Towards High-Resolution Monitoring for HPC and Data Center Analytics, Automation and Control

A thesis submitted to attain the degree of DOCTOR OF SCIENCES of ETH ZURICH (Dr. sc. ETH Zurich)
presented by
ANTONINO LIBRI
MSc EE University of Genoa, Italy
born on March 2nd, 1986
citizen of Italy

accepted on the recommendation of
Prof. Dr. Luca Benini, examiner
Prof. Dr. Ayse Coskun, co-examiner

2019
Acknowledgments

First, I would like to thank my supervisor, Luca Benini, for allowing me to pursue my doctorate in his research group, for his confidence in me, for his excellent guidance and precious support, and for providing a great working environment. Special thanks go to my co-supervisor Andrea Bartolini, for his valuable advices and endless support during this time.

I want to thank my colleagues and friends Davide Schiavone and Giovanni Rovere, for sharing with me this period - I will never forget all our travels together. I also thank all the people and collaborators with whom I had the pleasure to share my time and work with, both in Bologna - Francesco Beneventi, Andrea Borghesi, Daniele Cesarini, Francesco Conti, Alessandro Capotondi, Andrea Marongiu, Igor Loi - and here in Zurich - Lukas Cavigelli, Renzo Andri, Björn Forsberg, Germain Haugou, Antonio Pullini, Alfio Di Mauro, Michael Hersche, Robert Balas, Gianna Paulin, Florian Zaurba, Stefan Mach, Fabian Schuiki, Michael Schaffner, Pascal Hager, Pirmin Vogel, Matteo Spallanzani. In particular, I thank Renzo Andri, with whom I started my PhD in the same office, I supervised several projects, and I enjoyed a lot of coffee breaks and my time in Zurich. I would also like to thank Lukas Cavigelli for all the interesting technical discussions, and valuable advices. Moreover, I thank Andrea Cosettini, who shared
the office with me in the last part of my PhD, for all the interesting discussions, lunches and coffee breaks together.

Special thanks go to the support staff of the lab for their excellent work behind-the-scene, namely Hansjörg Gisler, Christoph Wicki, Adam Feigin, Fredi Kunz, Christine Haller, and Frank Gürkaynak.

Finally, I am grateful to my girlfriend Ersilia for supporting and tolerating me over so many years, thank you. The last thank goes to my great family, my father, my mom, my aunt, my uncle, my sister, and my cousins, for having always supported and encouraged me toward new challenges.
Abstract

In the last few years, Data Centers (DCs) and High Performance Computing (HPC) systems are drastically increasing in size and complexity, and the need for novel methods to support their automation, analytics, and control is garnering considerable attention. Based on this support, continuous enhancements in their monitoring infrastructures to provide more detailed information on the activity of the DC are going at the same pace. However, while the increase in measurement resolution (in time and space) is the direction where both industrial and academic researchers are pushing towards, the ensuing vast amount of data to handle entails new challenges, such as bottlenecks on the network bandwidth, storage, and monitoring software tools.

This thesis claims that to properly support the coming generation of complex DCs, as well as HPC systems on the race towards exascale computing, further enhancement are required on current monitoring infrastructures: in particular, novel solutions that take into account an optimized design of both hardware and software are crucial to drastically improve the quality of the monitoring, and thus to face challenges such as improving energy efficiency, promptly detect faults and anomalies, analyze, automate and control the clusters.

To this end, this thesis investigates several research questions toward the design of a new generation of DCs and HPC monitoring systems. In particular, in the first part we address the challenge
of designing the first monitoring platform that enables real-time State-of-the-Art (SoA) high-resolution profiling (i.e., all available performance counters, and the entire signal bandwidth of the power consumption at the plug - sampling at 20µs with an error below 1%) and analytics, both at the edge (node-level analysis) and on a centralized unit (cluster-level analysis). The monitoring system is entirely out-of-band (no overhead on the DC computing resources), scalable, technology agnostic, and low cost. Moreover, it is already installed in a SoA tier-1 HPC cluster (i.e., D.A.V.I.D.E.- 18th in Green500 November 2017).

We then shift our focus to measurement synchronization. Indeed, to exploit the insights coming from high-resolution measurements, fine-grain synchronization is a must: this allows to correlate the several and heterogeneous physical/architectural metrics acquired by the monitoring system, with application phases and states, and thus to have a detailed picture of the activity of the cluster over time. In this second part of the thesis, we evaluate the performance of the two most used synchronization protocols - Network Time Protocol (NTP) and Precision Time Protocol (PTP) - in a real supercomputing cluster (i.e., D.A.V.I.D.E.). We will show that by using our settings, it is possible to achieve microsecond synchronization with NTP (no extra hardware is required, thus can be used off-the-shelf in any DC) and sub-microsecond synchronization with PTP (owing to the dedicated hardware integrated into our embedded monitoring platforms and compute nodes).

