electronics

Article

Clustering Algorithms for Enhanced Trustworthiness on High-Performance Edge-Computing Devices

Marco Lapegna [1,*], Valeria Mele [1] and Diego Romano [2]

[1] Department Mathematics and Applications, University of Naples Federico II, Via Cintia-Monte S. Angelo, 80126 Napoli, Italy
[2] Institute of High Performance Computing and Networking, National Research Council, Via P. Castellino 111, 80131 Napoli, Italy
* Correspondence: marco.lapegna@unina.it

Abstract: Trustworthiness is a critical concern in edge-computing environments as edge devices often operate in challenging conditions and are prone to failures or external attacks. Despite significant progress, many solutions remain unexplored. An effective approach to this problem is the use of clustering algorithms, which are powerful machine-learning tools that can discover correlations within vast amounts of data. In the context of edge computing, clustering algorithms have become increasingly relevant as they can be employed to improve trustworthiness by classifying edge devices based on their behaviors or detecting attack patterns from insecure domains. In this context, we develop a new hybrid clustering algorithm for computing devices that is suitable for edge computing model-based infrastructures and that can categorize nodes based on their trustworthiness. This algorithm is thoroughly assessed and compared to two computing systems equipped with high-end GPU devices with respect to performance and energy consumption. The evaluation results highlight the feasibility of designing intelligent sensor networks to make decisions at the data-collection points, thereby, enhancing the trustworthiness and preventing attacks from unauthorized sources.

Keywords: clustering algorithms; edge-computing devices; high-performance computing; trustworthiness improvement

Citation: Lapegna, M.; Mele, V.; Romano, D. Clustering Algorithms for Enhanced Trustworthiness on High-Performance Edge-Computing Devices. *Electronics* **2023**, *12*, 1689. https://doi.org/10.3390/electronics12071689

Academic Editors: Marios Avgeris, Dimitrios Dechouniotis, Konstantinos Tsitseklis and Vitoropoulou Margarita

Received: 15 February 2023
Revised: 22 March 2023
Accepted: 28 March 2023
Published: 3 April 2023

1. Introduction

The emergence of the edge-computing paradigm is a response to the limitations of centralized data management in real-time applications. Centralized data management often results in network bandwidth saturation due to the transfer of large volumes of data, thereby, leading to delays and reduced efficiency. Edge computing offers a solution to this problem by moving decision points closer to the data source at the edge of sensor networks. This approach reduces the volume of data that needs to be transmitted between sensors and central servers and, thus, the network overhead.

The benefits of edge computing are extensively documented in the literature, such as in [1–3]). The authors of the present study are also involved in this research area, having developed applications for edge computing in the fields of environmental science and smart cities (e.g., [4–7]).

Edge computing offers additional advantages in addition to the network cost savings achieved by reducing the volume of data transmitted. The distributed infrastructure of edge computing enables improved fault tolerance by isolating and repairing single network failures without impacting the overall service, thus, leading to enhanced reliability and reduced repair costs. In addition, edge computing facilitates higher democratization and accessibility to knowledge through data distribution across various locations and organizations as well as increased privacy and security through the reduced spread of sensitive information on the network.

Furthermore, edge computing enables real-time performance improvement by eliminating dependencies on central servers for time-sensitive applications. From the above, the interplay between edge computing and Artificial Intelligence (AI) and machine-learning (ML) algorithms becomes evident. Specifically, the application of such techniques to edge computing has the potential to significantly enhance the trustworthiness of the infrastructure [8].

More precisely, the heterogeneity of devices and the presence of untrustworthy or insecure radio networks make securing data and maintaining privacy challenging. In order to mitigate the described risks, it is necessary to categorize devices using appropriate classification algorithms, also referred to as *clustering algorithms*. These algorithms are frequently employed in edge-computing environments not only for application development and network management but also with special attention to reliability improvement through device monitoring and classification (e.g., [8–10]).

In this scenario, the high-performance edge computing (HPEC) model, which leverages high-performance computing units deployed on low-power sensor devices, has gained widespread acceptance. This model enables the implementation of sophisticated algorithms through multi-core CPU or GPU computing units integrated on edge-computing boards, thereby, allowing effective real-time ML and AI applications at the network edge. A significant example of a device that enables such a model is the NVIDIA Jetson Nano Board, equipped with a quad-core ARM CPU and an Nvidia GPU with 128 CUDA cores. In this regard, it is worth noting that the utilization of advanced forms of parallelism within computer architectures is a crucial strategy for enhancing performance without compromising energy efficiency [11].

The present work then introduces a clustering algorithm based on this model leveraging the high-performance computing units on the board. The aim is to evaluate whether a low-power device achieves a good performance-to-energy trade off executing a machine-learning algorithm with the ultimate goal of enhancing the reliability of the environment for example through the analysis of network traffic data and the detection of attack patterns. The manuscript is structured as follows: Section 2 describes the research field and situates our work in relation to prior literature.

In Section 3, we present an edge computing-based infrastructure and the device's technical specifications employed for the experiments. Section 4 constitutes the paper's core, in which we propose a new hybrid version of a clustering algorithm, exploring various models of parallelism that can be leveraged. Section 5 analyzes and discusses the experiments conducted on four distinct datasets, aiming to assess the algorithm from multiple perspectives. Finally, Section 6 summarizes our findings and suggests roads for future research.

2. Related Works

Several works have dealt with the interaction between edge-computing environments and security and trustworthiness issues (e.g., [8–10,12–14]). They focused on various reliability and attack classification models but give less attention to performance-related issues on low-power devices.

In any case, the performance of edge devices is considerably inferior to that of high-end systems. Consequently, even if they are equipped with high-performance computing units, such as multi-core CPUs or GPUs, specific approaches are required for deploying AI algorithms in edge-computing environments.

One approach, known as *Framework Design*, focuses on adapting an AI model to the edge environment without significant modifications through the decomposition of the learning model across edge devices coordinated by a proximity server (as seen in works such as [15,16]). Another approach, *Model Adaption*, involves modifications to the model itself to enhance its effectiveness on devices with limited resources through techniques, such as data compression and model filtering through mathematical tools (e.g., [17]).

