Accelerating Anomaly Detection Process in the Manufacturing Setting Using HPC Systems

1st Javad Fadaie Ghotbi
High Performance Computing Center, Stuttgart
Stuttgart, Germany
javad.fadaieghotbi@hlrs.de

2nd Kamil Tokmakov
High Performance Computing Center, Stuttgart
Stuttgart, Germany
kamil.tokmakov@hlrs.de

3rd Julen Aperribay
IDEKO, member of Basques research and technology
Elgoibar, Spain
japerribay@ideko.es

4th Javier Martin
IDEKO, member of Basques research and technology
Elgoibar, Spain
jmartin@ideko.es

***Abstract*—In a manufacturing environment, where the production of high-value products is in high demand, machine availability and quality assurance are top priorities. The linear actuator plays a crucial role within these machines, and proper functioning and detect anomalies of the linear actuator is inevitable to maintain machine quality.**

To addressing this challenge, IDEKO employs an innovative approach involving the continuous monitoring of the linear actuator's position using multiple sensors. It utilizes clustering and classification methods to process time series position signals to detect anomalies in real-time or near real-time without interrupting machine operation. However, significant challenges arise due to the large volume of time series signals generated, and makes it impractical for IDEKO to effectively process and detect anomalies within these signals.

This paper introduces the HPC service, which is developed to expedite the anomaly detection process. This service offers a practical solution for end-users like IDEKO, enabling them to accelerate the detection of anomalies and ultimately improve manufacturing quality and efficiency within their specific settings.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Manufacturers of expensive high-value components prioritize machine availability and quality assurance, relying on advanced practices, such as predictive maintenance, assessing remaining component lifetimes, and diagnosing critical machine elements [1].

The linear actuator plays a crucial role among components in a machine tool. Thus, making the proper functioning of the actuator is essential to maintain machining quality. However, diagnosing the status of this component often requires using market-available products like AMM/C [2], which necessitates the machine to halt production for offline condition monitoring [3]. This interruption leads to reduced machine availability, which companies aim to avoid.

However, IDEKO has developed another approach, utilizing state-of-the-art techniques [4], where continuous data analysis occurs while the machine remains in operation, and real-time monitoring the linear actuator's state. The primary objective is to detect potential failures in advance, requiring the assessment of the linear actuator's health in nearly real-time. This task analyzes huge amounts of sensor data, that requires rapid responses and intensive computation. In this approach, K-means [5] a clustering algorithm is utilized to differentiate between normal and anomalous signals within a set of data. Subsequently, a classifier is employed to predict whether new signals are anomalous or not.

For the classification of this time series, the k-NN (k-Nearest Neighbors) algorithm has been selected, as it is widely used in the literature [6] and has demonstrated 100% accuracy in all conducted tests. As for the clustering algorithm, K-means has been chosen due to its capability to create exactly two clusters. Additionally, its relatively low time complexity and high computational efficiency make it a suitable choice for the task. The result of applying the K-means algorithm to a set of signals can be observed in Figure 1.

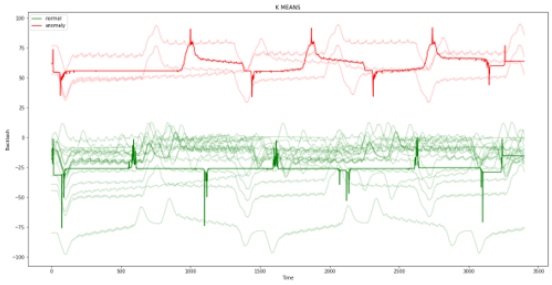

Fig. 1: K-means clustering for 22 signals.

This method has been successfully applied to linear actuators that employ a ball screw as a mechanical drive system (refer to Figure 2). In this particular scenario, we concentrate our monitoring on identifying errors related to the ball nut's positioning, since such errors cause highly accurate failures. The most relevant indicators for this analysis are the linear and angular positions of the axis, which are continuously acquired at a high frequency [7] to enable early anomaly detection.