Finally, when facing such high-resolution monitoring, the problem of handling a massive amount of data has to be taken into account: leveraging edge vs. centralized analytics is a key research question that has to be investigated. In the third part of the thesis, we will show how to exploit the potential of this new monitoring infrastructure with two use-cases focused on anomaly detection. In particular, the first one shows how to use semi-supervised learning for detecting anomalies with really high accuracy (F1-score close to 1) by using performance counters. In this case, we employ centralized analytics, owing to the low data monitoring rate required (minutes). The second use-case shows instead how to exploit high-resolution power measurements (20µs) directly on edge and carry out real-time inference to detect malware with extremely high accuracy (F1-score close to 1 and Malware Miss rate close to zero) by mean of a semi-supervised learning approach,
based on Power Spectral Density feature extraction and AutoEncoder Neural Networks.
Riassunto

Negli ultimi anni, Data Centers (DCs) e sistemi per High Performance Computing (HPC) stanno crescendo drasticamente in dimensione e complessità, e la necessità di nuovi metodi per supportare la loro automazione, analisi, e il loro controllo, sta guadagnando una attenzione considerevole. Sulla base di questo supporto, si stanno perfezionando le loro infrastrutture di monitoraggio, per fornire informazioni sempre più dettagliate sull’attività del DC. Tuttavia, mentre ricercatori accademici e aziende spingono verso un aumento della risoluzione di monitoraggio (nel tempo e nello spazio), la conseguente crescita della mole di dati da gestire implica nuove sfide, come ad esempio evitare colli di bottiglia nella banda di rete, nello storage e nei tools software utilizzati.

In questa tesi sosteniamo che per supportare propriamente la nuova generazione di DCs e sistemi HPC, ulteriori miglioramenti sono necessari nelle attuali infrastrutture di monitoring: in particolare, nuove soluzioni con un design ottimizzato sia a livello hardware che software sono cruciali per migliorare drasticamente la qualità del monitoring e quindi per affrontare sfide come aumentare l’efficienza energetica, rilevare prontamente guasti e anomalie, analizzare, automatizzare e controllare i clusters.

In questa direzione, questa tesi risponde a diverse domande di ricerca, proponendo il design di una nuova generazione di sistemi di
monitoraggio per DCs e HPC. In particolare, nella prima parte mostreremo il nostro approccio al design del primo sistema di monitoring per DCs, che permette un profiling dello stato del sistema in real-time e ad alta risoluzione (i.e., risoluzione di monitoraggio allo stato dell’arte, in cui teniamo traccia dei vari performance counters e dell’intera banda del segnale del consumo di potenza del server - frequenza di campionamento a 20 µs con un errore sotto l’1%), permettendo un analisi sia sull’edge (analisi a livello di nodo) che centralizzata (analisi a livello di cluster). Il sistema di monitoring è interamente out-of-band (nessun overhead sulle risorse di calcolo del DC), scalabile, indipendente dall’architettura del server e a basso costo. Inoltre, è già stato integrato in un cluster HPC tier-1 (i.e., D.A.V.I.D.E.- 18° in Green500 Novembre 2017).

Spostiamo in seguito il nostro focus sulla sincronizzazione delle misure. In particolare, per sfruttare a pieno l’informazione proveniente dalle misure ad alta risoluzione, è importante un alto livello di sincronizzazione: questo permette di correlare le diverse metriche eterogenee, sia fisiche che architetturali, e quindi di avere un immagine dettagliata dell’attività del cluster nel tempo. In questa seconda parte della tesi, valutiamo le performance dei due protocolli di sincronizzazione più utilizzati - il Network Time Protocol (NTP) e il Precision Time Protocol (PTP) - in un Top500 supercomputing cluster (i.e., D.A.V.I.D.E.). Mostreremo che usando le nostre configurazioni (sia software che a livello di topologia di rete) è possibile ottenere una sincronizzazione nell’ordine di pochi microsecondi con NTP (non è richiesto hardware dedicato, quindi può essere usato off-the-shelf in un qualunque DC) ed una sincronizzazione sotto al microsecondo con il PTP (grazie all’hardware dedicato, integrato sia nei nostri dispositivi embedded usati per il monitoraggio che nei nodi di calcolo del DC).

Infine, lavorando con un monitoraggio ad alta risoluzione, è importante gestire il problema della grande mole di dati da analizzare: come sfruttare abilmente un’analisi sull’edge vs. un’analisi a livello centralizzato è un topic di ricerca chiave che approfondiremo. In particolare, nella terza parte di questa tesi, mostreremo come sfruttare le potenzialità di questa nuova infrastruttura con due casi d’uso su anomaly detection. Il primo mostra come usare una tecnica di apprendimento semi-supervisionato per il rilevamento di anomalie, con utilizzo di performance counters, ottenendo un’alta accuratezza (F1-score vicino all’1). Questo approccio richiede una bassa frequenza di
monitoraggio (ordine dei minuti) e quindi è possibile sfruttare un’analisi a livello centralizzato. Il secondo caso d’uso mostra invece come sfruttare le misure di potenza ad alta risoluzione (20 µs) direttamente sull’edge e fare inferenza in real-time per rilevare malware con ottima accuratezza (F1-score vicino all’1 e percentuale di Malware non rilevati vicina allo zero) per mezzo di un approccio d’apprendimento semi-supervisionato, basato su reti neurali con AutoEncoders ed estrazione di features con Power Spectral Density.
## Contents