Finally, *Process Acceleration* involves rethinking traditional ML and deep-learning algorithms to incorporate forms of parallelism, both from a hardware perspective with specialized devices for AI algorithms and from a software perspective by optimizing algorithms for parallel computing in edge-computing environments (e.g., [18,19]). These approaches aim to address the limitations of edge devices and improve the execution of AI models in edge-computing environments.

Based on the analysis of the literature reviewed, there is a clear interest in implementing ML and AI methods and algorithms in edge-computing environments. However, to the best of our knowledge, there appears to be limited attention paid to the various forms of parallelism present in these methods in order to effectively exploit multi-core CPUs or GPU-based accelerators, which are now widely available in many low-power devices. For example, issues related to a reliable trade-off between performance and power consumption appear to be poorly explored. We propose that the use of such methodologies can enhance the ability to detect malfunctions and intrusions in the environment more quickly, thereby, allowing for timely countermeasures to be implemented.

The current study is part of our research about the adoption of specifically designed devices for the edge-computing environment to implement high-performance clustering algorithms, mainly by using the Process Acceleration approach. As previously documented in [20,21], we initially presented an adaptive K-means algorithm for multi-core CPUs. The implementation of this algorithm on GPU-based devices was then evaluated in [22]. Finally, in [23], we presented the first experiences related to security concerns. In this paper, we address some aspects not fully faced in our or other authors' works. More precisely, the relevant contributions of the paper can be summarized as follows:

- The implementation of a hybrid strategy that leverages the multi-core CPU and GPU on the Nvidia Jetson Nano board concurrently, leading to a further reduction in execution time.
- The adoption of a more precise method for measuring energy consumption with a comparison to two other high-end GPU-based systems.
- A study about the possibility of using algorithms to improve the trustworthiness and security of the edge-computing environment.

3. Computing Environment

A modern distributed infrastructure can be characterized as a layered architecture, as depicted in Figure 1, called a *computing continuum* [1].

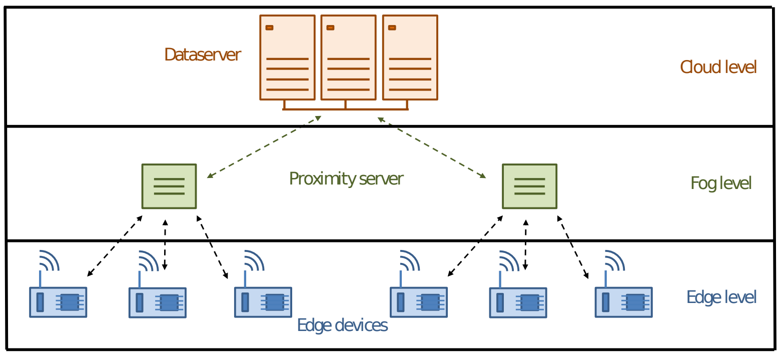

Figure 1. The computing continuum.

We have the sensor boards at the lower layer, known as the *Edge Level*. These boards work closely with high-performance, low-power computing resources to execute real-time applications with quick responsiveness services, utilizing data collected at the same location. However, despite their usefulness, these boards are hindered by limited memory and CPU with decreased computing power, requiring assistance from higher layers in the architecture for applications that demand more computational power.

At the middle layer, also called the *Fog Level*, we find proximity servers whose primary goal is to support edge-computing boards while reducing communication with servers located far on the network. These servers can be specific devices or one of the elected edge nodes designated as a trusted node. Regardless of their form, they play a critical role in the overall infrastructure: they are responsible for crucial tasks, such as intrusion detection, fault tolerance, data caching, workload balancing and dependability assurance.

The highest layer of the infrastructure is called the *Cloud Level*, and this encompasses centralized and powerful computing units, such as large clusters and high-performance dataservers. However, due to their distance from edge devices, these resources have significant latency, and thus are not well-suited for edge applications with time constraints. Despite this, they can provide additional computing resources and help construct global models.

It is crucial to emphasize that the security and trustworthiness of an edge-computing environment are of the utmost importance. The various layers of the architecture must work together to ensure that sensitive data is protected and that the system is resistant to tampering and other forms of compromise. Trustworthy nodes, secure communication protocols and robust intrusion detection and response mechanisms are critical components of a secure and reliable infrastructure.

The above shows that the lowest level of the described architecture, i.e., the edge level, is a critical issue. The presence of multiple edge devices seeking computational support from upper layers increases the likelihood of falling short of delivering highly responsive applications, presenting a significant challenge from the security point of view. However, by utilizing sensor boards equipped with advanced computing capabilities, we can significantly reduce the number of requests to the higher levels by implementing suitable security protocols concurrently to the edge applications, thereby, ensuring exceptional service quality and elevating the level of data privacy and trustworthiness.

The Nvidia Jetson Nano board [24] (2019) is a compact and powerful computing solution with dimensions of 70 mm $\times$ 45 mm, making it a suitable choice for space-constrained applications. The device features a quad-core ARM Cortex-A57 CPU with a peak performance of 22.8 Gflops, a GPU with 128 CUDA cores providing a peak performance of 471 Gflops, 4 GB of LPDDR4 memory and integrated GPU support for computer vision and deep-learning tasks.

The device also has Gigabit Ethernet, HDMI 2.0, USB 3.0 ports and an M.2 slot for additional storage. In addition, the Jetson Nano runs on the Ubuntu Linux operating system and supports popular machine-learning and AI frameworks. It also has a 40-pin GPIO header for external interfacing with peripherals and a 5V power supply with a Thermal Design Point (TDP) of 5 or 10 Watts. Section 5 outlines our experimental approach, where we leverage TDP to evaluate the upper limit of electric power needed during the execution of practical applications on the CPU.