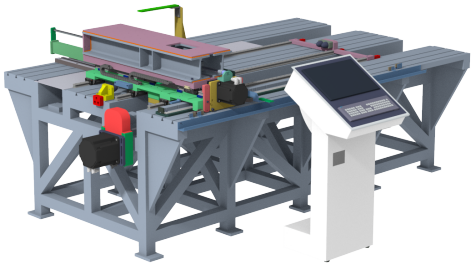

Fig. 2: Ball screw mechanical drive system linear actuator.

While this approach allows for near real-time machine monitoring, it faces problems in practice. Dealing with a large amount of data, including time series with up to 3400 data points every 7 seconds, and performing classification and clustering on as many as 1100 time series, becomes impossible on a regular desktop computer.

To overcome these challenges, SERRANO platform [8] is provisioning of hardware-agnostic kernels for acceleration of computationally intensive tasks in order to minimize the execution time. The acceleration targets various devices in cloud and edge environments, such as GPU and FPGA, as well as HPC platforms. Furthermore, the resources of cloud, edge, and HPC nodes introduce significant challenges for workload processing under real-time requirements. Therefore, SERRANO platform addresses this challenge by applying transprecision and approximate computing techniques at algorithmic, software and hardware layers. Previous work [9] has addressed the acceleration and approximation techniques for kernels using GPU and FPGA, whereas in this paper, we discuss kernels executed in HPC.

The HPC service within the SERRANO platform accelerates the workflow of anomaly detection in the HPC environment. Therefore, it provides a practical solution for IDEKO to enhance machine quality in their manufacturing setting.

The HPC service have been developed using OpenMP/MPI frameworks in the HPC systems, that provide access to remote powerful computing resources, HPC systems with thousands of computing nodes are capable of processing large amounts of data and compute-intensive applications in a minimal runtime.

This paper makes several key contributions. In Section II, we describe the data preprocessing step, that is necessary for the raw time series signals provided by IDEKO. Moreover, Section II discusses the parallelization techniques in K-means and K-nearest neighbors (k-NN) methods for time series signals. Section III introduces transprecision and approximate computing, these methods will be employed to optimize the computational and memory resources. Moving forward to Section IV, we evaluate the clustering and classification methods using the signal batches provided by IDEKO. Section V will provide descriptions of Verification, Validation, and Uncertainty Quantification (VVUQ) as necessary tools developed within our HPC service. This tool suggests the optimal parameter combinations for achieving maximum performance when executing the kernel.

II. Data Processing and Parallelization

First, we describe the data processing procedure within our framework. The input signals provided by IDEKO are in CSV format and are assumed to be received and available in the HPC system's workspace. Therefore, the initial step involves using a data converter tool to process these input signals. This tool extracts the signals based on their indices from the original CSV data and converts them into the binary format. Once the conversion is completed, the signals are organized into disjoint sets based on their indices in the binary format and are ready to be processed. By converting the input data in this way, the memory footprint is considerably less compared to the processing of the original CSV data. Additionally, it allows for the workload to be easily distributed among multiple processes.

A. Parallelization Techniques

To achieve maximum performance, the implementation employs a two-level parallelization strategy. The first level involves task parallelization, where the signals are uniformly distributed among processes to enable workload distribution. This ensures that each process has a roughly equal workload, allowing them to independently handle a batch of signals. Since there are no data dependencies between the signals, this approach maximizes efficiency and reduces processing time.

The second level of parallelization focuses on distributed memory parallelization within the algorithms. This involves utilizing the MPI communication protocol to exchange data across processes as needed to generate accurate outputs. By incorporating both task and distributed memory parallelizations, the HPC service aims to optimize performance and achieve the desired results efficiently. The details of these parallelization techniques will be described in this section.

