PAElla: Edge AI-Based Real-Time Malware Detection in Data Centers

(Article begins on next page)

19 October 2024

pAElla: Edge-AI based Real-Time Malware Detection in Data Centers

Antonio Libri, *Member, IEEE,* Andrea Bartolini, *Member, IEEE,* and Luca Benini, *Fellow, IEEE*

Abstract—The increasing use of Internet-of-Things (IoT) devices for monitoring a wide spectrum of applications, along with the challenges of "big data" streaming support they often require for data analysis, is nowadays pushing for an increased attention to the emerging edge computing paradigm. In particular, smart approaches to manage and analyze data directly on the network edge, are more and more investigated, and Artificial Intelligence (AI) powered edge computing is envisaged to be a promising direction. In this paper, we focus on Data Centers (DCs) and Supercomputers (SCs), where a new generation of high-resolution monitoring systems is being deployed, opening new opportunities for analysis like anomaly detection and security, but introducing new challenges for handling the vast amount of data it produces. In detail, we report on a novel lightweight and scalable approach to increase the security of DCs/SCs, that involves AI-powered edge computing on high-resolution power consumption. The method - called pAElla - targets real-time Malware Detection (MD), it runs on an out-of-band IoT-based monitoring system for DCs/SCs, and involves Power Spectral Density of power measurements, along with AutoEncoders. Results are promising, with an F1-score close to 1, and a False Alarm and Malware Miss rate close to 0%. We compare our method with State-of-the-Art MD techniques and show that, in the context of DCs/SCs, pAElla can cover a wider range of malware, significantly outperforming SoA approaches in terms of accuracy. Moreover, we propose a methodology for online training suitable for DCs/SCs in production, and release open dataset and code.

Index Terms—Artificial Intelligence, Edge Computing, Malware Detection, IoT monitoring, Data Center, Supercomputer.

I. INTRODUCTION

IN the era of the Internet-of-Things (IoT), commodity devices such as low-cost embedded systems can easily compose sensor networks and are increasingly used as monitoring infrastructures, from household appliances to industrial environments [1]–[4]. Depending on the target application, they can produce a massive amount of data that needs to be handled, and to tackle this challenge the emerging edge computing paradigm is receiving a tremendous amount of interest [5]–[7]. Indeed, by pushing data storage and analysis closer to the network edge, this approach allows to mitigate the network traffic load and meet requirements such as high scalability, low latency, and real-time response. However, at its basis, smart techniques to manage and analyze data directly on edge are of crucial importance. With this motivation, emerging

A. Libri and L. Benini are with D-ITET, ETH Zurich, Zurich, Switzerland (e-mail: {a.libri,lbenini}@iis.ee.ethz.ch).

A. Bartolini is with DEI, University of Bologna, Bologna, Italy (e-mail: a.bartolini@unibo.it).

L. Benini is also with DEI, University of Bologna, Bologna, Italy (e-mail: luca.benini@unibo.it).

methods based on Artificial Intelligence (AI) running on edge devices (a.k.a. edge AI or edge intelligence) are envisioned as a promising solution to address this challenge [8], [9].

This is true also in the context of Supercomputers (SCs) and Data Centers (DCs), where a new generation of high-resolution monitoring systems is being deployed [10]–[12], pushing the research boundaries over new opportunities to support their automation, analytics, and control. In particular, DiG [10] is the first out-of-band monitoring system for DCs/SCs that allows real-time edge intelligence on high-resolution measurements, and is already installed in a SC in production (*i.e.*, D.A.V.I.D.E., ranked 18th most efficient SC in the world, based on Green500 November 2017 [13]–[15]). DiG is based on low-cost IoT embedded computers, which are essential to interface with analog power sensors to collect very fine-grain power measurements of the DC servers (*i.e.*, up to 20 μs). Moreover, the system provides real-time on-board processing capabilities, coupled with ultra-precise time stamping (*i.e.*, below microseconds) and a vertical integration in a scalable distributed data analytics framework (*i.e.*, ExaMon [16]).
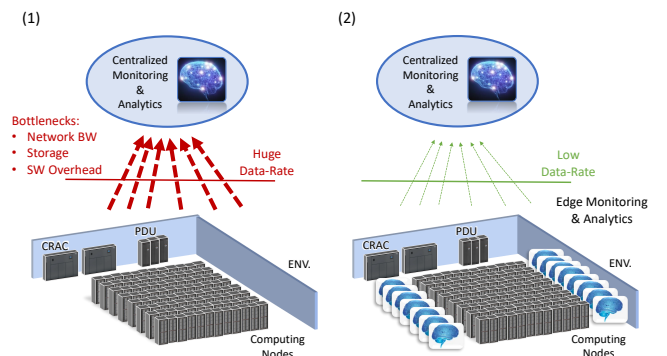
Fig. 1. High-Resolution monitoring bottlenecks in DCs.

These unique features allow us to carry out real-time high-frequency analysis that would not be possible otherwise: with in-band monitoring, like Intel RAPL, the time granularity is limited to the order of millisecond [17], while with current out-of-band monitoring solutions it is not possible to carry out on-board edge analytics [10]. As depicted in Figure 1.1, using an out-of-band monitoring with high-resolution measurements (*i.e.*, below millisecond) and no edge computing capabilities, would result in crucial bottlenecks, such as (i) overhead on the network bandwidth, (ii) overhead on the data storage capacity (to save measurements for post-processing analysis), (iii) and overhead on the software tools that have to handle the measurements (in real-time and offline) [10]. As matter of example, supposing to install a high-resolution power monitoring

system like HAEC [12], with a sampling rate of 500 kS/s (kilo Samples per second) on 4 custom sensors, on a large-scale SC like *Sunway TaihuLight* (3rd in Top500 of June 2019, and involving around 41 thousand compute nodes [18]), it would require a data collection network bandwidth of ~82 GS/s, with obvious overheads on software and storage to handle it. Instead, with a DiG-like monitoring infrastructure (Figure 1.2) we can exploit distributed computing resources to carry out dedicated analysis for each server (server-level analytics), and send at a much lower rate the detected events to a centralized monitoring unit, which has the complete view and holistic knowledge of the whole cluster (cluster-level analytics).

However, to perform real-time high-frequency analysis at the edge, smart data science approaches are essential. In this paper, we show how a novel approach involving AI and Machine Learning (ML) running on IoT monitoring devices, can allow increasing the cybersecurity of DCs, which is nowadays a high impact research topic [19]–[21]. Our edge AI approach targets real-time Malware Detection (MD) for DCs / SCs, and involves feature extraction analysis based on Power Spectral Density (PSD) estimation with the Welch method, along with an AutoEncoder (AE) - which is a particular Neural Network (NN) suitable for Anomaly Detection (AD) - to identify patterns of malware. While similar approaches are already used in the context of speech recognition [22], [23], to the best of our knowledge, this is the first time is used together with high-resolution power measurements for malware - and more in general anomaly - detection. We called the approach *pAElla*, which stands for "Power-AutoEncoder-WeLch for anomaLy and Attacks".

Contributions of the work:

1) A novel out-of-band approach, pAElla, running on IoT-based monitoring systems for real-time MD in DCs. Results in our big dataset (*i.e.*, more than ninety malware + 7 benchmarks, representative of SC and DC activity) report an overall F1-score close to 1, and a False Alarm and Malware Miss rate close to 0 %. We compare our approach with other State-of-the-Art (SoA) ML techniques for MD and show that, in the context of DCs / SCs, pAElla can cover a wider range of malware, outperforming the best SoA method by 24 % in terms of accuracy. In addition, notice that this approach can be used more in general also for AD, in cases where high resolution becomes essential to identify patterns, opening new opportunities for researchers.
2) We propose a methodology for online training in the DC infrastructure suitable to be run in a system in production like DiG.
3) The approach involves zero overhead on the servers processing elements, and it is suitable for large-scale systems, thanks to the on-board ML inference running at the edge, on each compute node.
4) We release both dataset and software we used for the analysis [24]. The dataset includes 7 benchmarks (6 scientific applications for SCs + the signature of the system in idle) and 95 malware of different sort. Notice that this is the first dataset of its kind, providing to the security

research community high-resolution measurements (*i.e.*, power measurements @20 μs + performance measurements @20 ms) for carrying out and benchmarking novel analyses.