4. Materials and Methods

The purpose of this section is twofold. First, it summarizes the adaptive K-means algorithm, which was originally introduced in [20,21] for multi-core CPUs. Secondly, it delves into the various parallel programming models that can be exploited in this algorithm and how they can be combined into a single hybrid approach with attention given to the architecture of the underlying computing systems.

The K-means algorithm is a well-studied machine-learning technique (e.g., [25]). It is an iterative procedure processing a dataset $S = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ of vectors with d components. The goal is to create a partition $\mathcal{P}_K = \{C_1, \dots, C_K\}$ of S, which is comprised of K subsets $C_k \subset S$, referred to as *clusters*, each containing N_k homogeneous elements as defined by a specified similarity metric. The algorithm typically employs the Euclidean distance to measure the similarity of the elements with a representative $\mathbf{c}_k$ of the cluster C_k called *centroids*. These are vectors in which each component is the average of the corresponding components of the items in the cluster.

At each iteration, each element $\mathbf{x}_n$ is moved to a new cluster C_δ determined by minimizing the Euclidean distance $||\mathbf{x}_n - \mathbf{c}_\delta||_2^2$ from the centroid $\mathbf{c}_\delta$. The procedure terminates when the number of reassignments of the elements is considered negligible.

A critical constraint of the K-means algorithm is that it can be challenging to define the value of K in specific problems from the input data. To overcome this, we developed an algorithm in [20,21], which adjusts the value of K until it meets the user's desired quality criteria.

Out of the numerous indices present in the literature, we utilized the root mean square standard deviation (RMSSD) [25] as a measure:

$$R_{\mathcal{P}K} = \left[\frac{\sum_{k=1}^{K} \sum_{\mathbf{x}_n \in C_k} ||\mathbf{s}_n - \mathbf{c}_k||_2^2}{d(N-K)} \right]^{1/2} \tag{1}$$

The RMSSD measures the average homogeneity of elements within clusters C_k. This method enables the algorithm to increase the number of clusters K until (1) no longer indicates a significant reduction with increasing a new cluster. This technique for determining the optimal number of clusters is called the "Elbow" method [26].

In our iterative procedure, as the number of clusters increases, the partition $\mathcal{P}_{K-1}$ is able to group elements into K clusters based on their similarity. Our algorithm focuses only on clusters with low affinity to reduce the computational cost primarily associated with element displacement among clusters. The standard deviation

$$V_k = \sqrt{\frac{1}{N_k - 1} \sum_{n=1}^{N_k} ||\mathbf{x}_n - \mathbf{c}_k||_2^2} \tag{2}$$

is used to measure the similarity of the elements in a single cluster. A reduction in the value of V_k correlates with an increase in the similarity of the elements $\mathbf{x}_n$ to the centroid $\mathbf{c}_k$. Therefore, at each iteration K, our algorithm divides the cluster $C_\gamma \in \mathcal{P}K - 1$, with the highest standard deviation, into two equally sized subclusters, $C\alpha$ and C_β. The new partition $\mathcal{P}K$ is then updated as follows:

$$\begin{array}{lll} K = 0 & \mathcal{P}_0 = \{C_0\} & \text{where} \quad C_0 \equiv S \\ K \geq 1 & \mathcal{P}_K = \mathcal{P}_{K-1} - \{C_\gamma\} \cup \{C_\alpha, C_\beta\} & \end{array} \tag{3}$$

The procedure may be terminated when the similarity of the elements within the clusters exhibits minimal or negligible change as indicated by the insignificance of variation in the root mean square standard deviation (RMSSD). Based on this definition, we present the following algorithm (Algorithm 1):

Algorithm 1: Adaptive Clustering Algoritm

```
(1) Initialize $K = 0$
(2) **repeat**:
    (2.1) Increase $K = K + 1$
    (2.2) determine the cluster $C_\gamma$ such that $V_\gamma = max_{k=1,\dots,K-1} V_k$
    (2.3) define the new partition of clusters $\mathcal{P}_K$ as in (3)
    (2.4) **repeat**:
        (2.4.1) **for** each cluster $C_k \in \mathcal{P}$: update clusters info
        (2.4.2) **for** each $\mathbf{x}_n \in S$: search the cluster $C_\delta$ such that
                the Euclidean distance $||\mathbf{x}_n - \mathbf{c}_\delta||_2^2$ is minimal
        (2.4.3) **for** each $\mathbf{x}_n \in S$: move $\mathbf{x}_n$ to $C_\delta$
      **until** (the number of reassignment is negligible)
    (2.5) update the RMSSD
  **until** (the variation of RMSSD is negligible)
```

Notably, the strategy of processing only subproblems with low values of a specific quality index is a general approach in various application domains, referred to as *adaptive procedures*. Notable examples include minimizing discretization errors in numerical methods or computational fluid dynamics (e.g., [27–29]). Hence, we deemed it appropriate to designate the above-mentioned procedure as the adaptive K-means algorithm.

The development of edge computing applications requires understanding the current architecture of high-performance computing systems, which comprise hybrid nodes integrating multi-core CPUs and floating-point accelerator devices, such as GPUs [30]. Multi-core CPUs allow a shared-memory multiple instruction multiple data (MIMD) programming paradigm, suitable for managing large concurrent tasks. On the other hand, accelerator devices provide exceptional performance on procedures characterized by pronounced data parallelism, utilizing the single instruction multiple data (SIMD) programming model. Thus, the Nvidia Jetson Nano board is well-suited for testing hybrid algorithms that combine these two programming models.

A multi-core CPU has multiple independent computing units with shared access to the main memory. To overcome the potential bottleneck caused by this shared access, each computing unit is equipped with multiple levels of cache memory and a private set of registers. This configuration enables the operating system (OS) to dispatch concurrent threads to each computing unit, resulting in the execution of an algorithm following the multiple instruction multiple data (MIMD) programming model. However, concurrent access to shared resources can result in race conditions, leading to unpredictable outcomes. To mitigate this issue, semaphores can be employed to define sections of code that can only be accessed by one thread at a time.