B. Parallel K-means

K-means is a non-supervised machine learning algorithm, and it is applied to cluster the data points in the data set into a specific number of groups based on their similarity. In this context, the K-means is used to cluster the time series signals into two disjoint groups. The K-means algorithm initially assigns two random centroids and proceeds to calculate the distance between the signals and these randomly selected centroids. The distance metric employed is Dynamic Time Wrapping (DTW) [10], which considers the temporal relation between time series signals and produces a higher-quality distance measure compared to the Euclidean norm. The signals are then assigned to the centroid class with the smallest distance. New centroids are computed by averaging the signals belonging to each class, and the assigned class for each signal is updated iteratively. This process is repeated until sufficient classification is achieved.

The parallelization scheme for K-means is outlined in Algorithm 1.

Algorithm 1 Parallel K-Means Algorithm with 10 Iterations

```
Distribute time series signal batches uniformly among processes.
Initialize two random centroid signals and broadcast them using MPI_Bcast method.
for i in range(10) do
    for each process do
        Compute distance to centroid signals using DTW metric.
        Assign signals to the closest centroid class.
    end for
    for each process do
        Compute new centroid locally by averaging.
    end for
    Gather local centroids using MPI_Reduce in the root process.
    Compute new global centroids by averaging gathered centroids and broadcast them using MPI_Bcast method.
end for
Gather all signal labels using MPI_Gather.
if current process is the root process then
    Output the signals with their labels.
end if
```

In this approach, the signals are uniformly distributed across the processes, with each process being responsible for a batch of signals. The random centroids are initially chosen in the root process and then broadcasted to all processes using *MPI_Bcast*. Each process independently computes the distance between its signals and centroids, and signals are then assigned to the centroid class that has a smaller distance. Concurrently, each process calculates its local centroids by averaging. The local centroids from each process are then gathered with *MPI_Gather* in the root process, where the global centroid is computed. The process is repeated by broadcasting this new centroid to all processes. Consequently, after 10 iterations, the signals are clustered into two distinct groups.

The following parallel K-means utilizes task parallelization and uses MPI methodology for communicating the data across processes. In this approach, the classification is performed using multiple processes, meanwhile ensuring unique classifications as long as initial centroids remain unchanged throughout each execution, therefore providing maximum performance and performing the classification in minimal time.

C. Parallel k-NN (K-Nearest Neighbors)

K-NN (K-Nearest Neighbors) is a supervised machine learning algorithm used for the classification of data points based on their similarities. In the context of time series analysis, k-NN is employed to classify inference signals using labeled training signals. K-NN calculates the distance between the inference signals and the training label signals, specifically, utilizing the DTW metric. Subsequently, a class label is assigned to the inference signals based on the majority class label among the k-nearest neighbors. The parallelization of k-NN is explained in Algorithm 2.

In this parallelization approach for k-NN, the training signals and their corresponding labels are initially distributed evenly across multiple processes. Subsequently, the inference signals are broadcasted to all processes using *MPI_Bcast*. Each process then independently computes the distances between the inference signals and the training signals using the DTW metric. Once their computations are complete, the distances from all processes are gathered using *MPI_Gather* into the root process. By considering the K-smallest distances and the majority class label within this set, the label of the inference signals are assigned. This parallelization approach for k-NN enhances performance and minimizes the total classification time; meanwhile, the classification quality remains unaffected when executed across numerous processors.

Algorithm 2 Parallel k-Nearest Neighbors Algorithm

```
Distribute signal batches uniformly among processes.
Place the inference signal in the root process.
Broadcast the inference signal to all processes.
Each process independently computes distances between training data and the inference signal using a DTW metric.
Gather distances from all processes using MPI_Gather into the root process.
Consider the k nearest distances and determine the label of the inference signal.
if current process is the root process then
    Output the signals with their labels.
end if
```

III. Transprecision and Approximation Computing Techniques

In Section II, we have discussed the parallelization of the clustering and classification algorithm used in the workflow of anomaly detection. In order to optimize the utilization of computational and memory resources, transprecision and approximate computing techniques are applied.