Finally, we highlight the fact that pAElla is suitable to be used in a real DC / SC in production, such as in D.A.V.I.D.E., that integrates DiG [13].

Outline: Section II introduces background information on performance counters based MD. Section III reports the related work. Section IV provides an overview of the DiG infrastructure that we use to carry out our analysis. Section V presents both the pAElla algorithm and its implementation on DiG, along with the methodology to run online training in a DC. Finally, we show in Section VI how we built the dataset for our analysis, together with the results of the MD and a comparison with SoA. We also report an analysis of the overheads in running pAElla on DiG, and some considerations on its scalability to large-scale DCs. We conclude the paper in Section VII.

II. BACKGROUND ON MALWARE DETECTION

The threat of malware and more in general cyber attacks is nowadays considerably increasing, and countermeasures to detect them are becoming more critical than ever. Examples spread from different kinds of backdoors, to trojans and ransomware, each of them having different Operating System (OS) and computational usage characteristics [25]. A prominent SoA technique for MD - and more in general for AD - is based on catching their dynamic micro-architectural execution patterns, employing monitoring the hardware performance counters available in modern processors [25]–[29]. By using this approach, we can benefit of (i) a lower overhead than using higher-level approaches - *e.g.*, monitoring patterns at application and OS level -, and (ii) also of the fact that these micro-architectural features are efficient to audit and harder for attackers to control, when they want to avoid detection in evasion attacks [25].

There are mainly two ways to train the anomaly-based detector to catch malware: (i) training on malware signatures [26], or (ii) training on "healthy" operating conditions of the system that we want to protect by malware attacks, to be able to identify possible changes that indicate an infection [25]. While the first method is a supervised learning approach, the second one can be seen as "semi-supervised" learning as we do not need any prior knowledge of the malware signatures, thus we can detect even zero-days attacks (*i.e.*, attacks which are never seen before) [25], [30]. The idea behind the second method can also be applied within the context of DCs and SCs: the supercomputing center can study the "regular" activity of their users, and thus create models on healthy machine states.

Drawbacks:

- it is not possible to monitor all the performance counters at the same time (*e.g.*, Intel allows the selection of only 4 performance counter at a time via the Performance Monitoring Unit - PMU -, which can become 8 by disabling the Hyperthreading [31]). This does not allow

to be flexible, as some malware are better described by certain performance counters rather than others, with a consequent degradation on the malware accuracy detection (we do not know a priori which malware is running, so which performance counter is better to select).

- Depending on the running malware, the time granularity of the monitoring becomes of primary importance to be able to catch their micro-architectural execution patterns [25]. However, as described in the Introduction Section, current SoA built-in tools to monitor performance counters are limited by sampling rate and edge computing capabilities.

As we show in our results, these drawbacks can be a limitation to discover malware in real-time in a DC.

Our Approach: To bridge this gap, we propose a solution that exploits the out-of-band and very high resolution power consumption measurements of DiG (no performance counters involved), in a lightweight algorithm that can run in real-time on the IoT devices at the edge of the DC. This method allows catching fine-grain activities of the malware that are not visible with the performance counter based approach.

III. RELATED WORK

Perf-Counters-based MD: In recent years, several works in the literature focused on the usage of performance counters for Anomaly and Malware Detection. In particular, [26] showed the feasibility of the technique by testing them with several supervised learning approaches (*i.e.*, k-Nearest Neighbors, decision tree, random forest, and Artificial Neural Networks) after training models on known malware. Later, [25] showed the feasibility of this method with an unsupervised learning approach, namely the one-class Support Vector Machine (oc-SVM), by training the model on healthy programs. Moreover, they suggested a list of performance counters that can, on average, better describe the malware signatures (we use this list - reported in Table I - to compare this method with our approach). In this direction, different methods were proposed to improve the technique, not only for MD, but also more in general for AD. In the context of DCs, works in [28]–[30], [32] showed how to use performance counters to detect anomalies, while [33] showed how to use them for detecting covert cryptocurrency mining. However, very recent works in [34], [35] bring into question the robustness of this technique for security, carrying out a study with a big dataset with more than ninety malware and reporting poor results in accuracy detection.

Power-based MD: Similar to the performance-counters-based approach, several works in literature proposed methods to detect malicious activity in IoT networks by mean of energy consumption footprint. To provide some examples, [36] presented a machine learning based approach to detect ransomware attacks by monitoring the power consumption of Android devices, or [37] proposed a cloud-assisted framework to defend connected vehicles against malware. While all these works targeted the security of embedded systems, a proof of concept toward the feasibility of power-based malware detection for general-purpose computers was proposed in [38].

In this paper, Bridges et al. use an unsupervised one-class anomaly detection ensemble, based on statistical features (*e.g.*, mean, variance, skewness and Kurtosis), with a power consumption sampling rate of 17 ms. However, as they claim in their paper, further research to increase the sampling rate is necessary, especially for accurate baselining of very small instructions sequences of malware. Moreover, although this was the first step on this kind of approach, to the best of our knowledge: (i) no other works proved yet the feasibility in a DC where embedded monitoring networks are involved, and edge intelligence for real-time analytics is required; (ii) there is not yet a robust dataset with several malware that proves the robustness of the technique.

Data Center Monitoring: Off-the-shelf methods to measure power and performance of DC compute nodes rely on in-band or out-of-band telemetry depending on the technology vendors. In particular, an example of in-band solution is Intel RAPL [17], which can provide a time granularity up to 1 ms, while examples of out-of-band solutions are IBM Amester [39] and IPMI [40], which can provide a resolutions of 250 μs and 1 s, respectively, but not AI-powered edge computing. Towards high-resolution monitoring of DCs, several works in recent years proposed solutions, such as [12], [41], and DiG [10] is the best-in-class SoA infrastructure that bridges the gap of very fine-grain monitoring (*i.e.*, 20 μs of time granularity) and edge computing.

PSD and NNs: While NNs, and more in particular AEs, are already used for anomaly and malware detection in different domains, including DCs [30], [42], the use of Fourier analysis together with NNs is a well-known procedure in speech recognition [22], [23]. We use a similar approach for malware detection in our study, involving PSD with Welch method, and an AE for ML inference on-board. To the best of our knowledge, this is the first work proving its feasibility for the cybersecurity of DCs.

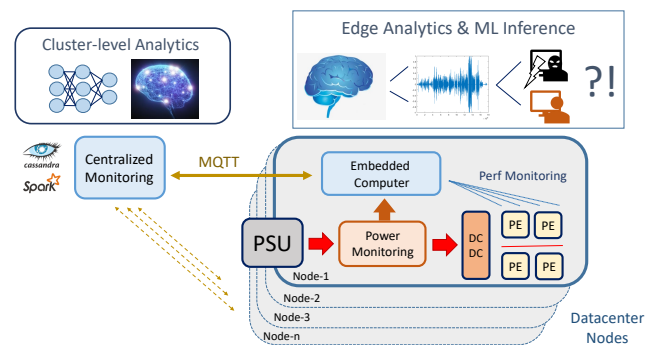## IV. DiG & AD ON THE EDGE OF A TOP500'S SUPERCOMPUTER

Fig. 2. Sketch of the DiG architecture.