In contrast, floating-point accelerators implement a separate parallel programming model, originally designed to enhance image processing operations, such as texture mapping, rendering and pixel manipulation. However, with the advent of high-level programming environments, these devices can also be used for general-purpose applications. They are based on thousands of computing cores executing instructions under a common control structure and implementing a vector processing form of SIMD.

This architecture makes them ideal for high-throughput computations exhibiting strong data parallelism. In addition, it supports traditional CPUs in applications characterized by massive vector operations. In this case, such computing units can perform much better than conventional CPUs. Examination of the computational kernel of Algorithm 1, Step 2.4, reveals different models of parallelism that can be applied to the data structures managed therein.

Task 2.4.1, responsible for updating the cluster information, affords a *parallelism at the cluster level*, where concurrent tasks can update the K data structure describing the clusters. In real problems, the number of clusters, K, is much smaller than the number of elements, N, thereby, enabling efficient management of this parallelism via multi-core CPUs. The operating system schedules independent threads on the CPU's computing units with P threads executing on P subpartitions $\mathcal{P}_K^j \subset \mathcal{P}_K$, $j = 1, \ldots,$ P, and clusters C_k are assigned to a thread t_j in a round-robin fashion. It is possible to then rewrite Step (2.4.1) as follows.

Refinement of Step 2.4.1

```
(2.4.1-a)   **for** each $\mathcal{P}_K^j$ $(j = 1, \ldots, P)$ **in parallel**
    (2.4.1-b)   **for** each cluster $C_k \in \mathcal{P}_K^j$: update clusters info
            **end parallel for**
```

Step 2.4.2 shows a thinner degree of parallelism indicated by the number of elements N. In this instance, a *parallelism at the element level* can be established, which can be executed by either a multi-core CPU utilizing the SPMD programming model or a GPU-based accelerator using the SIMD programming model. For such a reason, it is possible to implement this step through a hybrid procedure that splits the original dataset S into two subsets of elements S_{CPU} and S_{GPU}. More precisely, with the peak performances of the GPU (PP_{GPU}) and the CPU (PP_{CPU}), we then define the ratio Q by:

$$Q = \frac{PP_{GPU}}{PP_{CPU}} \tag{4}$$

Then, the number of elements N_{CPU} and N_{GPU} assigned to each computing unit can be set proportional to the performance ratio Q as defined by:

$$N = \frac{N}{1+Q} + \frac{NQ}{1+Q} = N_{CPU} + N_{GPU} \tag{5}$$

We define P as the number of threads in execution on the multi-core CPU, and Step 2.4.2 determines P subsets $S^j \subset S_{CPU}(j = 1, \ldots, P)$ comprised of N_{CPU}/P elements, each assigned to a different thread t_j according to the SPMD programming model. The remaining set S_{GPU}, comprised of N_{GPU} elements, can then be processed by the floating-point accelerator device through the SIMD programming model executed. Thus, Step 2.4.2 is revised to reflect the concurrent execution of Step 2.4.2-a on the CPU and Step 2.4.2-b on the GPU.

```
Refinement of Step 2.4.2
(2.4.2-a) **for** each $\mathbf{x}_n \in S^{GPU}$ **in parallel**
        search the cluster of belonging $C_\delta$
    **end parallel for**
( 2.4.2-b) **for** each $S^j$  $(j = 1, \ldots, P)$ **in parallel**
        **for** each $\mathbf{x}_n \in S^j$: search the cluster of belonging $C_\delta$
    **end parallel for**
```

Step 2.4.3 represents a critical region of the algorithm as it involves the reallocation of elements $\mathbf{x}_n$ to new clusters with a high likelihood of race conditions arising from concurrent access to the data structures of the algorithm by multiple threads. This step is the only section of the algorithm executed as a sequential procedure in Algorithm 1 to address this issue.

The described algorithm can then be utilized to enhance the security and trustworthiness in an edge-computing environment by classifying nodes based on their behavior. For example, let S be the set of devices connected to the network, and then each device $\mathbf{x}_n \in S$ can be represented by a d-dimensional vector that describes its static characteristics, such as the type of connection, the computing power, the available memory or the kind of connected sensors as well as its dynamic features, such as the remaining battery charge, the workload, the actual availability, the scalability and the historical behavior. A complete set of metrics that can be used to evaluate the dependability of an edge-computing environment can be found in [31].

The algorithm can, therefore, operate by grouping nodes with similar features from the trustworthiness point of view and treating them as a single entity. Once these nodes are identified, the algorithm can implement security protocols to protect the system, such as limiting access to specific data or isolating the node from the rest of the system. Furthermore, utilizing a clustering algorithm for node classification based on their trustworthiness optimizes resource usage, reducing energy consumption and increasing the execution speed of applications, thus, enabling the creation of a more efficient and secure edge-computing environment that can support a wide range of applications.

5. Experiments and Discussion

5.1. Design

This section focuses on experiments aimed to evaluate the adaptive K-means algorithm from both the performance and energy consumption points of view, when executed on low-power devices for edge-computing environments. As a reference device, we employed the Nvidia Jetson Nano board as described in Section 3.

The experimentation session was divided into three parts, each with a different objective. The first set of experiments was aimed at determining the best value of TDP to use by evaluating the total energy required to perform the tests. The second set, on the other hand, was aimed at measuring the algorithm's performance on the considered datasets by assessing the gain obtained from using the parallel resources available on the board. Finally, the third set of experiments was aimed at comparing the executions with two other high-level GPU-based systems by evaluating the trade-off between performance and energy consumption.

As case studies for the experiments, we chose four datasets from the Machine Learning Repository at the University of California [32]. These datasets were chosen to represent a variety of use cases associated with security threats and trustworthiness issues in the context of edge computing in order to provide stronger validation for the achieved results. Moreover, they display variations in terms of the number of clusters (K), the number of elements (N) and the number of attributes (d).