A. Transprecision Techniques in Implementation

Transprecision techniques optimize the utilization of memory resources by employing lower data precision during computation. This approach effectively reduces the memory footprint, resulting in a reduction in execution runtime. Importantly, despite the use of lower precision data, the accuracy of the final result is not necessarily compromised.

We have incorporated transprecision techniques into the implementation by utilizing template data types. Initially, we can have the input data in various precisions. As mentioned in Section II, the data is converted into binary format using template data types. This allows us to convert the initial data to different precisions, including lower precision. Additionally, our implementation includes template data structures, enabling us to dynamically apply different data types and execute the application with varying data types. Consequently, we can evaluate the optimization in execution time by employing lower data precision, while monitoring that the accuracy of the results is not significantly deteriorated.

B. Approximation Computing Techniques in K-means

Approximation computing techniques are employed to improve performance and reduce execution time in computations, while at the cost of altering the final result and compromising accuracy. Quality-based control loop is a concept used in

approximation computing. These loops determine the number of times specific parts of an algorithm are executed until the desired level of convergence is achieved. Quality-based control loops continuously monitor an internal metric and terminate iterations, when a condition based on the internal state and user-specified parameters is met, as shown in Formula (1) below. Typically, users rely on default values for these parameters or set conservative values, which may lead to sub-optimal performance.

$$while(error < epsilon)\{...\} \tag{1}$$

Quality-based control loop techniques can be incorporated into the implementation of K-means by monitoring the distance between centroids during each iteration. The while loop will terminate, if the difference between these centroids is smaller than a specified control parameter (epsilon). While the default implementation involves 10 iterations, different values of exponents such as *e-4*, *e-2* and *e-1* can be applied for epsilon. The aim is to reduce the number of iterations and execution time by using a larger epsilon value. However, it should be noted that clustering accuracy may be affected due to the approximation technique.

The parallel K-means implementation is integrated with approximation techniques outlined in Algorithm 3. It is expected that the execution runtime for clustering will decrease, without significant deterioration in the quality of classification.

Algorithm 3 Parallel K-Means Algorithm Using Approximation Computing Technique

```
Set control parameter ϵ.
Distribute training signal batches uniformly among processes.
Initialize two random centroid signals and broadcast them using MPI_Bcast method.
while difference between centroids is greater than ϵ do
    for each process do
        Compute distance to centroid signals using DTW metric.
        Assign signals to the closest centroid class.
    end for
    for each process do
        Compute new centroid locally by averaging.
    end for
    Gather local centroids using MPI_Reduce in the root process.
    Compute new global centroids by averaging gathered centroids and broadcast them using MPI_Bcast method.
end while
Gather all signal labels using MPI_Gather.
if current process is the root process then
    Output the signals with their labels.
end if
```

It should be noted that the approximation techniques for k-NN algorithm are not applicable due to a precise nature of the algorithm, therefore the relaxation parameters, such as less precise data type, were not applied.

IV. Numerical experiments and discussion

In this section, we conduct a numerical experiment using data batches provided by IDEKO to evaluate the execution time of the parallel K-means and k-NN algorithms, described in Section II. Additionally, we analyze the impact of employing approximation and transprecision computing techniques. For these experiments, HPE Apollo (Hawk) [11] supercomputer at High-Performance Computing Center, Stuttgart (HLRS) was utilised.

A. Parallel K-means

To evaluate the scalability property of parallel K-means, 110 position time series signals provided by IDEKO are utilized. As mentioned in Section II, these signals are originally in CSV format. It is assumed that data conversion has already been performed, and the signals are in binary format, separated by their indices. Table I demonstrates the scalability of the parallel K-means, where an increase in the number of processes leads to a reduction in average execution time. It also concludes that the maximum speedup can be achieved with 64 cores. Increasing the number of cores to 128 does not necessarily improve the execution time.