DiG [10] is the first monitoring system for DCs/SCs already in production, that allows (i) very high frequency monitoring (*i.e.*, up to 20 μs), (ii) real-time on-board processing for AI on the edge of the cluster (*e.g.*, feature extraction

algorithms and ML inference), (iii) ultra-precise time stamping (*i.e.*, sub-microseconds synchronizations - below the sampling period - between the cluster nodes) and (iv) vertical integration in a scalable distributed data analytics framework (*i.e.*, the open source ExaMon system [16]). The monitoring infrastructure is completely out-of-band, scalable, and low cost. Moreover, DiG is technology agnostic - it is already installed on several HPC systems, based on different CPU architectures (*i.e.*, ARM, Intel, and IBM).

As depicted in Figure 2, the monitoring system architecture involves:

- a power sensor, which is placed on each DC compute node, between the power supply unit and the DC-DC (Direct Current to Direct Current) converters that provide power for all the processing elements (PE) / electrical components within the node. The power sensing is based on a current mirror and shunt resistor to measure the current, and on a voltage divider to measure the voltage. We use it to acquire SoA high-resolution power measurements, with a time granularity of 20 μs and a precision below 1 % (σ);
- an IoT embedded computer (*i.e.*, BeagleBone Black - BBB) dedicated for each compute node, that we use to carry out server-level analytics. The system is interfaced with the power sensor via a 12-bit 8-channels SAR ADC, and with existing in-band / out-of-band telemetries to collect hardware performance counters (*e.g.*, Amester [39], IPMI [40], and RAPL [17]). Moreover, it includes (i) hardware support for the Precision Time Protocol (PTP), which allows sub-microsecond measurements synchronization [43], [44], (ii) two Programmable Real-Time Units (PRU0 and PRU1), that we exploit for real-time feature extraction on-board, and (iii) an ARM Cortex-A8 processor with NEON technology, useful for DSP processing and edge ML inference (*e.g.*, by leveraging the ARM NN SDK [45], which enables efficient translation of existing NN frameworks - such as TensorFlow - to ARM Cortex-A CPUs);
- a scalable distributed data analytics framework, namely ExaMon [15], [16], that we use to collect at a lower rate - from seconds to milliseconds - power and performance activity of the all cluster and thus to carry out cluster-level analytics. To send data from the distributed monitoring agents (*i.e.*, daemons running on the BBBs) to the centralized monitoring unit, we adopted MQTT [46], which is a robust, lightweight and scalable publish-subscribe protocol, already used for large-scale systems both in industry and academia (*e.g.*, Amazon, Facebook, [46], [47]). Moreover, we use Apache Cassandra [48] to store data in a scalable database and exploit Apache Spark [49] as Big Data engine to perform cluster-level ML analytics both in streaming and batch mode.

DiG provides a flexible platform to collect fine-grain measurements of the compute nodes activity, and prototype novel anomaly detection methods. Furthermore, DiG allows to test new strategies for data-science on-board, to handle both the considerable amount of data that otherwise would impact on the communication network and the real-time processing.

V. *pAElla*: ALGORITHM & IMPLEMENTATION

In this section, we explain the different phases of the pAElla approach, namely (i) real-time Feature Extraction and (ii) on-board MD inference, and their implementation in the DiG infrastructure. Furthermore, we propose a methodology for online training in a DC.

A. Edge Real-Time Feature Extraction

For the Feature Extraction phase, we use the PSD with the Welch method [50]. This technique is based on the periodogram method, with the difference that the estimated power spectrum contains less noise, but with a penalty in the frequency resolution. In particular, the signal is split up into overlapping segments, which are then windowed and used to compute periodograms with the Fast Fourier Transform (FFT). Finally, it is computed the squared magnitude of the result, and the individual periodograms are averaged. We selected it over other approaches - *e.g.*, Discrete Wavelet Transform (DWT) or Wavelet Packet Transform (WPT) - for (i) its capability to unveiling relevant frequency components in all the signal bandwidth (w.r.t. DWT and WPT, which instead provide temporal information at a price of a lower frequency resolution) and (ii) for its low computational complexity [50]. Moreover, PSDs already proved to be a robust feature extraction method to be used together with NNs, in applications such as speech recognition [22], [23].

To find a good time window to use for the FFTs, we run on the monitored compute node a synthetic benchmark that generates a known pattern, and we monitored it with DiG. In particular, we use a pulse train of instructions at 1 kHz, where we alternate high load computational phases with idle phases. Figure 3 shows the results of the PSD analysis in a time window of ~164 ms (*i.e.*, 8192 data samples), and different lengths of the FFTs, namely 4096, 2048 and 1024 points (*i.e.*, ~82 ms, ~41 ms and ~20.5 ms, respectively). Except for the one at 1024 samples, the other two can detect with an approximately good precision the main peak at 1 kHz (magnitude greater than 5 dB), plus its harmonics in all the signal bandwidth 0–25 kHz.

To respect the real-time constraints in the DiG IoT devices, we selected the PSD with FFTs at 2048 samples, which correspond to an output of 1025 data samples - *i.e.*, $(FFT\_samples/2) + 1$ - that we use as input features for the phase of MD inference. Finally, we compute consecutive PSDs with sliding windows of 20 ms (1000 power samples). As shown in Section VI, this decision proved to be a good choice in terms of both (i) MD accuracy and (ii) to create a consistent dataset to compare with SoA MD techniques. However, future works can investigate the results of the MD inference also with different lengths of the PSD, FFTs, and sliding windows, to deploy pAElla in other kinds of IoT devices, with different memory / processing requirements.
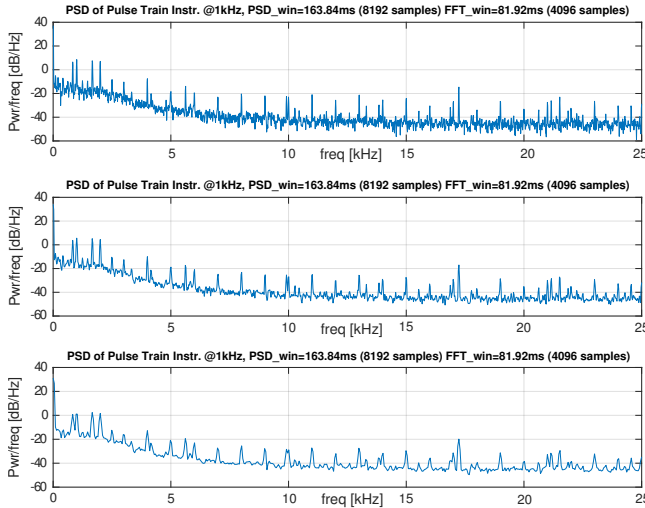
Fig. 3. Comparison between PSDs of a Pulse Train of instructions at 1 kHz for different FFT lengths.

B. Edge Malware Detection Inference

For the ML inference phase, we use an AE, which is a particular kind of Neural Network suitable for Anomaly Detection [30], [42]. As described in Section II, the idea is to train a model on "healthy" activity of the DC (*i.e.*, with no malware involved), and try to identify possible anomalies in these signatures when a malware is running in background. With an AE we can do exactly this. By learning to replicate the most salient features in the training data, the AE tries to reconstruct its input x on its output y. When facing anomalies, the model it worsens its reconstruction performance.

