 5 presents the metrics we obtained with the Bambu compiler and for two devices, namely Zynq and Cyclone V. The number of 2D points equals $32768 = W$.

Table 5: Synthesis metrics for the function main of the sorting procedure

W	Device	Register allocation	Total estimated area	Estimated number of DSP	Number of allocated multiplexers
32768	Zynq	101	19436	6	92
32768	Cyclone V	109	24178	4	90

The meaning attached to metrics is clear, except for the area metric. It is about the number of allocated registers for the device by the compiler, the estimated number of digital signal processing units assigned to the design, and the number of allocated multiplexers, the basic hardware units. According to [Nane et al. (2016)], to evaluate the area of the allocated circuit,

authors consider logic, DSP, and memory usage. For logic area, in Xilinx devices such as the Zynq and Cyclone V boards, Bambu reports the total number of fracturable 6-LUTs (Look Up Table: an array that replaces runtime computation with a simpler array indexing operation.), each of which can be used to implement any single function of up to six variables or any two functions that together use at most five distinct variables. Note that the Zynq circuit has only 13,300 logic slices, each with four 6-input LUTs.

The methodology we follow in the subsection measures the metrics mentioned above for our design or part of the design. We also used different compilers/tools to cross-validate the results or check our assumptions' scope. Notice that we are no more interested in the properties of algorithm 1, but rather by the properties of the design regarding the quantity of hardware component, from a computer architecture point of view, necessary to execute the algorithm on the FPGA class of hardware architecture.

Summarizing the results presented in Table 5 we observe that the area is 20% more costly for the Cyclone V circuit compared to the Zynq circuit. The difference for the other metrics is less significant. This observation is due to the more aggressive synthesis for Zynq rather than for the Cyclone V circuit.

Table 6 shows the performance metrics for the same sorting code and in using Vivado HLS. The target device is the Zynq board and W has been set to 64. The "Total" line gives the number of resources consumed by the synthesis, and the "Available" line shows the amount available for that resource on the board. The RAM blocks (BRAM) are used to store large amounts of data inside the FPGA. The rate of use of the other resources, notably DSP (Digital Signal Processor), FF (FIFO), LUT (Lookup Table), and URAM (Ultra RAM) are systematically under the bar of what is available. This observation allows us to conclude that we could increase the value of W for the Zynq circuit.

Table 6: Vivado performance metrics of the sorting procedure

Name	BRAM_18k	DSP48E	FF	LTU	URAM
DSP	-	-	-	-	-
Expression	-	4	0	144	-
FIFO	-	-	-	-	-
Instance	4	0	1908	3309	0
Memory	2	-	2	1	0
Multiplexer	-	-	-	257	-
Register	-	-	252	-	-
Total	6	4	2162	3711	0
Available	280	220	106400	53200	0
Utilization	2	1	2	6	0

Figure 11 shows the performance metrics of the circuit when the input of the Quartus tool is given by the VHDL code obtained with the Vivado HLS tool. We receive other performance metrics than the previous ones, among them the Fan-out. Fan-out coefficient refers to the number of doors of the same type driven by the output end of a door or load capacity. The fan-out coefficient reflects the load capacity of the gate circuit, and it is a quality metric that we could use to distinguish different syntheses.

Our second set of experiments consists of synthesizing the code corresponding to steps 3, 4,