1 Introduction 1

1.1 Thesis Overview ................................. 3
    1.1.1 High-Resolution Monitoring System Design . 4
    1.1.2 Fine-Grain Synchronization in Data Centers . 5
    1.1.3 Edge vs. Centralized Analytics ............. 6

1.2 Summary of Contributions ......................... 7

1.3 List of Publications .............................. 8

2 DiG: Enable High-Resolution Data Center Monitoring 11

2.1 Introduction ....................................... 12

2.2 Monitoring System Architecture .................. 15
    2.2.1 High-Resolution Power Sensing ............. 16
    2.2.2 Embedded Monitoring & Edge Analytics ..... 19
    2.2.3 Centralized Monitoring & Cluster Analytics . 21

2.3 Experimental Results ............................. 23
    2.3.1 Monitoring Agents Performance ............. 23
    2.3.2 Power Measurements Validation ............ 25
    2.3.3 Feature Extraction Benchmarking .......... 27
    2.3.4 Edge ML Benchmarking ...................... 31

2.4 Related Work ................................... 33

2.5 Conclusion ....................................... 34
## 3 Fine-Grain Synchronization in Data Centers

3.1 Introduction .................................................. 36
3.2 Data Centers and Fine Grain Synch ......................... 39
3.3 Related Work .................................................. 41
3.4 Time Synchronization Protocols ............................. 44
   3.4.1 Network Topology ........................................ 44
   3.4.2 Message Synchronization Pattern ..................... 45
   3.4.3 Timestamping Methods .................................. 47
   3.4.4 Possible Sources of Jitter .............................. 49
3.5 Experimental Results ........................................ 50
   3.5.1 Synchronization among DC/SC nodes .................. 51
   3.5.2 Synchronization of the Out-of-Band Monitoring System ........................................ 54
   3.5.3 Linear Drift, Scalability and CPU load .............. 56
   3.5.4 How-To Guide - NTP/PTP Settings .................... 58
3.6 Conclusion .................................................... 58

## 4 Centralized-AI based Online Anomaly Detection in Data Centers

4.1 Introduction .................................................. 66
4.2 Related Work .................................................. 67
4.3 Automated Online AD ........................................ 68
   4.3.1 Methodology for Automated AD ....................... 68
   4.3.2 Semi-supervised Learning Based on AEs ............ 69
   4.3.3 Autoencoders Training ................................. 70
   4.3.4 Online Inference ....................................... 72
4.4 Experimental Results ........................................ 72
4.5 Conclusion .................................................... 74

## 5 pAElla: Edge-AI based Malware Detection in Data Centers

5.1 Introduction .................................................. 78
5.2 Background on Malware Detection .......................... 81
5.3 Related Work .................................................. 83
5.4 pAElla: Algorithm & Implementation ....................... 84
   5.4.1 Edge Real-Time Feature Extraction .................... 84
   5.4.2 Edge Malware Detection Inference ..................... 86
   5.4.3 Algorithm Implementation in the DiG IoT Devices 87
Chapter 1

Introduction

Over the last few years, Data Centers (DCs) have extremely grown in size and complexity, and are progressively becoming highly automated computing systems that are essential for all forms of large-scale processing, including High Performance Computing (HPC), also known as (a.k.a.) Supercomputers (SCs). Today more than ever, the needs for novel methods to automate, analyze, and control large-scale systems are growing [1]. In addition, energy efficiency is another key aspect that is crucial on the race toward exascale computing. Indeed, power and energy consumption, bounded by the end of Dennard scaling, start to show their impact on limiting SCs peak performance and cost effectiveness [2,3]. A clear example is given by the trend of last two years in Top500 [4] (i.e., November 2017 vs. November 2019): if we compare the current most powerful SC, Summit, with the previous one, Sunway TaihuLight, we have an improvement in energy efficiency of ~2.4x (4.7 GFlops/W - Flops = Floating point operations per second - vs. 6 GFlops/W), while an enhancement in performance of only ~60% (148.6 PFlops vs. 93 PFlops). This is a direct effect of hitting the power wall in supercomputing installations and demonstrates that
we are clearly in an era driven by novel hardware (HW) technologies and efficient usage of the resources.