In particular, the AE consists of one or more internal (hidden) layers H, and involves mainly 2 phases: an encoding phase $H = enc(x)$, and a decoding phase that tries to reconstruct the input $y = dec(H)$ under certain constraints (*e.g.*, the dimension of the hidden layers can be smaller than the dimension of the input). These constraints allow to not just simply learn how to reconstruct a perfect copy of the input, namely the identity function $dec(enc(x)) = x$, but instead to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x, and thus to discover interesting structures about the data. The difference between input and output is the reconstruction error E_r, and we can use it as an anomaly score to detect malware. Indeed, after training, the AE can reconstruct healthy data very well (*i.e.*, E_r is small and comparable to the error at training time), while failing to do so with anomalous signatures that the AE has not encountered yet (*i.e.*, large E_r w.r.t. the training error).

After an empirical evaluation, we chose a network with 4 fully-connected layers (3 hidden layers plus the output layer), where we implement the phases of encoding-encoding-decoding-decoding, respectively. As shown in our results, this network proved to be a good option in terms of accuracy and computational demands, especially for training and inference time. In particular, the 3 hidden layers employ {8,4,4} neurons, while - as in every AE - the output layer has the same dimension of the input, that in our case corresponds to 1025

features (*i.e.*, PSD dimension). As regularization term, we employ the L1-norm [51], while as activation functions for the neurons we use the hyperbolic tangent (tanh) and the rectified linear unit (relu), in the sequence {tanh-relu-relu-tanh} for the 4 layers, respectively. Moreover, before processing the PSDs as input features, we pre-process them, by removing the mean and scaling to unit variance.

Lastly, we adopted a threshold-based method, with 2 thresholds, to distinguish between malware and healthy states. The first threshold, namely T_E, is related to the reconstruction error and allows us to understand if a PSD is an outlier. To compute E_r, we use the Mean Square Error (MSE) between input and output of the AE. Then we tag as anomalies all the PSDs where E_r is greater than T_E. The second threshold, T_O, is related to the percentage of outliers detected. If this percentage is greater than T_O, we detected a potential malware running in the server, and thus we can raise an alarm for the system admin of the DC.

C. Algorithm Implementation in the DiG IoT Devices

The on-board implementation of the pAElla algorithm involves mainly 3 phases: (i) real-time acquisition of the power consumption measurements, that we carry out with the PRU, (ii) real-time computation of the PSD, that we perform in floating-point on the ARM processor, and (iii) real-time MD inference, where we exploit the NEON SIMD architecture extension. Figure 4 highlights their implementation in the BBB.
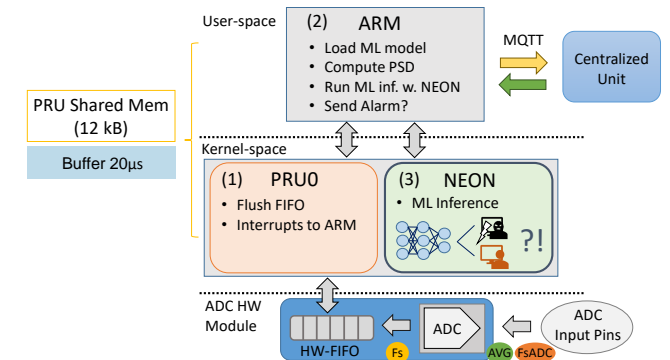
Fig. 4. Implementation of pAElla in the IoT embedded systems of DiG.

Phase1: The bottom layer represents the ADC hardware module. We exploit the ADC continuous sampling mode to continuously sample the two input channels (*i.e.*, current and voltage), and store them in a hardware FIFO (HW-FIFO). By tuning the ADC sampling frequency (FsADC) and the hardware averaging (AVG), it is possible to set the frequency (Fs) at which samples enter the software layers. The best trade-off corresponds to a sampling rate (FsADC) of 800 kS/s per channel - maximum rate when using two channels - and hardware average every 16 samples. This is equivalent to 50 kS/s (*i.e.*, Fs equals to 20 µs) and allows to (i) cover the entire signal bandwidth of the server power consumption, and (ii) reach a measurement precision below 1 % (σ) of uncertainty (a.k.a. oversampling and averaging method [52]).

When the HW-FIFO reaches a watermark on the number of samples acquired (we set it to 32), an interrupt is raised and the samples are flushed by the PRU0 into the PRU Shared memory (12 kB).

Phase2: When a given number of samples is collected (we set it to 2048 samples, which correspond to 4 kB in memory - 2 B each), the PRU0 raises an interrupt to the ARM through a message passing protocol named RPMsg. The ARM can then access the PRU Shared Memory to read the samples and store them in RAM. As soon as it reaches a time window of 8192 samples (163.84 ms) it uses the samples to compute the PSD. In this way, we can achieve 100 % hard real-time demand by not losing power samples, and completely offload the data acquisition phase to the PRU, making the ARM processor available for other tasks, such as computing the PSD and carrying out ML inference. Moreover, by computing the PSD in the ARM processor, we can exploit the Floating Point Unit (which is not available in the PRU) to avoid losing precision when computing the PSD.

Phase3: When the PSD is calculated, the ARM stores it in RAM. After the collection of a batch of PSDs, we exploit the NEON SIMD architecture extension of the ARM to run MD inference. In particular, depending on the activity is currently running in the DC (*e.g.*, system in idle, or application X), the centralized monitoring unit - which has the complete knowledge of the status of the cluster - communicate to the ARM which ML model to load for inference. At this point, if a malware is detected, we send an alarm to the system admins of the DC.

D. Methodology for Online Training

For the training phase, we propose a methodology that is suitable to run online on DCs / SCs in production, which integrate a DiG-like monitoring infrastructure (*e.g.*, D.A.V.I.D.E.). It is noteworthy that this approach can be used more in general also for AD, in cases where high-resolution analysis plays a key role in identifying anomalous patterns. In particular, we take acquisitions during healthy-states of the DC. We compute for these acquisitions the PSDs on DiG, and pass these data to the corresponding compute nodes for training the AE models (*i.e.*, each node trains its AE - for example with GPUs).

This distributed approach allows to (i) reduce both data pre-processing and training time, along with the amount of data to be communicated on the network if using instead a centralized method (*e.g.*, pre-processing and training on the centralized monitoring unit); (ii) scale to large DCs / SCs. Moreover, the training time and overhead are not a critical concern for the servers, since (i) the training phase takes place only at the beginning, and then at a very low rate, and mostly depending on new applications - or substantially modified versions of previously trained applications - that are running; and (ii) this activity can be scheduled during maintenance periods.

After some experiments, we use for training in our tests the Adagrad algorithm [53], with MSE as loss function, a batch size equal to 8, and 5 epochs. After the training phase, the ML models are loaded on the embedded monitoring boards, ready for inference on new data. Notice that each AE is trained

with data coming from its corresponding node. This allows to take into account specific activity related to that node, such as particular programs installed and running in background, but also possible differences in the hardware (*e.g.*, large variations of material properties when moving to 10–7 nm chips [54]).

VI. EXPERIMENTAL EVALUATION

In this chapter, we report the results of the experimental evaluation carried out with our approach to detect malware - and anomalies - in a DC / SC, and a comparison with SoA techniques that exploit performance counters. We conclude the section by reporting on the performance of the pAElla algorithm running in the DiG infrastructure.

A. Building Dataset for Analysis and Comparison with SoA

To create a robust dataset of malware, for testing our approach and compare it with SoA techniques, we downloaded a collection of 95 malware from *virusshare.com*, which is a repository of malware samples available for security research. The collected malware are of any kind, spreading from different type of backdoors, trojans, and ransomware. To emulate a normal activity of the cluster, we acquired signatures of 7 benchmarks, namely the system in idle, plus 6 scientific applications which are widely used for parallel computing, such as Quantum Espresso (QE) [55], HPLinpack (HPL) [56], High Performance Conjugate Gradients (HPCG) [57], Gromacs [58] and NAS Parallel Benchmarks (NPB) [59], that we use in 2 versions, the one that exploits 9 cores (NPB_btC9) and the one that uses 16 cores (NPB_btC16).