TABLE I: Scalability of Parallel K-means

Cores	Average execution time/sec	Speedup
1	402.418	-
2	208.136	1.9X
4	103.316	3.8X
8	51.3941	7.8X
16	27.2799	14.7X
32	15.1545	26.5X
64	10.1071	40.3X
128	10.1449	39.8X

In a similar way, the parallel K-means algorithm was tested with several data batches, including 110, 330, 550, 770, 990, and 1100 position signals. Figure 3 presents the maximum speedup for each data batch. We observed that the maximum speedup increased as the size of the data batches containing signals grew.

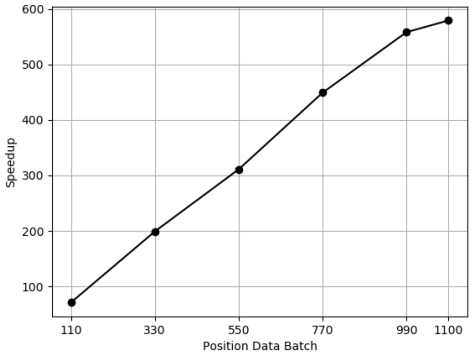

Fig. 3: Speedup of Parallel K-means for Different Data Batch Sizes

B. Parallel k-NN

To assess the scalability of parallel k-NN, a single inference signal is selected for classification alongside 110 training signals with their respective labels. Table II demonstrates

the decrease in execution time as the number of processes increments. The lowest execution time is observed with 128 cores.

TABLE II: Scalability of Parallel k-NN

Cores	Time/sec	Speedup
1	20.073	-
2	10.5859	1.8X
4	5.44434	3.6X
8	3.11582	6.4X
16	1.38314	14.5X
32	0.789509	25.7X
64	0.485754	41.8X
128	0.408626	50.1X

Figure 4 depicts the maximum speedup achieved through parallel k-NN when applied to multiple position training data batches, alongside their respective labels.

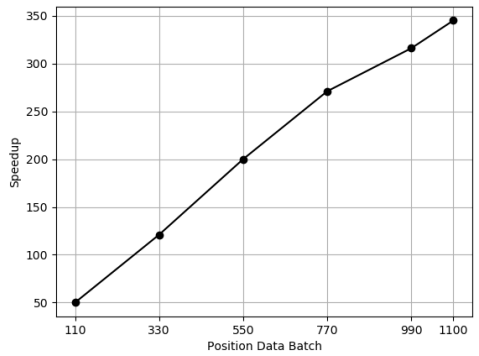

Fig. 4: Speedup of Parallel k-NN for Different Position Training Data Batch Sizes

It has been observed that the process of anomaly detection, which involves clustering (K-means) and classification (k-NN) of time series signals with different data batches, can be significantly accelerated through the parallelization of these kernels on HPC systems.

C. Transprecision and approximation computing techniques

We have tested the impact of quality based control loop in parallel K-means. To carry out this, different values of the control parameter (epsilon) are selected and re-executed the parallel K-means clustering on the 110 position signals with 64 Cores. As a result, we observed a significant reduction in the number of iterations from the initial value of 10 within the loop, down to just 3 (see Table III). Consequently, this decrease resulted in approximately a 3.1X improvement in speedup. Notably, the accuracy of clustering was not changed, and remained consistent across the different values of epsilon tested.

Transprecision techniques are applied to the parallel K-means. By employing two different data types: double precision (double) and single precision (float) for the input data. Table IV presents the results of executing parallel K-means clustering on 110 position signals using 64 cores, demonstrating the impact of selecting lower data precision for input data on the improvement of execution time.

TABLE III: Approximation computing techniques in parallel K-means

Epsilon	Iteration	Time/sec	Speedup
e-4	3	3.13392	3.1X
e-3	3	2.95766	3.3X
e-2	3	3.05517	3.3X
e-1	3	3.16488	3.1X

TABLE IV: Tranprecision computing techniques in parallel K-means

Input data percision	Time/sec	Speedup
$double$	10.1071	-
$float$	9.02348	1.1X

V. Verification, Validation and Uncertainty Quantification Framework

In Section III, the integration of the transprecision and approximation computing techniques into the kernel implementation was discussed. This integration enables an optimal use of memory and compute resources by allowing the kernel to be executed with multiple data precision for the input data, as well as adjusting the computation density of the kernel. By combining these parameters, there exists a large range of possibilities for parallel execution of the kernel.