In this direction, industrial and academic researchers are paying closer attention to a more efficient management of storage, networking, and computational/infrastructure resources, and how to analyze and control them in real-time using automated technologies based on Artificial Intelligence (AI) and Machine Learning (ML). Thanks to these techniques, also tasks like detection and prediction of anomalies/failures, or DC security and data protection, can be now tackled with automated approaches (cf. Chapter 4, Chapter 5, [5–8]). However, meeting these challenges requires customized and highly automated DC designs that are aware of their performance and power consumption workloads, and are tightly coupled with the needs of the running applications. Therefore, novel and effective solutions based on Hardware/Software (HW/SW) co-design for DC monitoring infrastructures are required [1].

Recent work [2] reports four criteria to assess the quality of a monitoring infrastructure: (i) high accuracy, (ii) sampling rate, (iii) measurement of individual components, and (iv) scalability to a large number of sampling points. State-of-the-Art (SoA) monitoring systems include built-in tools to collect per-component power and performance measurements (e.g., Intel RAPL, IBM Amester [9,10]), and custom solutions to get high-resolution insights on power consumption (e.g., HDEEM, HAEC [2,11,12]). However, with the increase in measurement resolution (in time and space) and the ensuing vast amount of data to handle, new challenges arise, such as bottlenecks on the network bandwidth, storage, and SW tools that have to handle the measurements. This becomes much more evident when facing large-scale DCs (i.e., as we will show in Chapter 2, using HAEC - which is a SoA high-resolution monitoring system - in large-scale clusters like Sunway TaihuLight would not be feasible). There is a need to develop novel cost-effective solutions for high-resolution power and performance monitoring, while ensuring (i) real-time analytics at large scale, (ii) the flexibility of being installed in different DC architectures, and (iii) the robustness needed for production environments.

Fine-grain synchronization is another key component to exploit the full potential of high-resolution monitoring infrastructures [2,11,13]. The capability of these systems to adapt to specific application
requirements relies on sensing and correlating several distributed physical parameters with application phases and states. Meeting such requirements allows achieving a better use of the resources, higher throughput, and higher energy-efficiency [13]. As the capability of drawing such correlations relies on the synchronization across a network of nodes and measuring devices, the evaluation of synchronization protocols performance becomes a critical component.

Finally, when facing such high-resolution monitoring, the problem of dealing with a vast amount of data has to be taken into account: leveraging edge vs. centralized analytics is a crucial research question that has to be investigated.

1.1 Thesis Overview

In this thesis, we investigate several research questions towards the design of new generation of DC monitoring infrastructures. In the first part, the challenges of a high-resolution DC monitoring system from a HW point of view are addressed. Later the work is extended on the evaluation of time synchronization protocols and how to exploit their best performance in a DC, along with how to address the trade-off between edge vs. centralized analytics in the monitoring system with two high-impact studies on Anomaly Detection (AD). The thesis covers all the layers involved in a monitoring infrastructure, from (i) the HW design to interface with custom sensors, to (ii) the SW layers for collecting and analyzing measurements, up to (iii) data synchronization and (iv) ML analytics.

Figure 1.1 shows an overview of the thesis structure. After the introduction, we proceed in Chapter 2 by reporting on current SoA high-resolution monitoring infrastructures, their main issues, and how we addressed them with DiG [1] (i.e., Dwarf in a Giant), which is our Internet-of-Things (IoT)-based highly scalable monitoring system to carry out high-resolution monitoring and analytics, and that is already integrated in a production environment. We continue in Chapter 3 by evaluating the performance of SoA synchronization protocols for fine-grain synchronization, carrying out performance measurements in a SoA HPC cluster (i.e., D.A.V.I.D.E. [3]), and providing SW settings to can be used in common DCs. We then exploit our monitoring
platform to investigate on novel approaches for AD (Chapter 5) and security (Chapter 5) of DCs, by leveraging coarse and fine grain measurements between the edge and the centralized monitoring unit of DiG. We conclude the thesis in Chapter 6, with an overview of the main results and possible future works.

The following sections provide for each topic a summary of the work, main challenges, and contributions. In addition, project involvements and external contributions are reported.

1.1.1 High-Resolution Monitoring System Design

Effective solutions for high-performance analytics, automation, and control are crucial for today and future large-scale DCs and SCs. DC designs can no longer be decoupled from the applications, and the need for supporting computational/infrastructure resources efficiently and control them in real-time with automated technologies, is increasing with the complexity and size of these systems. While industrial and academic researchers are pushing towards high-resolution monitoring systems for high-performance analytics [2,9–12], the problem of handling the huge amount of data - coming from all the heterogeneous
1.1. THESIS OVERVIEW

sensors - is still an issue for current monitoring system designs when scaling to large clusters, and has to be taken into account.

In this part of the thesis, we report on the HW and SW co-design of our monitoring infrastructure - DiG [1] - which 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 [3, 14, 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]). Finally, it is technology agnostic, and as we will show in Chapter 2, we installed it in different architectures such as ARM, Intel, and IBM.

This part of the work was supported by E4 Computer Engineering SpA, the supercomputing center CINECA, the EU H2020 FET projects ANTAREX (g.a. 671623) and OPRECOMP (g.a. 732631), and the EuroLab-4-HPC project.