For safety reasons, we run the tests on a compute node that is completely isolated from the rest of the network available to the users of our facilities. The node is an Intel Xeon E5-2600 v3 (Haswell). To compare pAElla with SoA techniques, we collected for each acquisition both (i) hardware performance counters and (ii) PSDs computed from the high-resolution power measurements. In particular, we analyzed the performance counters suggested by Tang et al. [25], plus further metrics available with our open source monitoring tool, ExaMon [16]. We summarize a list of the selected metrics in Table I (per-core metrics) and Table II (per-cpu metrics). Each table shows the "raw" metrics, which are the ones we can directly select and monitor, and "derived" metrics, which are the ones we compute with ExaMon on top of the raw metrics (*e.g.*, per-core load, IPS, DRAM energy). As reported in Section II, the PMU of Haswell Intel processors allows to sample a maximum of only 8 performance counters simultaneously. For this reason, we collected all the listed performance counters in multiple runs of the benchmarks / malware.

For the performance metrics, we use a time granularity of 20 ms *i.e.*, in line with SoA DCs / SCs monitoring tools [15]), dedicating 2 cores for ExaMon to prevent that the in-band monitoring can affect the measurements with noise. For this purpose, we exploit *isolcpus*, which is a kernel parameter that can be set from the boot loader configurations. Instead, for the high-resolution power measurements, we can benefit of a time granularity of 20 μs, thanks to the out-of-band monitoring of DiG that allows to do not subtract computing resources from

TABLE I
SELECTED PER-CORE PERF COUNTERS BASING ON SOA [25]

Raw Metrics

UOPS_RETIRED (Retired Micro-ops)
ICACHE (Instruction Cache misses)
LONGEST_LAT_CACHE (Core cacheable demand req. missed L3)
MEM_LOAD_UOPS_L3_HIT_RETIRED (Retired load uops in L3)
BR_MISP_RETIRED (Mispredicted branch instr. retired)
UOPS_ISSUED (Uops issued to Reservation Station - RS)
IDQ_UOPS_NOT_DELIVERED (Uops not deliv. to RAT per thread)
INT_MISC (Core Cycles the allocator was stalled)
BR_INST_RETIRED.NEAR_RETURN (Instr. retired direct near calls)
BR_INST_RETIRED.NEAR_CALL (Direct and indirect near call ret.)
BR_INST_EXEC.ALL_DIRECT_NEAR_CALL (Retired direct calls)
BR_INST_EXEC.TAKEN_INDIRECT_NEAR_CALL (Ret. indir. calls)
DTLB_STORE_MISSES.STLB_HIT (Store operations miss first TLB)
ARITH.DIVIDER_UOPS (Any uop executed by the Divider)
DTLB_LOAD_MISSES.STLB_HIT (Load operations miss first DTLB)
MEM_LOAD_UOPS_RETIRED.L3_MISS (Miss in last-level L3)
MEM_LOAD_UOPS_RETIRED.L2_MISS (Miss in mid-level L2)
MEM_LOAD_UOPS_RETIRED.L1_MISS (Ret. load uops miss in L1)
BR_MISP_EXEC.ALL_BRANCHES (Retired mispr conditional branch)
BR_MISP_EXEC.TAKEN_RETURN_NEAR (Retired mispr. indir. br.)
C3 (Clock cycles in C3 state)
C3res (C3 residency - Clock cycl. in C3 state between 2 sampling time)
C6 (Clock cycles in C6 state)
C6res (C6 residency - Clock cycl. in C6 state between 2 sampling time)
temp (Cores temperature)

Derived Metrics

load_core (per-core load)
IPS (Instructions per Second)
CPI (Cycles per Instruction)

Note. More info about the metrics are available in [16], [25], [60].

TABLE II
SELECTED PER-CPU PERF COUNTERS

Raw Metrics

C2 (Clock cycles in C2 state)
C3 (Clock cycles in C3 state)
C6 (Clock cycles in C6 state)
C2res (C2 residency)
C3res (C3 residency)
C6res (C6 residency)
freq_ref (Core frequency)
erg_units (Energy units)
temp_pkg (Package temperature)

Derived Metrics

erg_dram (Energy DRAM consumed)
erg_pkg (Package Energy consumed)

Note. More info in [16], [60].

users. In particular, we compute the PSDs in time windows of 163.84 ms (8192 samples), with FFT of 2048 points, and sliding windows of 1000 samples between two consecutive PSDs. Notice that the sliding window corresponds to 20 ms, which is consistent with the granularity of the performance counters. In this way, we can construct a dataset that has for each row the different samples acquired over time, and for each column the several features (*i.e.*, performance counters and PSDs). Moreover, to show that simple coarse grain statistics of the power measurements are not enough to detect malware,

we include as features also standard deviation, mean, max, and min value in the interval of 20 ms.

To build the dataset, we acquired first signatures of healthy benchmarks (normal activity of the DC with no malware involved), and then signatures of the same benchmarks but with malware in background. In particular, we collected 30 acquisitions for each benchmark and 2 acquisitions for each of the 95 malware running in background with every benchmark. Notice that for each run of the malware, we use a completely fresh installation of the OS, to prevent that different malicious activities can interfere and thus invalidate the analysis. We release both dataset and software to analyze the data open source, at the following link [24].

B. Malware Detection Results

To evaluate the performance of our approach, and compare it with SoA techniques for MD, we tested the acquisitions in our dataset with both performance metrics and PSD features, and with different ML algorithms suitable for Anomaly and Malware Detection, namely oc-SVM [25], Isolation Forest (IF) [61] and Autoencoders. For oc-SVM and IF, we use the Scikit-learn [62] implementation developed in python, while for the AE we use Keras [63] with TensorFlow [64] as back-end.

After an empirical evaluation, we adopted standardization (Scikit-learn *StandardScaler*) like in pAElla (settings described in Section V-B), to pre-process the features also for oc-SVM and IF. Moreover, for the oc-SVM we use the Principal Component Analysis (PCA) to reduce the dimension of the feature space to 25 components (a.k.a. dimensionality reduction), and the polynomial kernel with 0.1 as kernel coefficient (*i.e.*, "gamma" in the Scikit-learn API). Instead, for IF we obtain better results without PCA, and with the contamination parameter set to "auto" (*i.e.*, decision function threshold determined as in [61]). Finally, as in pAElla, we use for oc-SVM and IF a threshold-based method, but in their case with only one threshold on the percentage of outliers found in the benchmark, that we set to 30 % (*i.e.*, if the number of anomalous samples is above this threshold, we label it as a malware).

To train the ML models (1 model for each benchmark), we use only healthy benchmarks, while for validation (useful to find proper settings for our models without incurring to overfitting) and test, we use both healthy and malicious signatures. In particular, we split the healthy benchmarks subset to 60 % for training, 20 % for validation and 20 % for test, while we split the malicious subset to 50 % for validation and 50 % for test. Notice also that together with the analysis of the performance counters, we include the coarse-grain power statistics at 20 ms.