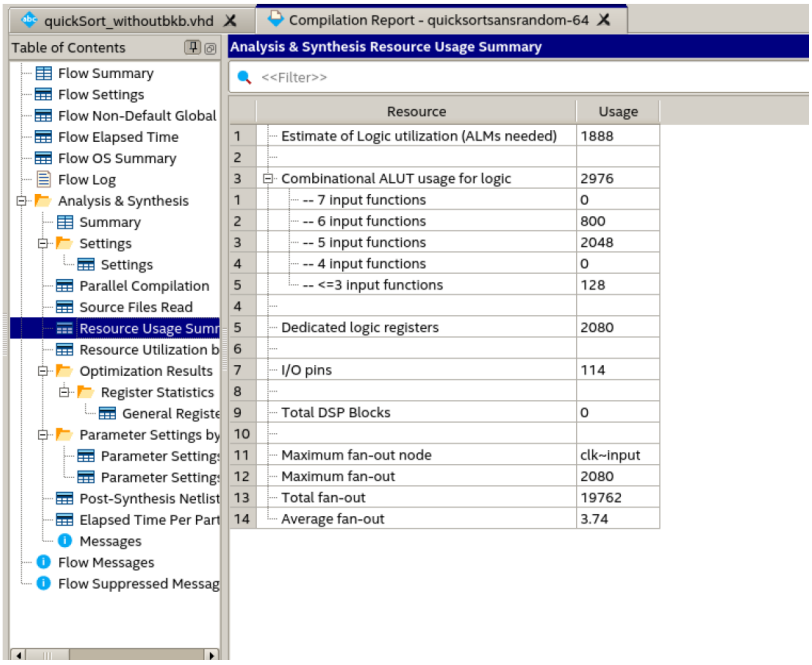


Figure 11: Quartus performance metrics of the sorting algorithm when we use the VHDL obtained by Vivado as input.

5, and 6 of the algorithm 1.

4.3. Lessons learned from the experiments

The scattering and dissimilarity measure points of view showed that obtaining centroids can be controlled through our data stream algorithm settings. We also highlighted that our algorithm had limitations when the data was time-stamped. Using algorithms other than kmeans or kmeans++ as an offline brick could solve this nature of data problems. We think about spectral clustering algorithms. Another critical point concerns the overlapping of communication (delivery of data from the sensors to the application) and computation (clustering). We observed a limited throughput of communication on low-cost machines. So we can multiply the number of iterations of our algorithm, meaning that we can frequently call kmeans/kmeans++ on W data without penalty.

From a reconfigurable computing point of view, we proposed to look at three specific tools, each measuring different performance metrics. Of course, each of these metrics makes sense and provides insights for the expert to estimate the quality and performance of the synthesized circuit. On this plan, the general question is, "What is the best compromise on the metrics to obtain a maximum W value, given a reconfigurable circuit?". This question opens new directions for our work.

5. Conclusion

In this paper, we proposed a generic framework for data stream clustering on low-cost machines. Our context is the Internet of Things and Edge computing contexts. Nowadays, there

is an emphasis on creating better software and machine learning algorithms that can run efficiently on resource-constrained devices. Multiple projects to address machine learning problems on low-cost hardware have emerged recently. These projects, such as TinyML[8] for Arduino, are driven by the idea of processing on embedded hardware. In the embedded field, we can mention MCU boards (Wio Terminal, STM32 F7 F4 L4 Lo, for example), which have a lot of RAM, CPU performance, and FPU performance, allowing AI processing. For example, the RAM of the Wio is 192kB, part of which is reserved for storing the program.

We instantiated the proposed framework with k-means or k-means++ for the local search and Quicksort for the sorting step. We also instantiated the removing step at the end of one iteration with a regular sampling technique, but other methods are also possible with no sorting step. Our framework requires two parameters. The first one is the window size W, and the second one is the number of points to remove. This fact constitutes the originality of our work, and it makes a distinction from other works.

We realized experiments under a realistic workbench composed of an MQTT server and on various hardware. The datasets we used consider 2D points are generated randomly or taken from a practical situation of CO_2 measures. Two datasets have timestamps for the first component of the 2D points. The goal of the experiments is to obtain some guidelines about the best circumstances to apply this algorithm and the maximum gain in computational time without compromising the overall quality of the partition.

We showed that, due to the low-end hardware, the rate of transmitting data to the MQTT server, located on low-end hardware, can not be high. The throughput for receiving data is limited because the processor clock speed is limited. We calibrate our algorithm consequently in considering window sizes of 256 or 2048 2D points. The idea is to overlap the reception of the data stream with the computational task. We found that we had time to execute the k-means or k-means++ algorithms as well as sorting. The interest in choosing well-known algorithms is the potential for parallelism or multithreading they have. Thus, our codes are ready, thanks to OpenMP, to be executed on hardware with multiple cores such as a Raspberry Pi 4.