1.1.2 Fine-Grain Synchronization in Data Centers

One major challenge of high-resolution DC monitoring systems is synchronization. Such infrastructures include a large variety of sensors for measuring architectural and physical run-time parameters. To carry out high-performance analytics, it is crucial to correlate applications running in the compute nodes with their architectural events (i.e., performance counters) and physical metrics (i.e., power and energy consumption) [12, 13]. However, DCs / SCs applications usually run on multiple nodes [17] (in large-scale systems, node counts is in the order of several thousand, e.g., Sunway TaihuLight, 3rd in Top500 of November 2019, involves around 41 thousand compute nodes [18]), each consisting of several processing elements and measuring points. As a result, the monitoring system can be seen as a multitude of agents that measure different metrics for the HW components. Ultimately, the capability of correlating these monitoring points to form useful application metrics is bounded by their sampling rate as well as by the synchronization
between the monitoring agents, which is essential for the global ordering of events. Moreover, as reported in prior works [19–23], accurate and precise synchronization solutions are essential for profiling, debugging, scheduling and maintenance techniques, and thus for improving the performance of applications, and more in general of the DC/SC infrastructure. In this part of the thesis, we evaluate the two widely used synchronization protocols, i.e., the Network Time Protocol (NTP) and the Precision Time Protocol (PTP), within DC clusters, and investigate on how to achieve the best performance to address the aforementioned challenges.

This part of the work was supported by the EU H2020 FET-HPC project ANTAREX (g.a. 671623), the FP7 ERC Advance project MULTITHERMAN (g.a. 291125) and E4 Computer Engineering SpA.

1.1.3 Edge vs. Centralized Analytics

One of the main challenges of high-resolution monitoring is the huge amount of data to be handled. Generally, DCs/SCs monitoring systems are characterized by a vertical infrastructure, where power and performance measurements are collected from all compute nodes and stored in a database for analysis (e.g., HDEEM, powerDAM, PowerInsight [2, 11, 24]). On the one hand this design allows to (i) have a profiling picture over time of the overall cluster activity, and (ii) carry out offline analysis, on the other hand this vertical structure becomes a bottleneck when facing with high-resolution measurements, several heterogeneous sensors, and large-scale clusters. In this case, balancing the data processing between the centralized monitoring unit and the network-edge to alleviate the computational, storage, and bandwidth burdens would be a preferable choice. This last part of the thesis addresses such a trade-off by investigating some of the challenges reported in the introduction (i.e., Anomaly Detection and security of DCs). With this aim, we use our monitoring system DiG, to test the feasibility of novel ML approaches, and propose two new methodologies based on semi-supervised learning with AutoEncoder (AE) Neural Networks (NNs) for anomaly and malware detection in DCs.

This part of the work was supported by EU H2020 project IoTwins (g.a. 857191) and E4 Computer Engineering SpA.
1.2 Summary of Contributions

The main contributions of this thesis can be summarized as follows:

1. The design of the first out-of-band high-resolution highly-scalable monitoring infrastructure for DCs/SCs- namely DiG- that exploits both edge (node-level) and centralized (cluster-level) computing for DC AI-analytics, automation, and control. The system provides best-in-class time resolution of power measurements (20 µs), with a precision below 1%, and can be interfaced with built-in tools (e.g., IBM Amester) to collect performance metrics. Moreover, it is technology agnostic and low-cost (no motherboard redesign required). Finally, the system is already installed in a real tier-1 SC in production (i.e., D.A.V.I.D.E. [3]).

2. A performance evaluation of the two widely used synchronization protocols, NTP and PTP, in terms of accuracy, precision, and scalability, when they are used in the context of DCs/SCs. We demonstrate that with a proper configuration, both synchronization protocols can achieve the target synchronization of sub-milliseconds needed for many applications in today’s DCs/SCs.

3. A novel automated method for Anomaly Detection in HPC systems and DCs, based on AutoEncoder NNs. Our technique exploits semi-supervised learning on coarse-grain power and performance measurements, and achieves a high accuracy (around 90–95% of detection accuracy) in a dataset of several months of acquisitions with real supercomputing applications from the D.A.V.I.D.E. HPC community.

4. A novel out-of-band approach, we named it pAElla, running on IoT-based monitoring systems for real-time Malware Detection (MD) in DCs. Results report an overall F1-score close to 1 (0.998), and False Alarm (FA) and Malware Miss (MM) rates close to 0% (0% and 0.5%, respectively). We compare our approach with other SoA ML techniques for MD and show that, in the context of DCs/SCs, pAElla can cover a wider range of malware, outperforming other SoA methods.
5. A methodology for online ML training in the DC infrastructure, suitable to be run in a monitoring system in production like DiG.

6. Open source release of SW and dataset of both ML methods. Notice that the dataset of pAElla is the first of its kind, providing to the security research community high-resolution power and performance measurements for carrying out and benchmarking novel analyses.