Table III reports the results of our analysis in the test set. Namely, the (i) False Alarm (FA) rate (= False Positive - FP - rate) - *i.e.*, healthy benchmarks erroneously labeled as malware; (ii) the Malware Miss (MM) rate (= False Negative - FN - rate) - *i.e.*, malware not detected; and (iii) the weighted F1-score [28] (best value at 1, and worst at 0), which measure the test accuracy based on the following formula:

TABLE III
FINAL RESULTS OF THE MD ANALYSIS AND COMPARISON WITH SoA APPROACHES

ML Approach	Benchmark	Perf Metrics + Power Statistics @20ms			PSD of Power @20us		
		False Alarm	Malware Miss	F1-score	False Alarm	Malware Miss	F1-score
oc-SVM	Idle	1.000	0.042	0.645	1.000	0.000	0.664
	QE	1.000	0.011	0.660	0.000	0.221	0.876
	HPL	1.000	0.095	0.621	0.167	0.968	0.053
	HPCG	1.000	0.600	0.332	0.167	0.032	0.906
	Gromacs	1.000	0.074	0.631	0.000	0.032	0.984
	NPB_btC9	1.000	0.853	0.137	0.000	1.000	0.000
	NPB_btC16	1.000	0.884	0.109	0.000	0.042	0.978
	Overall	1.000	0.365	0.480	0.190	0.328	0.721
IF	Idle	0.000	0.053	0.973	0.000	0.979	0.041
	QE	0.000	0.137	0.927	0.000	0.968	0.061
	HPL	0.000	0.642	0.527	0.000	0.979	0.041
	HPCG	0.000	0.000	1.000	0.000	1.000	0.000
	Gromacs	0.000	0.874	0.224	0.000	1.000	0.000
	NPB_btC9	1.000	0.000	0.664	0.000	1.000	0.000
	NPB_btC16	0.667	0.000	0.748	0.000	0.958	0.081
	Overall	0.238	0.244	0.758	0.000	0.983	0.033
AE	Idle	0.000	0.579	0.593	**0.000**	**0.032**	**0.984**
	QE	0.000	0.937	0.119	**0.000**	**0.000**	**1.000**
	HPL	0.000	0.947	0.100	**0.000**	**0.000**	**1.000**
	HPCG	0.167	0.042	0.901	**0.000**	**0.000**	**1.000**
	Gromacs	0.000	0.947	0.100	**0.000**	**0.000**	**1.000**
	NPB_btC9	0.000	0.084	0.956	**0.000**	**0.000**	**1.000**
	NPB_btC16	0.000	0.189	0.895	**0.000**	**0.000**	**1.000**
	Overall	0.024	0.532	0.627	**0.000**	**0.005**	**0.998**

Note. For best results we want False Alarm rate (healthy benchmarks seen as malware) and Malware Miss rate (malware not detected) close to 0, while F1-score close to 1. Results related to pAElla are highlighted in bold.

$$F1score = \frac{2TP \cdot W_M}{2TP \cdot W_M + FP \cdot W_H + FN \cdot W_M} \quad (1)$$

where we weighted the True Positives (TP), FP, and FN by the number of instances of each class (Malware and Healthy), via the two weights W_M and W_H, to take into account the imbalance of the dataset between number of malware and number of healthy acquisitions. Indeed, if we do not consider a weighted F1-score, then a naive classifier that marks every sample as malware would achieve an overall F1-score of 0.95, as approximately 95 % of our dataset consists of malware acquisitions. Moreover, notice that, as we are focusing on the detection of malware, it is really important that the MM rate is close to zero, and of course, also that the FA rate is reasonably low, to avoid too many false alarms to handle, and the F1-score is close to 1 (best accuracy).

Notice that, to the best of our knowledge, this is the first work based on fine grain monitoring of power and performance, that targets SCs and DC compute nodes, along with their requirements (*e.g.*, scalability, and reasonable overhead for in-band monitoring, to do not impact computing resources), and that reports a comprehensive analysis with a vast number of malware. When comparing with the analysis via performance counters, in line with other SoA works in literature targeting anomaly detection in SCs [28], [65], we obtain superior results when using IF and AE, rather than oc-SVM. In particular, in our experience the main problem with oc-SVM is that the feature space is not well separable for

performance counters, and thus it is difficult to find a good tuning (*e.g.*, kernel coefficient, PCA components, etc.) and set a proper threshold to distinguish between normal activity and malware (*i.e.*, both healthy and malicious signatures are always seen as anomaly, leading to a FA rate of 1 and an overall F1-score of 0.48).

Instead, when using IF with performance metrics (+ coarse-grain power statistics) we observed a high F1-score, with 0 % of FA rate, in the system in idle, HPCG, and QE, while we obtained a poor F1-score for Gromacs and HPL, and a really low F1-score with high FA rate for NPB_btC9, and NPB_btC16. When looking at the overall F1-score, we obtain a low value of 0.758, which is in line with results in [34], [35], when using tree-based models with performance counters for malware detection (reason why they do not advice this technique for security purposes). Finally, in the case of AE, we obtain poor scores for all benchmarks, except for HPCG, NPB_btC9, and NPB_btC16, and an overall F1-score of 0.627.

To understand which ML algorithm performs better with PSDs of fine-grain power measurements as input features, we tested all previous methods, and report the results in the three rightmost columns of Table III. In particular, when using oc-SVM we can find slightly better outcomes than using the same approach with performance counters, with an overall F1-score od 0.721 and a FA rate of 0.19. In particular, we observed a good F1-score with 0 % of FA rate for Gromacs and NPB_btC16 (0.978 and 0.984, respectively), while slightly worse results for QE and HPCG (where the model was not able

to detect all healthy benchmarks). We obtained then a poor F1-score with HPL and NPB_btC9, and even did not find a good threshold for the system in idle (*i.e.*, the model always sees everything as malware). In the case of IF, the results drop to really poor performance, with an overall F1-score close to zero as the MM rate is close to 1 (*i.e.*, the model struggles to distinguish between the malicious and healthy patterns).

Lastly, the combination AEs + PSDs (*i.e.*, pAElla method, highlighted in bold in the table) shows promising results. In particular, we found that the reconstruction error is definitely larger during anomalous periods compared to the normal ones, and thus by setting proper thresholds (*e.g.*, 0.91 for the reconstruction error, and 30 % for the quantity of anomalous samples in the benchmarks) we can obtain an F1-score equal to 1 for almost all benchmarks (overall F1-score equal to 0.998), and both a FA and MM rate close to 0 %. Comparing these scores with previous results, pAElla outperforms the best tested SoA method (IF) with an improvement in accuracy of 24 %. As the pAElla approach is completely out-of-band (zero overhead on the computing resources) and simple to implement in off-the-shelf low-cost IoT devices, we believe it can push the boundaries of security in DCs to new levels.

C. Computational Load & Scalability

With respect to the computational load, pAElla is a very lightweight approach. Thanks to the PRU offloading, we can run the whole feature extraction phase in the DiG IoT devices with less than 1 % of the ARM CPU (used to transfer data from the PRU shared memory into the main RAM). Moreover, we can compute PSDs in the ARM - length of 8129 samples, FFTs of 2048 samples, and sliding window of 1000 samples (*i.e.*, 20 ms) - within ~19 ms, respecting the real-time constraint of the PSD sliding window. It is noteworthy that we used not optimized code for the implementation of the PSD (*i.e.*, Welch method in python), which means we can further improve this performance. Then, we can run edge ML inference in batch of 500 PSDs (*i.e.*, every 10 s, considering a sliding window of 20 ms), exploiting the NEON with Keras and TensorFlow, in less than 0.8 s. Future works can study the performance of lightweight frameworks, such as TensorFlow Light [66] or ARM NN [45], and also the accuracy of pAElla with different length of PSD, FFTs and sliding windows. Finally, the training phase took for us around 36 s on a DC server equipped with a GeForce GTX 1080 NVIDIA (GPU usage between 14–15 %), and with code written in Keras plus TensorFlow for GPU as a back-end.