- The Firewall dataset [33] comprises $N = 65{,}532$ elements, each representing an access on a firewall. These elements are characterized by $d = 11$ numerical features and are to be classified into $K = 4$ clusters.
- The Drive dataset [34] aims to detect defective components in a car. It comprises $N = 58{,}509$ electric drive signals described by $d = 49$ numerical attributes. The signals are to be classified into $K = 11$ different classes.
- The Phishing dataset [35] is related to the identification of phishing websites. It contains $N = 1353$ elements, each representing a website described through $d = 10$ features. These websites are to be classified into $K = 3$ clusters (phishing, legitimate or suspicious).
- The Wireless dataset [36] involves classifying $N = 7840$ 2.4 GHz radio signals based on their strengths in an indoor location. They are classified into $K = 4$ locations based on $d = 5$ features.

5.2. Experiments

The first set of experiments evaluates energy efficiency with the Nvidia Jetson Nano power set to 5 and 10 Watts. To estimate the energy consumption, the product of the execution time (in seconds) and TDP (in Joules/sec) is considered. More specifically:

$$H^{(TDP)} = T_4 \cdot TDP = [sec] \cdot \frac{[Joule]}{[sec]} \tag{6}$$

where T_4 represents the execution time with four threads on the multi-core CPU, and $H^{(TDP)}$ in (6), therefore, serves as an estimate of the energy consumed in executing Algorithm 1 with a given TDP setting.

Despite the fact that TDP is a measure of the maximum amount of heat that a CPU can generate under maximum workload and is often used as a reference parameter for selecting the CPU and cooling system, it is commonly used to compare the energy efficiency of CPUs when running the same application [37,38]. CPUs with the same TDP should exhibit similar performance, which means that a CPU with a smaller execution time for an application, at the same TDP, can be considered more energy-efficient. This comparison can be assessed through the following ratio:

$$R_1 = H^{(5)} / H^{(10)}$$

The previous expression measures how much lower the estimation of energy consumption of the execution is with the 5 Watt setting compared to those with the 10 Watt setting used as a baseline.

Figure 2 shows the total execution times in seconds T_4 with the two settings, and we also present the values of the energy consumption ratio:

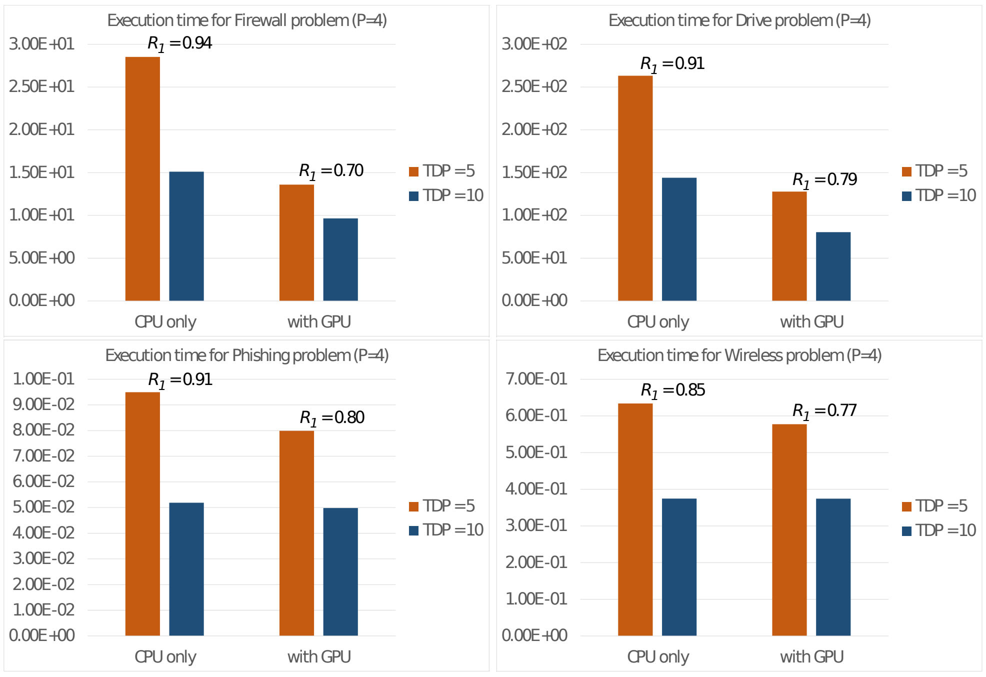

Figure 2. Execution times on the Nvidia Jetson Nano with the two settings 5 and 10 Watts and energy consumption ratio R_1.

The results revealed that the ratio $R_1 = H^{(5)}/H^{(10)}$ is always less than 1, indicating that the 5 Watt setting requires a lower overall energy consumption despite the longer execution times. As a result, all further experiments on the Nvidia Jetson Nano board were conducted using the 5 Watt setting.

The second set of experiments evaluates the algorithm performance on the Nvidia Jetson nano board. We first measured its execution time (T_P) as it ran only on the host CPU with P threads by assigning all N elements to the CPU with $Q = 0$ in Equation (5). Furthermore, we also measured execution time (T_P^*) while allocating N elements on both the CPU and GPU as described in Equation (5) by using $Q = 471/22 = 21$. Finally, from these two values, we compute the ratio:

$$R_2 = T_P/T_P^*$$

This metric quantifies the performance improvement achieved by utilizing all available computational resources on the board (i.e., GPU + CPU) compared to using only the CPU for execution. Figure 3 displays the execution times on the CPU alone (T_1 and T_4), as well as with the GPU (T_1^* and T_4^*) along with the performance gains (R_2).

Regarding the criticality of energy consumption in low-power devices, a third set of experiments was performed to compare the trade-off between the energy consumption and execution time on the Nvidia Jetson Nano with two other systems that utilize a GPU as a floating-point accelerator:

(TK20) utilizes a host CPU with a 4-core Intel Core i7-950 clocked at 3.07 GHz. The accompanying floating point accelerator is an Nvidia Tesla K20c GPU introduced in 2012, boasting 2496 CUDA cores with a clock speed of 0.706 GHz, a global memory capacity of 5 GBytes and a TDP of 225 Watts.