1.3 List of Publications

The content of this thesis and the main contributions have been published in the following conference and journal papers:


The following publications with contributions by the author focus on related topics, but are not included in this thesis:
1.3. LIST OF PUBLICATIONS


Further publications with contributions by the author not explicitly covered by this thesis are:


Chapter 2

DiG: Enable High-Resolution Data Center Monitoring

Today, DCs and HPC systems are becoming increasingly large and complex, and we need novel and scalable methods to support their automation, analytics and control. Tracking their power consumption and performance counters is essential for these methods. SoA monitoring systems provide built-in tools to collect power and performance measurements, and custom solutions to get high-resolution insight on their power consumption. However, with the increase in measurement resolution and the ensuing huge amount of measurement data to handle, new challenges arise, such as bottlenecks on the network bandwidth, storage and SW overhead on the monitoring units. To face these challenges we propose a novel monitoring platform for DCs, which enables real-time high-resolution profiling (i.e., all available performance counters and the entire signal bandwidth of the power consumption at the plug - sampling up to 20 µs - with an error below 1%) and analytics, both at the edge (node-level analysis) and on a centralized unit (cluster-level analysis). The monitoring infrastructure is entirely out-of-band, scalable, low cost and technology agnostic, and

\section{Introduction}

DCs and SCs are becoming increasingly complex and the need for novel methods to support their automation, analytics and control is garnering considerable attention [33]. In this direction, industrial and academic researchers are pushing toward the use of AI and ML techniques to address non-trivial challenges such as efficient management of computational/infrastructure resources, detection of anomalies and failures, and security. As an example, Duplyakin \textit{et al.} [34] have shown how to get high-confidence predictions of the time-to-completion and energy consumption of scientific applications (which can help for a more efficient usage of the resources) via Active Learning techniques applied to regression problems. Other examples are based on unsupervised learning, such as [35] which introduces methods for real-time AD on streaming data (useful for an early warning about problems in the system and the hosted applications), and [7] that shows a way to detect malware using HW features.

All these techniques exploit low-level monitoring of the HW of the DC infrastructure (i.e., application and system performance, and related power and energy consumption). In particular, depending on the target use-case, some features can reveal more information than others: a highly flexible monitoring has to collect as many metrics as possible. On the other hand, this implies to face three main bottlenecks related to the large amount of monitoring data produced: (i) overhead on the network’s bandwidth, (ii) overhead on the data storage capacity (to save measurements for post-processing analysis) and (iii) overhead on the SW tools that have to handle the measurements (in real-time and offline).

This is depicted in Figure 2.1 (left), which shows that the node dedicated to the monitoring SW stack has the complete view of the status of the cluster (and thus can exploit measurements for ML analysis), but has to deal with the above mentioned bottlenecks. To give an example, Ilsche \textit{et al.} developed a high resolution power monitoring system (i.e., HAEC [36]) that supports a sampling rate of
2.1. INTRODUCTION

500 kS/s (kilo Samples per second) on 4 custom sensors. In general, high resolution power monitoring instrumentation is the current trend for both industrial and academic HPC facilities and DCs [2,11,36]. This is useful to appreciate the power consumption of application phases, but of course the finer the granularity the more difficult is to scale to a large number of nodes in a machine. For instance, instrumenting with HAEC the SC Sunway TaihuLight (~41 thousand compute nodes [18]) would require a data collection network bandwidth of around 82 GS/s, with obvious overheads on SW and storage to handle it.

An intuitive solution is to bring some of the “monitoring intelligence” to the edge and codesign the monitoring infrastructure to leverage data analysis between distributed monitoring agents and a centralized unit. This is represented in Figure 2.1 (right), which shows distributed smart monitoring agents that can carry out real-time analysis per node (e.g., feature extraction, ML inference, etc.) and share information with the centralized monitoring at a much lower rate (e.g., detection of an anomaly in a node, plus other measurements at a lower rate needed for cluster level analysis). In this distributed architecture, each monitoring agent has the complete knowledge of the status of its node, while the centralized monitoring unit has the complete view of the cluster, thus can carry out analysis at a higher level.

Current SoA monitoring solutions allow to collect measurements in-band and out-of-band by means of built-in tools (e.g., Amester [10]...
or RAPL [37], which expose HW performance counters) or custom sensors (e.g., HDEEM [2] or HAEC [36], which provide fine grain power measurements), where the benefit of the out-of-band solution is no overhead on the computing resources. However, to the best of our knowledge, there is not yet a monitoring infrastructure for compute clusters and high performance machines that provides a flexible way to analyze all possible features (i.e., all available HW performance counters and the entire signal bandwidth of the node’s power consumption). This chapter, focuses on a novel scalable and high resolution monitoring infrastructure for DCs and HPC systems. The system is completely out-of-band and provides a highly flexible environment to work both at the edge and at cluster-level for DC analytics, automation and control.