The question that arises is what the optimal combination of these parameters is in order to achieve an efficient parallel execution of the kernel. The VVUQ (Verification, Validation, and Uncertainty Quantification) framework addresses this issue. This framework considers all of the parameters and determines the optimal parameter combination, including the number of cores, input precision, and computation density, in order to improve the performance of parallel applications.

To illustrate how this framework operates, consider the parallel K-means classifications as an example. In this scenario, the parallel K-means will be executed with the mixed data precision scenario for the input data, where $\{double, float\}$ are the available options. For approximation techniques, the control parameter epsilon takes any of the following values: $\{e-1, e-2, e-3, e-4\}$. Additionally, for parallel execution, the number of cores ($Cores$) takes values of {1, 2, 4, ..., 128, 256}. We execute the parallel K-means script described in Listing 1, while simultaneously monitoring the execution runtime, and centroid accuracy in each iteration, therefore, automatically generating multiple profiling files. This approach allows us to comprehensively evaluate the performance and trade-offs associated with different combinations of data precision and approximation techniques, providing complete insights of parallel execution of the K-means.

Listing 1: Bash script for parallel configuration

```
for InputPercision in {double,float} do
    for epsilon in {e-2,e-3,e-4} do
        for Cores in {1, ...,128,256} do
            mpirun np $Cores $epsilon
        done
    done
done
```

The VVUQ framework comes into play to analyze these profiling files. In the VVUQ interface (Listing 2), the kernel name and input data size can be specified, while leaving other parameters at their default values.

Listing 2: Bash parameters for VVUQ

```
kernel="K-means"
data_size="110"

input_data_precision="NULL"

control_paramter_offset="DEFAULT"
control_paramter_endset="DEFAULT"


.
.
./vvuq $kernel $data_size $input_data...
```

By utilizing the provided bash script, the optimal combination of the parameters to execute the parallel K-means can be obtained. Table V presents the optimal configuration for minimizing the execution time of the parallel K-means. It suggests using 68 cores and float precision for the input data. Additionally, considering a control parameter of 0.0001 is recommended.

TABLE V: Parallel parameter achieved by VVUQ framework

num_procs	control	Time/s	Input data precision
68	0.0001	0.000297	float

One potential drawback of utilizing this VVUQ framework is that it may lead to increased costs, as it requires more computational time to consider all combinations of data precision, control parameter (epsilon), and multiple numbers of cores, and then quantify uncertainties and validate the results. To tackle these difficulties, we plan to firstly employ the VVUQ framework on a small number of use cases. We will then combine these outcomes with the techniques used in Machine Learning (ML), such as Gradient Descent method, to derive the best parameter values for new data batches. By doing so, we can overcome the computational expenses associated with using the VVUQ framework while enhancing the efficiency and dependability of the HPC service.

For example, through the application of the VVUQ framework, we have obtained the minimum execution time for different data batches in each kernel. As a result, we have employed the Gradient Descent method [12] to derive a nonlinear formula that approximates the minimum execution time. This formula is subsequently utilized to estimate the minimum execution time required for the execution of parallel applications for signals we have obtained. In this approach, the Gradient Descent method was applied to compute the optimal parameters a, b, c, d in the following equation:

$$y = a.x^3 + b.x^2 + c.x + d$$

In this context, x represents the control parameter, denoting data batches, while y represents the minimum execution time. Utilizing the Gradient Descent method, we compute the optimal values for a, b, c, d. These parameters are used for predicting the minimum execution time of parallel applications for random signal batches on the HPC service. For instance, when running K-Means with the batch data size of 450, the above mentioned approach suggests the recommended number of cores of 359 cores to achieve the minimum execution time of 7.849 seconds.