(RTX) is based on a host CPU with an 8-core Intel Core i9-9900K running at a frequency of 3.6 GHz. The accelerator unit is the more recent Nvidia GeForce RTX 3070 GPU introduced in 2020, having 5888 CUDA cores operating at 1.75 GHz, 8 GBytes of global memory and a TDP of 220 Watts.

Figure 4 then displays the execution time T_4^* and the values of the ratio

$$R_3 = H^{(GPU)}/H^{(Jetson)},$$

which measures how much higher the energy consumption $H^{(GPU)}$ of the two high-end GPU boards are compared to the equivalent value of the Nvidia Jetson Nano $H^{(Jetson)}$ used as a baseline.

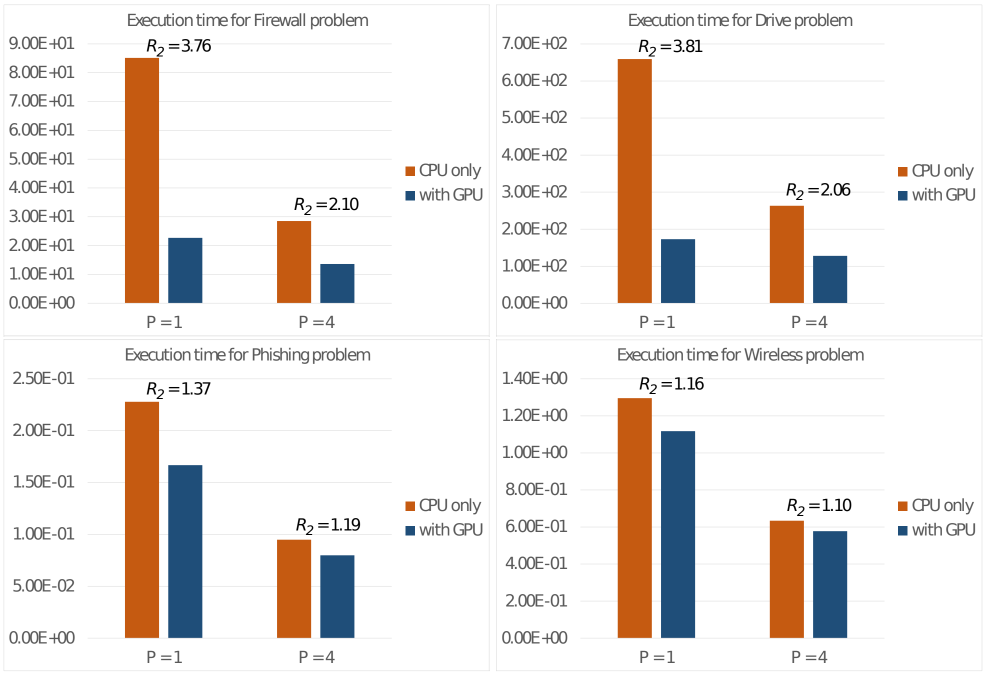

Figure 3. Execution times (in seconds) of Algorithm 1 on the Nvidia Jetson Nano board and performance gains R_2 achieved using a GPU.

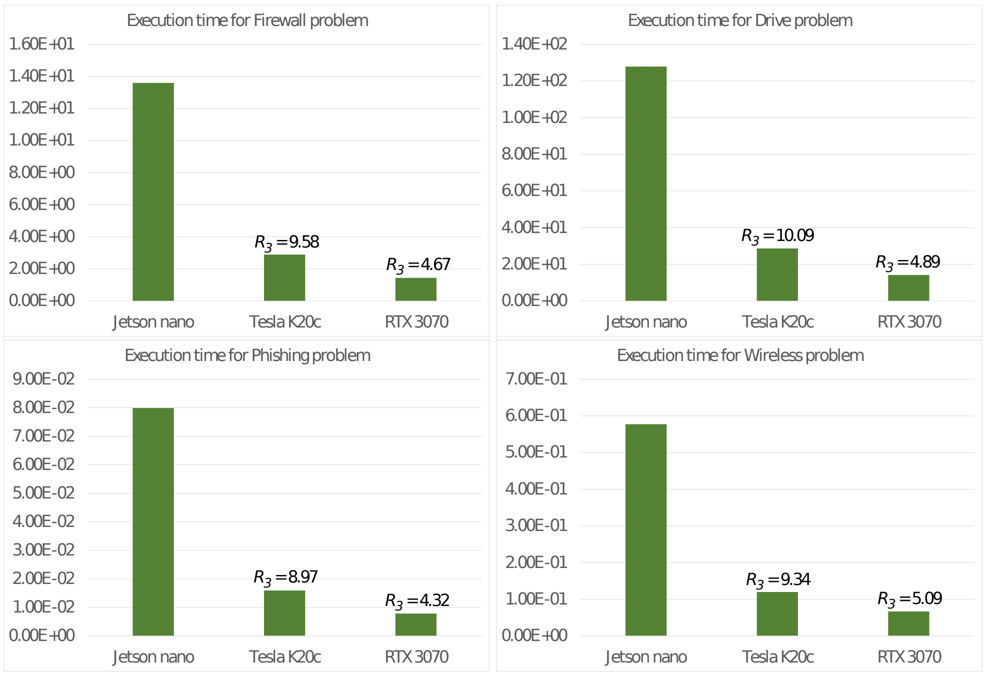

Figure 4. Execution times (in seconds) and energy consumption ratio R_3 of the two high end GPU boards compared with the Nvidia Jetson Nano board.

5.3. Discussion

The experiment's results, as shown in Figure 3, reveal a general and noticeable improvement in performance due to utilizing both the multi-core CPU and GPU concurrently. This improvement is reflected by the performance gain R_2. However, it is important to note that the actual performance gain of the described method strongly depends on the size of the problem, specifically the number of clusters K, the number of elements N and the dimension d.