Contribution:

1. design of an out-of-band monitoring infrastructure - we named it DiG- that exploits edge monitoring agents and centralized cluster-level monitoring for DC analytics, automation and control. The platform design provides a highly flexible environment to tackle different challenges. We designed a custom power sensor at the plug to monitor the power consumption at high resolution, covering the entire signal bandwidth (sampling up to 20 µs) with a measurement precision below 1% (σ) (therefore also suitable for the most rigorous power measurement requirements to benchmark a computing system in Top500 [38]). The system allows to interface with existing out-of-band telemetry (e.g., Amester [10], IPMI [39]), but also with in-band built-in tools if required (e.g., RAPL [37]). All the measurements are synchronized at sub-microseconds precision to obtain a detailed picture over time of the nodes and cluster state. We adopted a scalable and lightweight interface to the centralized monitoring (i.e., MQTT [40]) to support large-scale computing centers. The monitoring infrastructure is technology agnostic (i.e., already tested on different architectures, such as Intel, ARM, and IBM) and low cost (i.e., the custom power sensor does not require any motherboard redesign).
2.2. Monitoring System Architecture

2. we report (i) the performance of the monitoring agents (i.e., measurements granularity, precision, synchronization, SW overhead and scalability), along with (ii) an extensive campaign of ML inference benchmarks running on the dedicated embedded computers and based on deep Residual Networks (a.k.a. ResNets [41]), to obtain an assessment of their real-time inference capabilities, and (iii) an extensive set of tests based on frequency-domain analysis to show the capability of the high resolution monitoring to unveil high-frequency components directly related to the computation activity and including also two use-cases that can be used for AD.

3. we validated and calibrated our high-resolution power measurements, and provide detailed information on accuracy and precision (best-in-class with respect to - w.r.t. - SoA DC monitoring systems); moreover, we integrated the monitoring infrastructure in a SoA HPC cluster (i.e., D.A.V.I.D.E. [3]) that is already in production and available to the users community since more than one year.

4. We provide detailed information of both HW and SW architectures for the whole monitoring infrastructure (Sections 2.2.1 and 2.2.2 describe the edge infrastructure, while 2.2.3 describes the cluster-level SW, namely ExaMon, which can be downloaded from [16]).

Outline: Section 2.2 presents the monitoring architecture. Its performance is analyzed in Section 2.3, together with several case studies of frequency-domain analysis on the high-resolution power measurements. We report related work in Section 2.4 and conclude the chapter in Section 2.5.

2.2 Monitoring System Architecture

One of the main challenges we faced during the monitoring system design was to make it suitable for different HW architectures and low cost. With this goal, we targeted only what is missing on today’s built-in monitoring solutions [36]: a custom power sensor that allows high resolution measurements. We placed it at the node power source
to avoid motherboard re-design or modification. We then interfaced it with a dedicated low-cost embedded computer (one per node) that is suitable for monitoring applications.

A second challenge was to make the system highly flexible in terms of monitoring capabilities. With this goal, we interfaced the embedded computer with built-in tools to get per-component monitoring (i.e., HW performance counters) and have the complete knowledge of the status of the node. This information, along with the high resolution power monitoring, can reveal not only insights on application behavior but also patterns on performance/failures of components (e.g., fans, HDDs).

Finally, we exploited a scalable and lightweight interface (i.e., MQTT) to send information to a centralized monitoring unit and perform cluster-level analytics. Figure 2.2 shows the main components of the monitoring system that will be described in this section.

![Figure 2.2: Sketch of the monitoring system architecture.](image)

### 2.2.1 High-Resolution Power Sensing

To provide high resolution measurements of the nodes’ power consumption we placed a power sensing module between the Power Supply Unit (PSU) and the DC-DC converters that provide power for all the processing elements (PE) / electrical components within the node. Figure 2.3 shows the schematic of the power sensing module. We
use a voltage divider based on high precision resistors to measure the voltage and a current transducer to measure the current. Their outputs are then connected to the Analog-to-Digital Converter (ADC) integrated in the embedded monitoring board, via first-order low-pass filters needed to counter aliasing effects. Indeed, due to the high operating frequencies of DCs / HPC nodes, the power consumption is highly dynamic, and therefore an anti-aliasing filter is required. We have chosen a voltage divider as it provides a simple but effective solution to properly scale the voltage in input to the ADC without any additional HW (e.g., an isolated power supply would be needed if using active components).

For the current transducer, we tested two configurations: one based on a Hall Effect (HE) sensor and one based on a current mirror and shunt resistor. Thanks to the high output linearity, both solutions obtain satisfactory results. We tested the first configuration with Intel Xeon E5-2600 (Haswell) and ARM Cavium TunderX architectures, while the second one on a cluster based on OpenPOWER IBM POWER8 (i.e., D.A.V.I.D.E. [3]). Figure 2.4 shows the three architectures. In particular, the HE sensor is the Allegro MicroSystems ACS770 [42]. It can measure currents in the range 0–100 A with low-intrusiveness, good linearity and high precision sensitivity (40 mV/A). It has a typical bandwidth of 120 kHz and an internal conductor resistance of 100 µΩ, which implies a negligible
power loss. All these features make it suitable for integration with standard PSUs used on Intel and ARM architectures.