According to our setting, we also showed that we control the quality of the obtained centroids. For that requirement, we have used the Gini index for the dispersion of centroids and the Jaccard index for measuring the similarity between centroids.

We propose to address the following critical challenges of distributed edge AI for the future. All of them are centered on the robustness issues and detailed as follows.

We want to test our framework with, in the middle, others clustering algorithms. For instance, Mini batch k-means has the main advantage of reducing the computational cost of finding a partition. This cost is proportional to the size of the sample batch used, and this difference is more evident when the number of clusters is larger.

We want to test our framework with more than two dimensions for the sample sets. Inconsistent conclusions could be drawn regarding the influence of the number of attributes on the quality of measures.

Given these objectives, it is worth exploring other strategies related to the selection points to remove, i.e., how the window's updating is computed during the iterations.

The question of scaling and dimensioning the proposed architecture (see Figure 2) is important to address large-scale artifacts such as smart buildings and even smart cities. But in this case, we also need to examine the appropriate communication technology that could be radio

[8] https://www.tinyml.org/

communication. So the question needs to be examined under the angle of both the algorithmic part and the technology side.

At least, we want to go further with the FPGA implementation of our framework.

Acknowledgements:

We would like to thank Rêve de Scènes Urbaines (RSU) in supporting this work. RSU is an Industrial Demonstrator for Sustainable Cities project, initiated in 2016 by the French Ministries of Ecology and Housing. Like the Sorbonne Paris Nord university, RSU is located on the territory of Plaine Commune, north of Paris. This work was partially realized during Christophe Cérin's CNRS delegation at the University of Grenoble Alpes (UGA) and the Grenoble Computer Science Laboratory (LIGLAB) in the DATAMOVE-INRIA team. Thus, the work has been partially supported by the research programme on edge intelligence at the Multi-disciplinary Institute on Artificial Intelligence MIAI at Grenoble Alpes (ANR-19-P3IA-0003). Datasets and C++ codes are publicly available from `https://github.com/christophe-cerin/mosquitto-clustering`.

References

Aggarwal, C.C., Han, J., Wang, J., Yu, P.S., 2003. A framework for clustering evolving data streams, in: Freytag, J.C., Lockemann, P.C., Abiteboul, S., Carey, M.J., Selinger, P.G., Heuer, A. (Eds.), Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003, Morgan Kaufmann. pp. 81–92. URL: http://www.vldb.org/conf/2003/papers/S04P02.pdf, doi:10.1016/B978-012722442-8/50016-1.

Arthur, D., Vassilvitskii, S., 2007. K-means++: The advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, USA. p. 1027–1035.

Attaoui, M.O., Azzag, H., Lebbah, M., Keskes, N., 2020. Multi-objective data stream clustering, in: Coello, C.A.C. (Ed.), GECCO '20: Genetic and Evolutionary Computation Conference, Companion Volume, Cancún, Mexico, July 8-12, 2020, ACM. pp. 113–114. URL: https://doi.org/10.1145/3377929.3389930, doi:10.1145/3377929.3389930.

Attaoui, M.O., Azzag, H., Lebbah, M., Keskes, N., 2021. Subspace data stream clustering with global and local weighting models. Neural Comput. Appl. 33, 3691–3712. URL: https://doi.org/10.1007/s00521-020-05184-z, doi:10.1007/s00521-020-05184-z.

Béjar Alonso, J., 2013. K-means vs Mini Batch Kmeans: A comparison. Technical Report. Universitat Politècnica de Catalunya.

Bifet, A., Holmes, G., Pfahringer, B., Read, J., Kranen, P., Kremer, H., Jansen, T., Seidl, T., 2011. MOA: A real-time analytics open source framework, in: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (Eds.), Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III, Springer. pp. 617–620. URL: https://doi.org/10.1007/978-3-642-23808-6_41, doi:10.1007/978-3-642-23808-6_41.