We highlight also that pAElla is a highly scalable AI approach, thanks to the edge computing paradigm. As a matter of example, supposing to carrying out the approach without edge computing in a SoA large-scale system, such as Sunway TaihuLight (~41 thousand compute nodes [18]), we should handle a rate of ~20.5 GB/s (*i.e.*, 41 k nodes $\times$ 50 kS/s per node $\times$ 10 B for each sample, which includes raw power and timestamp). Considering that it would be impossible to analyze this amount of data offline (the database capacity would reach its limit in few minutes), and that for real-time computing with only a centralized monitoring unit several bottlenecks would arise (*e.g.*, high-latency, high SW overhead to handle the data, high network traffic burden, and also the possibility to loose samples), edge AI is the right direction to face this challenge. With its design, pAElla requires only few megabytes for training and a few bytes to send alarms when potential malware are detected.

Moreover, we underline the fact that using performance metrics, like in SoA techniques, but with a high monitoring rate to detect malware, would be hard in Top 500 clusters, like Sunway TaihuLight. Indeed, just as matter of example, supposing to collect all 25 per-core performance metrics reported in Table I at the granularity of 20 μs, and considering that Sunway TaihuLight integrates 260 cores per CPU [18], this would result to a rate of 325 MS/s to be handled either in-band in the compute servers with live analysis (with the consequent overhead in the computing resources), or out-of-band by sending these data to a centralized monitoring unit, with the consequent overhead in the network infrastructure (*i.e.*, roughly 13.3 TS/s).

Finally, an interesting extension of our approach is to use open-source distributed stream processing systems, like Storm, Flink, or Spark Streaming [67]–[69] to carry our real-time analytics. As shown in several works in literature (*e.g.*, [70]–[72]), the common use-case exploits a distributed messaging system, like Apache Kafka [73], that runs on the IoT devices and sends data to dedicated servers hosting the streaming analytics software. These servers carry out the distributed real-time analytics services, with frameworks such as Storm. As an example, as reported in [74], Storm can handle a data rate of 1 MS/s per node with messages of 100 B on servers that include 2 Intel Xeon E5645, running at 2.4 GHz and with 24 GB of memory. In other words, at the cost of using additional dedicated servers it is possible to benefit from a robust and flexible approach for training, and also for maintenance, deployment, and update of the streaming service on the edge. Of course, drawbacks in our case are that (i) to do not overwhelm the data network, the processing servers have to be as close as possible to the data source (*i.e.*, BBB), and (ii) the cost of the extra servers for distributed processing would grow with the size of the data center.

Other possibilities would be (i) to run pAElla on the DiG IoT devices as we do, and then use these distributed systems only for training and deployment of the ML models; or (ii) execute these stream services directly on the IoT devices. However, as shown in [67], which benchmarks them in a Raspberry Pi 3 (ARMv8 Quad-Core 1 GB RAM vs. ARM A8 Single-Core 1 GB RAM in our BBB), the resources needed (memory and CPU load) for running these services are not negligible, even considering only data ingestion and delivery and not considering actual computation. A more detailed quantitative assessment of the feasibility of this approach with our specific edge hardware platform (BBB) is an interesting topic for future work.

VII. CONCLUSION

This work reports on novel method - namely pAElla- to increasing the cybersecurity of DCs / SCs, involving AI-powered

edge computing. The method targets real-time malware detection running on an out-of-band IoT-based monitoring system for DCs, and involves feature extraction analysis based on Power Spectral Density of power measurements, along with AE Neural Networks. Results obtained with a robust dataset of malware are promising, outperforming SoA techniques by 24 % in accuracy, with an overall F1-score close to 1, and a False Alarm and Malware Miss rate close to 0 %. Moreover, we propose a methodology suitable for online training in DCs / SCs in production, and release SW code and dataset open source [24]. We envisage this approach as support (pre-filter) for Intrusion Detection Systems (IDS), and encourage further research in this direction. Finally, the approach can also be used in a more general context of Anomaly Detection, opening new perspectives for future works.

ACKNOWLEDGMENT

REFERENCES

[1] S. S. I. Samuel, "A review of connectivity challenges in iot-smart home," in *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, March 2016, pp. 1–4.

[2] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb 2018.

[3] F. Zhang, M. Liu, Z. Zhou, and W. Shen, "An iot-based online monitoring system for continuous steel casting," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1355–1363, Dec 2016.

[4] M. Magno, L. Sigrist, A. Gomez, L. Cavigelli, A. Libri, E. Popovici, and L. Benini, "Smarteg: An autonomous wireless sensor node for high accuracy accelerometer-based monitoring," *Sensors*, vol. 19, no. 12, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/12/2747

[5] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, Aug 2019.

[6] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," 2019.

[7] X. An, X. Zhou, X. Lü, F. Lin, and L. Yang, "Sample selected extreme learning machine based intrusion detection in fog computing and mec," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[8] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, Sep. 2019.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," 2019.

[10] A. Libri, A. Bartolini, and L. Benini, "Dig: Enabling out-of-band scalable high-resolution monitoring for data-center analytics, automation and control," in *The 2nd International Industry/University Workshop on Data-center Automation, Analytics, and Control*, 2018.

[11] ——, "Dig: Enabling out-of-band scalable high-resolution monitoring for data-center analytics, automation and control (extended)," 2018.

[12] T. Ilsche, D. Hackenberg, S. Graul, R. Schöne, and J. Schuchart, "Power measurements for compute nodes: Improving sampling rates, granularity and accuracy," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, Dec 2015, pp. 1–8.

[13] W. A. Ahmad, A. Bartolini, F. Beneventi, L. Benini, A. Borghesi, M. Cicala, P. Forestieri, C. Gianfreda, D. Gregori, A. Libri, F. Spiga, and S. Tinti, "Design of an energy aware petaflops class high performance cluster based on power architecture," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017, pp. 964–973.

[14] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, D. Gregori, S. Tinti, C. Gianfreda, and P. Altoè, "The D.A.V.I.D.E. big-data-powered fine-grain power and performance monitoring support," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, ser. CF '18. New York, NY, USA: ACM, 2018, pp. 303–308. [Online]. Available: http://doi.acm.org/10.1145/3203217.3205863

[15] A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, and C. Cavazzoni, "Paving the way toward energy-aware and automated datacentre," in *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, ser. ICPP 2019. New York, NY, USA: ACM, 2019, pp. 8:1–8:8. [Online]. Available: http://doi.acm.org/10.1145/3339186.3339215

[16] A. Bartolini, A. Borghesi, A. Libri, and F. Beneventi, *Examon HPC Monitoring*. [Online]. Available: https://github.com/EEESlab/examon

[17] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, Aug 2010, pp. 189–194.

[18] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The sunway taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, p. 072001, Jun 2016.

[19] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "On technical security issues in cloud computing," in *2009 IEEE International Conference on Cloud Computing*, Sep. 2009, pp. 109–116.

[20] M. R. Watson, N. Shirazi, A. K. Marnerides, A. Mauthe, and D. Hutchison, "Malware detection in cloud computing infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192–205, March 2016.

[21] K. Popović and Ž. Hocenski, "Cloud computing security issues and challenges," in *The 33rd International Convention MIPRO*, May 2010, pp. 344–349.

[22] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[23] V. Wan, Y. Agiomyrgiannakis, H. Silen, and J. Vit, "Google's next-generation real-time unit-selection synthesizer using sequence-to-sequence lstm-based autoencoders." in *INTERSPEECH*, 2017, pp. 1143–1147.

[24] A. Libri, *pAElla open source software and dataset*, 2019, https://github.com/alibriee/paella.

[25] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Cham: Springer International Publishing, 2014, pp. 109–129.