From the figure, we note that this gain is not as substantial when dealing with low-dimensional problems, such as the Phishing problem (having a dataset of $N = 1353$ elements) or the Wireless problem (with $N = 7840$ elements). This is because these smaller problems do not fully tap into the computational capabilities of the GPU in Step 2.4.2 of the algorithm. Problems of larger dimensions (e.g., the Firewall and Drive problems with $N = 65{,}532$ and $N = 58{,}509$ elements, respectively) instead allow a full utilization of the computational power of the computing units.

Figure 4 shows the results of the energy consumption tests, indicating that the total energy consumption of Algorithm 1 when it ran on the Nvidia Jetson Nano was significantly lower compared to that of the other two considered devices. Despite the longer execution time, the energy consumption of the Nvidia Jetson Nano ranged from 4 to 10 times less as confirmed by the measured values of R_3. This means a higher number of operations performed per single Joule consumed. We did not observe any significant differences in the R_3 ratio with respect to dataset size.

The experiments yielded outcomes showing that it is possible to achieve a favorable trade-off between low energy consumption and high performance when implementing clustering algorithms on advanced edge-computing devices, such as the Nvidia Jetson Nano board. While energy efficiency can be maximized, this comes at the cost of decreased performance, which may not be ideal for specific real-time applications. However, with the low cost of these devices, available on the consumer market for as little as 100 to 200 Euros, it is easy to increase the number of data collection and processing points if necessary. The use of proximity servers to secure the network edge also exemplifies the versatility and efficiency of this approach, as this enables the aggregation of computational power only in areas where it is needed without using more energy-intensive devices.

6. Conclusions

Edge computing has recently gained significant attention due to the increasing use of low-power devices that offer a fusion of computing power and sensors all in one compact board. However, despite its benefits, this environment also poses several serious challenges, such as the likelihood of failures and security breaches caused by the heterogeneous nature of resources and decentralized control structure.

To tackle this challenge, we focused our efforts into creating a hybrid clustering algorithm that categorizes nodes based on their trustworthiness, thus, enhancing the security aspect in edge computing systems. To assess the algorithm's performance, we performed experiments using the Nvidia Jetson Nano board—a handy device suitable as both an edge node and a proximity server. We aimed to gauge the algorithm's potential in providing decision-making support for security-related issues at the network's edge, thereby, eliminating the need for central servers. Furthermore, we tested several datasets to cover a wide range of scenarios by optimizing the algorithm for the board's architecture through a hybrid SPMD/SIMD approach.

The study results indicate that high-performance and low-power edge-computing devices offer a better balance between performance and energy efficiency compared with high-end computing systems. Furthermore, effectively utilizing these devices with hybrid clustering algorithms can facilitate the broader deployment of trustworthy and smart nodes at the network edge without the need for central servers. This advancement opens up new possibilities for multiple scientific and societal applications.

In any case, beyond a testing session on real data that also utilizes tools for energy consumption estimation other than TDP, further research and experiments are necessary before the actual and systematic utilization of our method in a real-world environment. For instance, the development of a distributed-memory version of the clustering algorithm running on several edge devices could allow the implementation of a federated learning model. Furthermore, high-performance FPGA-based accelerator devices, such as Xilinx Alveo, have recently shown promise in implementing methods for ML. We propose that the combination of these additional research efforts could contribute to achieving an efficient, reliable and secure edge-computing environment. These aspects will be the topics of future works.

Author Contributions: Methodology, M.L. and D.R.; Software, M.L. and V.M.; Validation, D.R.; Formal analysis, M.L.; Investigation, M.L., V.M. and D.R.; Resources, M.L.; Data curation, V.M.; Writing—original draft, M.L. and D.R.; Writing—review and editing, M.L., V.M. and D.R.; Supervision, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing, funded by the European Union in the framework of the Next Generation EU plan.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Beckman, P.; Dongarra, J.; Ferrier, N.; Fox, G.; Moore, T.; Reed, D.; Beck, M. Harnessing the Computing Continuum for Programming Our World. In *Fog Computing: Theory and Practice*; Zomaya, A., Abbas, A., Khan, S., Eds.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2020.
2. Cheptsov, A.; Koller, B.; Adami, D.; Davoli, F.; Mueller, S.; Meyer, N.; Lazzari, P.; Salon, S.; Watzl, J.; Schiffers, M.; et al. E-Infrastructure for Remote Instrumentation. *Comput. Statd. Interfaces* **2021**, *34*, 476–484. [CrossRef]
3. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
4. di Luccio, D.; Riccio, A.; Galletti, A.; Laccetti, G.; Lapegna, M.; Marcellino, L.; Kosta, S.; Montella, R. Coastal Marine Data Crowdsourcing Using the Internet of Floating Things: Improving the Results of a Water Quality Model. *IEEE Access* **2020**, *8*, 101209–101223. [CrossRef]
5. Romano, D.; Lapegna, M.; Mele, V.; Laccetti, G. Designing a GPU-parallel algorithm for raw SAR data compression: A focus on parallel performance estimation. *Future Gener. Comput. Syst.* **2020**, *112*, 695–708. [CrossRef]
6. Lapegna, M.; Stranieri, S. DClu: A Direction-Based Clustering Algorithm for VANETs Management. In *Innovative Mobile and Internet Services in Ubiquitous Computing, Proceedings of the 15th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2021), Asan, Republic of Korea, 1–3 July 2021;* Lecture Notes in Networks and Systems; Springer Nature: Cham, Switzerland, 2022; Volume 279, pp. 253–262.
7. Balzano, W.; Lapegna, M.; Stranieri, S.; Vitale, F. Competitive-blockchain-based parking system with fairness constraints. *Soft Comput.* **2022**, *26*, 4151–4162 [CrossRef]
8. Deng, S.; Zhao, H.; Fang, W.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet Things J.* **2020**, *7*, 7457–7469. [CrossRef]
9. Bao, F.; Chen, I.; Chang, M.; Cho, J. Hierarchical Trust Management for Wireless Sensor Networks and its Applications to Trust-Based Routing and Intrusion Detection. *IEEE Trans. Netw. Serv. Manag.* **2012**, *9*, 169–183. [CrossRef]
10. Kammoun, N.; Abassi, R.; Guemara, S. Towards a New Clustering Algorithm based on Trust Management and Edge Computing for IoT. In Proceedings of the 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 1570–1575.
11. Dongarra, J.; Gannon, D.; Fox, G.; Kennedy, K. The Impact of Multi-core on Computational Science Software. *CTWatch Q.* **2007**, *3*, 1–10.
12. Wang, T.; Wang, P.; Cai, S.; Ma, Y.; Liu, A.; Xie, M. A Unified Trustworthy Environment Establishment Based on Edge Computing in Industrial IoT. *IEEE Trans. Ind. Inf.* **2020**, *16*, 6083–6091. [CrossRef]
13. Sun, Y.; Ochiai, H.; Esaki, H. Decentralized Deep Learning for Multi-Access Edge Computing: A Survey on Communication Efficiency and Trustworthiness. *IEEE Trans. Artif. Intell.* **2022**, *3*, 963–972. [CrossRef]
14. Wang, T.; Qiu, L.; Sangaiah, A.K.; Liu, A.; Bhuiyan, M.Z.A.; Ma, Y. Edge-Computing-Based Trustworthy Data Collection Model in the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4218–4227. [CrossRef]
15. Li, E.; Zhou, Z.; Chen, X. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In Proceedings of the 2018 Workshop on Mobile Edge Communications, MECOMM@SIGCOMM 2018, Budapest, Hungary, 20 August 2018; pp. 31–36.
16. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017.

17. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
18. Lee, J.; Eshraghian, J.K.; Cho, K.; Eshraghian, K. Adaptive precision CNN accelerator using radix-x parallel-connected memristor crossbars. *arXiv* **2019**, arXiv:1906.09395.
19. Sze, V.; Chen, Y.; Yang, T.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
20. Lapegna, M.; Mele, V.; Romano, D. An Adaptive Strategy for Dynamic Data Clustering with the K-Means Algorithm. In *Parallel Processing and Applied Mathematics, Proceedings of the PPAM 2019, Bialystok, Poland, 8–11 September 2019;* Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12044, pp. 101–110.
21. Laccetti, G.; Lapegna, M.; Mele, V.; Romano, D.; Szustak, L. Performance enhancement of a dynamic K-means algorithm through a parallel adaptive strategy on multi-core CPUs. *J. Parallel Distrib. Comput.* **2020**, *145*, 34–41. [CrossRef]
22. Lapegna, M.; Balzano, W.; Meyer, N.; Romano, D. Clustering Algorithms on Low-Power and High-Performance Devices for Edge Computing Environments. *Sensors* **2021**, *21*, 5395. [CrossRef]
23. Laccetti, G.; Lapegna, M.; Montella, R. Toward a high-performance clustering algorithm for securing edge computing environments. In Proceedings of the 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Taormina, Italy, 16–19 May 2022; pp. 820–825.
24. Nvidia Jetson Nano Documentation. Available online: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed on 12 February 2023).
25. Gan, D.G.; Ma, C.; Wu, J. *Data Clustering: Theory, Algorithms, and Applications*; ASA-SIAM Series on Statistics and Applied Probability; SIAM: Philadelphia, PA, USA; ASA: Alexandria, VA, USA, 2007.
26. Halkidi, M.; Batistakis, Y.; Vazirgiannis, M. On clustering validation techniques. *Journ. Intell. Inf. Syst.* **2001**, *17*, 107–145. [CrossRef]
27. Laccetti, G.; Lapegna, M.; Mele, V.; Montella, R. An adaptive algorithm for high-dimensional integrals on heterogeneous CPU-GPU systems. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4945. [CrossRef]
28. Thompson, J.F. A survey of dynamically-adaptive grids in the numerical solution of partial differential equations. *Appl. Numer. Math.* **1985**, *1*, 3–27. [CrossRef]
29. Haase, W.; Misegades, K.; Naar, M. Adaptive grids in numerical fluid dynamics. *Numer. Methods Fluids* **1985**, *5*, 515–528. [CrossRef]
30. The Top 500 List. Available online: https://www.top500.org (accessed on 12 February 2023).
31. Junior, F.M.R.; Kamienski, C.A. A Survey on Trustworthiness for the Internet of Things. *IEEE Access* **2021**, *9*, 42493–42514. [CrossRef]
32. Dua, D.; Graff, C. UCI Machine Learning Repository. In *Science*; School of Information and Computer, University of California: Irvine, CA, USA, 2019. Available online: http://archive.ics.uci.edu/ml (accessed on 12 February 2023)
33. Ertam, F.; Kaya, M. Classification of firewall log files with multiclass support vector machine. In Proceedings of the sixth International Symposium on Digital Forensic and Security (ISDFS), Antalya, Turkey, 22–25 March 2018; pp. 1–4.
34. Paschke, F.; Bayer, C.; Bator, M.; Mönks, U.; Dicks, A.; Enge-Rosenblatt, O.; Lohweg, V. Sensorlose Zustandsüberwachung an Synchronmotoren. In Proceedings of the Workshop Computational Intelligence, Dortmund, Germany, 5–6 December 2013; Volume 46, pp. 211–225.
35. Abdelhamid, N.; Ayesh, A.; Thabtah, F. Phishing detection based Associative Classification data mining. *Expert Syst. Appl.* **2014**, *41*, 5948–5959. [CrossRef]
36. AlHajri, M.I.; Ali, N.T.; Shubair, R.M. Classification of Indoor Environments for IoT Applications: A Machine Learning Approach *IEEE Antennas Wirel. Propag. Lett.* **2018**, *17*, 2164–2168.
37. Tahoori, M.B.; Pedram, M. *Energy-Efficient Computing*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013
38. Ouyang, Y.; Liu, X.; Luo, J.; Zhao, Y. Characterizing Energy Efficiency and Performance of Parallel Workloads on Multicore Systems. *IEEE Trans. Comput.* **2018**, *62*, 1506–1520.