Carnein, M., Trautmann, H., 2019. Optimizing data stream representation: An extensive survey on stream clustering algorithms. Business & Information Systems Engineering (BISE) 61, 277–297. doi:10.1007/s12599-019-00576-5.

Cérin, C., Gaudiot, J., Lebbah, M., Yuehgoh, F., 2017. Return of experience on the mean-shift clustering for heterogeneous architecture use case, in: Nie, J., Obradovic, Z., Suzumura, T., Ghosh, R., Nambiar, R., Wang, C., Zang, H., Baeza-Yates, R., Hu, X., Kepner, J., Cuzzocrea, A., Tang, J., Toyoda, M. (Eds.), 2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017, IEEE Computer Society. pp. 3499–3507. URL: https://doi.org/10.1109/BigData.2017.8258339, doi:10.1109/BigData.2017.8258339.

Cérin, C., Jiang, C., Saad, W., Mekni, E., . Where are the optimization potential of machine learning kernels?, in: (https://conferences.computer.org/data com/2019/#!/home), C.S. (Ed.), IEEE International Symposium on Cloud and Service Computing (SC2), https://conferences.computer.org/datacom/2019/pdfs/DataCom2019-3MYdIPKpqxiurNWZaDmspf/8Zjs1VsMXV4HWbHVXzYXj/5uUilQa2p9bLtuicjXJEqZ.pdf.

Fisher, D.H., 1987. Knowledge acquisition via incremental conceptual clustering. Mach. Learn. 2, 139–172. URL: https://doi.org/10.1007/BF00114265, doi:10.1007/BF00114265.

Fisher, D.H., 1996. Iterative optimization and simplification of hierarchical clusterings. J. Artif. Intell. Res. 4, 147–178. URL: https://doi.org/10.1613/jair.276, doi:10.1613/jair.276.

Forgy, F., 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics .

Friedman, E., Tzoumas, K., 2016. Introduction to Apache Flink: Stream Processing for Real Time and Beyond. 1st ed., O'Reilly Media, Inc.

Fritzke, B., 1994. A growing neural gas network learns topologies, in: Proceedings of the 7th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA. p. 625–632.

Gauvrit, N., Delahaye, J.P., 2006. Order o diameter. a "natural" measure of scattering. Mathematics and social sciences, volume: 175, pages: 41–51. URL: http://journals.openedition.org/msh/3553, doi:10.4000/msh.3553.

Ghesmoune, M., Lebbah, M., Azzag, H., 2016a. A new growing neural gas for clustering data streams. Neural Networks 78, 36–50. URL: https://doi.org/10.1016/j.neunet.2016.02.003, doi:10.1016/j.neunet.2016.02.003.

Ghesmoune, M., Lebbah, M., Azzag, H., 2016b. State-of-the-art on clustering data streams. Big Data Analytics 1, 13. URL: https://doi.org/10.1186/s41044-016-0011-3, doi:10.1186/s41044-016-0011-3.

Howes, L., Hower, D., Gaster, B., 2016. Chapter 5 - hsa memory model, in: mei W. Hwu, W. (Ed.), Heterogeneous System Architecture. Morgan Kaufmann, Boston, pp. 53–75. URL: https://www.sciencedirect.com/science/article/pii/B9780128003862000043, doi:https://doi.org/10.1016/B978-0-12-800386-2.00004-3.

Hwu, W.M.W., 2016. Chapter 1 - introduction, in: mei W. Hwu, W. (Ed.), Heterogeneous System Architecture. Morgan Kaufmann, Boston, pp. 1–5. URL: https://www.sciencedirect.com/science/article/pii/B9780128003862000092, doi:https://doi.org/10.1016/B978-0-12-800386-2.00009-2.

Lloyd, S.P., 1982. Least squares quantization in PCM. IEEE Trans. Inf. Theory 28, 129–136. URL: https://doi.org/10.1109/TIT.1982.1056489, doi:10.1109/TIT.1982.1056489.