[26] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485970

[27] H. Sayadi, N. Patel, S. M. P D, A. Sasan, S. Rafatirad, and H. Homayoun, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018, pp. 1–6.

[28] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Online diagnosis of performance variation in hpc systems using machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 883–896, April 2019.

[29] T. Zhang, Y. Zhang, and R. B. Lee, "Dos attacks on your memory in cloud," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 253–265. [Online]. Available: http://doi.acm.org/10.1145/3052973.3052978

[30] A. Borghesi, A. Libri, L. Benini, and A. Bartolini, "Online anomaly detection in hpc systems," 2019.

[31] Intel Corporation, *Intel® 64 and IA32 Architectures Performance Monitoring Events*, 2017 December, Revision1.0.

[32] E. Ates, O. Tuncer, A. Turk, V. J. Leung, J. Brandt, M. Egele, and A. K. Coskun, "Taxonomist: Application detection through rich monitoring data," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 92–105.

[33] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. Gunter, F. Zaffar, M. Caesar, and N. Borisov, "Mining on someone else's dime: Mitigating covert

mining operations in clouds and enterprises," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds. Cham: Springer International Publishing, 2017, pp. 287–310.

[34] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. New York, NY, USA: ACM, 2018, pp. 457–468. [Online]. Available: http://doi.acm.org/10.1145/3196494.3196515

[35] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *Proceedings of 40th IEEE Symposium on Security and Privacy (S&P19)*, 2019.

[36] A. Azmoodeh, A. Dehghantanha, M. Conti, and K.-K. R. Choo, "Detecting crypto-ransomware in iot networks based on energy consumption footprint," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 4, pp. 1141–1152, Aug 2018. [Online]. Available: https://doi.org/10.1007/s12652-017-0558-5

[37] T. Zhang, H. Antunes, and S. Aggarwal, "Defending connected vehicles against malware: Challenges and a solution framework," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 10–21, Feb 2014.

[38] R. Bridges, J. Hernández Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell, "Towards malware detection via cpu power consumption: Data collection design and analytics," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug 2018, pp. 1680–1684.

[39] T. Rosedahl, M. Broyles, C. Lefurgy, B. Christensen, and W. Feng, "Power/Performance Controlling Techniques in OpenPOWER," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham: Springer International Publishing, 2017, pp. 275–289.

[40] Intel, Hewlett-Packard, NEC, Dell, and T. Rep., *IPMI Specification, V2.0, Rev. 1.1*, 2013.

[41] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon, and Y. Georgiou, "Hdeem: High definition energy efficiency monitoring," in *Energy Efficient Supercomputing Workshop (E2SC), 2014*, Nov 2014, pp. 1–10.

[42] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, Aug 2019. [Online]. Available: https://doi.org/10.1007/s12652-018-0803-6

[43] A. Libri, A. Bartolini, M. Magno, and L. Benini, "Evaluation of synchronization protocols for fine-grain hpc sensor data time-stamping and collection," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 818–825.

[44] A. Libri, A. Bartolini, D. Cesarini, and L. Benini, "Evaluation of ntp/ptp fine-grain synchronization performance in hpc clusters," in *2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems (ANDARE 2018)*, 2018.

[45] *Arm NN SDK*. [Online]. Available: https://developer.arm.com/products/processors/machine-learning/arm-nn

[46] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s – a publish/subscribe protocol for wireless sensor networks," in *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, Jan 2008, pp. 791–798.

[47] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 1038–1043.

[48] *Apache cassandra*. [Online]. Available: http://cassandra.apache.org/

[49] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

[50] P. Welch, "The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[51] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, "Deep learning. vol. 1," 2016.

[52] C. Villa-Angulo, I. O. Hernandez-Fuentes, R. Villa-Angulo, and E. Donkor, "Bit-resolution improvement of an optically sampled time-interleaved analog-to-digital converter based on data averaging," *IEEE*

[53] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[54] C. Kachris, B. Falsafi, and D. Soudris, *Hardware Accelerators in Data Centers*. Springer International Publishing, 2018. [Online]. Available: https://books.google.ch/books?id=XnhqDwAAQBAJ

[55] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of Physics: Condensed Matter*, vol. 21, no. 39, p. 395502, 2009.

[56] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.728

[57] J. Dongarra, M. A. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 3–10, 2016. [Online]. Available: https://doi.org/10.1177/1094342015593158

[58] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen, "Gromacs: Fast, flexible, and free," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20291

[59] D. H. Bailey, *NAS Parallel Benchmarks*. Boston, MA: Springer US, 2011, pp. 1254–1259. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_133

[60] Intel, *Intel Haswell Performance Counters List*, 2019, https://download.01.org/perfmon/HSX/.

[61] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, Dec 2008, pp. 413–422.

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

[63] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[64] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[65] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 634 – 644, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197619301721

[66] *TensorFlow Lite*. [Online]. Available: https://www.tensorflow.org/mobile/tflite/

[67] H. Lee and J. Oh and K. Kim and H. Yeon, "A data streaming performance evaluation using resource constrained edge device," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2017, pp. 628–633.

[68] P. Ta-Shma and A. Akbar and G. Gerson-Golan and G. Hadash and F. Carrez and K. Moessner, "An Ingestion and Analytics Architecture for IoT Applied to Smart City Use Cases," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 765–774, April 2018.

[69] H. Isah and T. Abughofa and S. Mahfuz and D. Ajerla and F. Zulkernine and S. Khan, "A Survey of Distributed Data Stream Processing Frameworks," *IEEE Access*, vol. 7, pp. 154 300–154 316, 2019.

[70] S. Chanthakit and P. Keeratiwintakorn and C. Rattanapoka, "An IoT System Design with Real-Time Stream Processing and Data Flow Integration," in *2019 Research, Invention, and Innovation Congress (RI2C)*, Dec 2019, pp. 1–5.

[71] Syafrudin, Muhammad and Alfian, Ganjar and Fitriyani, Norma Latif and Rhee, Jongtae, "Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time

Monitoring System in Automotive Manufacturing," *Sensors*, vol. 18, no. 9, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/9/2946

[72] S. Chintapalli and D. Dagit and B. Evans and R. Farivar and T. Graves and M. Holderbaugh and Z. Liu and K. Nusbaum and K. Patil and B. J. Peng and P. Poulosky, "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1789–1792.

[73] Thein, Khin Me Me, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[74] *Apache Storm*. [Online]. Available: https://storm.apache.org/about/scalable.html

Antonio Libri received a M.Sc. degree in electrical engineering from the University of Genova, Italy, in 2013, with a thesis on Wireless Sensor Networks carried out at University College Cork, Ireland. After two years of working as an embedded software engineer in Socowave Ltd, Cork, Ireland, he joined in 2015 ETH Zurich, Switzerland, pursuing a Ph.D. degree. His research interests focus on data monitoring, synchronization, and AI analytics for automation and control of Data Centers / Supercomputers.

Andrea Bartolini received the PhD degree in electrical engineering from University of Bologna, Italy, in 2011. He is currently assistant professor at the Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna. Before, he was post-doctoral researcher in ETH Zurich, Switzerland. His research interests concern dynamic resource management, ranging from embedded to HPC systems with special emphasis on software-level thermal and power-aware techniques.

Luca Benini is professor of Digital Circuits and Systems at ETH Zurich, Switzerland, and is also professor at University of Bologna, Italy. His research interests are in energy-efficient multicore SoC and system design, smart sensors and sensor networks. He has published more than 1000 papers in peer-reviewed international journals and conferences, four books and several book chapters. He is a fellow of the ACM and Member of the Academia Europea, and is the recipient of the IEEE CAS Mac Van Valkenburg Award 2016.