By using this method, optimal parameters for executing parallel applications in the HPC system have been determined. This approach eliminates the need to consider all possible cases, such as data precision or core configuration. Consequently, it reduces computational expenses related to the VVUQ framework while boosting the efficiency and reliability of the HPC service.

VI. Conclusion

We have discussed anomaly detection as a crucial challenge in manufacturing settings, and it is imperative that anomalies are detected in near-real-time. The anomaly detection process involves clustering and classifying time-series signals that monitor the position of the linear actuator. The framework was presented, which applies parallelization techniques and an HPC system for real-time anomaly identification. Our framework employed transprecision and approximation computing techniques to provide efficient memory and resource usage. Additionally, the VVUQ method and ML method were introduced to select optimal parameters for parallel application execution. Consequently, IDEKO, as the end user of this HPC service, can effectively detect anomalies in their manufacturing processes.

Acknowledgment

This publication was funded by the European Commission through the Horizon 2020 Program (H2020, call H2020-ICT-2020-2) under grant agreement 101017168.

References

[1] Aivaliotis, Panagiotis, Konstantinos Georgoulias, and George Chryssolouris. "The use of Digital Twin for predictive maintenance in manufacturing." International Journal of Computer Integrated Manufacturing 32, no. 11 (2019): 1067-1080.

[2] Siemens AG. (s.f.). MindSphere Anlyze MyMachine /Condition. https://documentation.mindsphere.io/resources/html/Analyze-MyMachine-condition-opman/en-US/114549568779.html. Accessed on September 18, 2023.

[3] J. Trout. (s.f.). Breaking Down Condition Monitoring. Noria. https://www.reliableplant.com/condition-monitoring-31760. Accessed on September 18, 2023.
[4] Blázquez-García, Ane, Angel Conde, Usue Mori, and Jose A. Lozano. "A review on outlier/anomaly detection in time series data." ACM Computing Surveys (CSUR) 54, no. 3 (2021): 1-33.
[5] Hartigan, John A., and Manchek A. Wong. "Algorithm AS 136: A k-means clustering algorithm." Journal of the royal statistical society. series c (applied statistics) 28, no. 1 (1979): 100-108.
[6] Abanda, Amaia, Usue Mori, and Jose A. Lozano. "A review on distance based time series classification." Data Mining and Knowledge Discovery 33, no. 2 (2019): 378-412.
[7] Trabesinger, Stefan, Andre Butzerin, Daniel Schall, and Rudolf Pichler. "Analysis of high frequency data of a machine tool via edge computing." Procedia Manufacturing 45 (2020): 343-348.
[8] Kretsis, Aristotelis, Panagiotis Kokkinos, Polyzois Soumplis, Juan Jose Vegas Olmos, Marcell Fehér, Márton Sipos, Daniel E. Lucani et al. "Serrano: Transparent application deployment in a secure, accelerated and cognitive cloud continuum." In 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), pp. 55-60. IEEE, 2021.
[9] Ferikoglou, Aggelos, Ioannis Oroutzoglou, Argyris Kokkinis, Dimitrios Danopoulos, Dimosthenis Masouros, Efthymios Chondrogiannis, Aitor Fernández Gómez et al. "Towards efficient HW acceleration in edge-cloud infrastructures: the SERRANO approach." In International Conference on Embedded Computer Systems, pp. 354-367. Cham: Springer International Publishing, 2021.
[10] Berndt, Donald J., and James Clifford. "Using dynamic time warping to find patterns in time series." In Proceedings of the 3rd international conference on knowledge discovery and data mining, pp. 359-370. 1994.
[11] HLRS - High-Performance Computing Center Stuttgart, "Hawk", https://www.hlrs.de/solutions/systems/hpe-apollo-hawk, 2023. Accessed on September 18, 2023.
[12] Jorge, Nocedal, and J. Wright Stephen. Numerical optimization. Spinger, 2006.