Martinetz, T.M., Schulten, K.J., 1991. A "neural gas" network learns topologies, in: Kohonen, T., Mäkisara, K., Simula, O., Kangas, J. (Eds.), Proceedings of the International Conference on Artificial Neural Networks 1991 (Espoo, Finland), Amsterdam; New York: North-Holland. pp. 397–402.

Nane, R., Sima, V.M., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y.T., Hsiao, H., Brown, S.D., Ferrandi, F., Anderson, J.H., Bertels, K., 2016. A survey and evaluation of FPGA high-level synthesis tools. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 35, 1591–1604. URL: https://doi.org/10.1109/TCAD.2015.2513673, doi:10.1109/TCAD.2015.2513673.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Sculley, D., 2010a. Web-scale k-means clustering, in: Proceedings of the 19th International Conference on World Wide Web, Association for Computing Machinery, New York, NY, USA. p. 1177–1178. URL: https://www.eecs.tufts.edu/$\sim$dsculley/papers/fastkmeans.pdf, doi:10.1145/1772690.1772862.

Sculley, D., 2010b. Web-scale k-means clustering, in: Rappa, M., Jones, P., Freire, J., Chakrabarti, S. (Eds.), Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010, ACM. pp. 1177–1178. URL: https://doi.org/10.1145/1772690.1772862, doi:10.1145/1772690.1772862.

Terzo, O., Djemame, K., Scionti, A., Pezuela, C., 2019. Heterogeneous Computing Architectures: Challenges and Vision (1st ed.). CRC Press.

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I., 2013. Discretized streams: Fault-tolerant streaming computation at scale, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, Association for Computing Machinery, New York, NY, USA. p. 423–438. URL: https://doi.org/10.1145/2517349.2522737, doi:10.1145/2517349.2522737.

Zahran, M., 2019. Heterogeneous Computing: Hardware and Software Perspectives. Association for Computing Machinery, New York, NY, USA.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. Birch: An efficient data clustering method for very large databases. SIGMOD Rec. 25, 103–114. URL: https://doi.org/10.1145/235968.233324, doi:10.1145/235968.233324.

Table for notations used in the paper

Notation	Meaning
M	Memory size available on the computing board
n	Number of memory partitions we consider
l	Subsets such that each subset fits in memory
k	Number of desired centroids or clusters
W	Window size
c	Constant factor
$\mathcal{O}(nkdi)$	Complexity time, with n being the number of d-dimensional vectors (to be clustered), k the number of clusters, i the number of iterations needed until convergence of clustering, $\mathcal{O}(n \log n)$ being the time for sequential sorting, and $\mathcal{O}(n)$ the time for a linear search.
I	The total inertia
d	The distance of a point to the gravity center of one cluster.
D_2	The Greenwood index
D_∞	The "diameter" in the usual sense of mathematics of graphs
D_1	The Gini index
$J(A, B)$	The Jaccard coefficient
$d_J(A, B)$	The Jaccard distance

Acronyms or abbreviation Table

Acronym	Definition
GAFAM	Google, Amazon, Facebook, Apple, Microsoft
BATX	Baidu, Alibaba, Tencent, Xiaomi
FPGA	Field-programmable gate array
MQTT	Message Queuing Telemetry Transport
MOA	Massive Online Analsys
CF	Clustering Feature
SMP	Shared-memory multiprocessing
OMP	OpenMP programming language
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
COBWEB	A conceptual clustering system that organizes data so as to maximize inference ability
GAG	Growing Neural Gas
NUMA	Non-uniform memory access
HDL	Hardware description language
VERILOG	A hardware description language used for modelling electronic systems; Verilog is an HDL description language based on C language, on the other hand, VHDL is also an HDL but it is based on Ada and Pascal languages.
VHDL	Very High-Speed Integrated Circuit; Hardware Description Language: a hardware description language used to describe digital and mixed-signal systems.
LUT	Lookup Table
FF	FIFO
BRAM	RAM block
DSP	Digital Signal Processor
URAM	Ultra RAM