![Figure 2.4: Different architectures where DiG was tested.](image)

For the configuration based on current mirror and shunt resistor, we exploited the same current mirror already used by the Baseboard Management Controller (BMC) on IBM POWER8 to measure at a coarse grain the total node power consumption. This provides a twofold benefit: measuring currents in a wider range (0–250 A) and avoiding extra cost for current sensing components. Figure 2.5 shows an overview of a compute node of D.A.V.I.D.E., where we it is visible the integration with DiG (i.e., the red circle highlights the power sensing module + the embedded computer). Specifically, D.A.V.I.D.E. consists of 45 IBM OpenPOWER compute nodes, plus two management nodes, grouped in 3 racks. Each node has an Open Compute Project (OCP) form factor, based on IBM Minsky, and includes 2 IBM POWER8 processors and 4 NVIDIA Tesla P100 with NVLink optimized for best performance. Moreover, the system is liquid-cooled, and all the nodes exploit an EDR InfiniBand connection (100 Gbit/s). Based on Green500 of November 2017 [43], D.A.V.I.D.E. was ranked 18th most efficient worldwide SC, with a peak performance (Rmax - Maximal LINPACK performance achieved) of 615.4 TFlops and a power efficiency of ~7.8 GFlops/W. The cluster was developed in collaboration with E4 Computer Engineering SpA and Wistron Corporation, for the Partnership for Advanced Computing in Europe.
2.2. MONITORING SYSTEM ARCHITECTURE

Figure 2.5: Overview of a D.A.V.I.D.E. compute node.

(PRACE) program, and hosted in the Italian supercomputing center CINECA [3].

Finally, to avoid a possible measurement accuracy degradation due to heating effects on the resistors of the voltage dividers, we have chosen resistors with equal Temperature Coefficient of Resistance (TCR), and placed them close to each other (similar temperature on both) and far from external sources of heating (to avoid uneven heating effects).

2.2.2 Embedded Monitoring & Edge Analytics

We selected a Beaglebone Black (BBB) [44] as embedded computing board as it provides several interesting features off-the-shelf: (i) it includes a 12-bit Successive Approximation Register (SAR) ADC needed for the power-sensing module, (ii) HW support for PTP which allows sub-microsecond measurements synchronization [13,26], and (iii) an ARM Cortex-A8 processor with NEON technology, useful for Digital Signal Processing (DSP) 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). Moreover, the Sitara AM335x chip used in BBB includes two programmable real-time units (PRUs) useful for real-time acquisition and extra processing on-board.

It should be noted that standard computing servers already integrate an embedded system used for real-time monitoring and management, namely the BMC. This is usually a closed platform with no access to the firmware, but thanks to the recent OpenBMC project [10] few vendors started to release it open-source. We decided to do not use the BMC for this purpose as (i) we needed to add extra HW to integrate an ADC and interface it with the custom power sensor (the BBB already includes an ADC), (ii) it does not provide HW support for PTP, and (iii) it is based on an old ARM processor family (i.e., ARM11 [46]) which is not a good choice for edge analytics. Moreover, (iv) it is a critical component to ensure safe working conditions of the nodes, thus it is more convenient to do not overwhelm its limited processing resources with our monitoring / edge-analytics SW stack.

Figure 2.6 reports the embedded monitoring stack design (notice that a second version, adapted to the needs of our MD use-case, is showed in Chapter 5). The bottom layer represents the ADC HW module. We exploit the ADC continuous sampling mode to continuously sample the two input channels (i.e., current and voltage), average and store them in a HW FIFO that is managed by a kernel driver. By tuning the ADC sampling frequency (FsADC) and the HW averaging (AVG), it is possible to set the frequency (Fs) at which samples enter the SW layers.

When the HW FIFO reaches a watermark on the number of samples acquired (we set it to 16), an interrupt is raised and the samples are flushed into the main memory (kernel FIFO) by the ADC driver. In particular, the ADC driver involves two routines (IRQ handlers): the top half, which monitors the watermark on the HW FIFO, and the bottom half that is used to flush the samples into the kernel FIFO.

Finally, the power measurements are exposed via the Industrial I/O (IIO) Subsystem API to a user-space monitoring daemon, which is in charge of converting data from integer to Watt and associate them with a timestamp. This daemon also collects node’s performance measurements from HW performance counters via built-in tools (e.g., IPMI [39], Amester [10] and RAPL [37]). In this way, we can perform
edge analytics on a target use-case (e.g., ML inference for AD) and send the results, together with the power and performance measurements at a lower rate, to the centralized monitoring unit for cluster-level analytics.

It should be noted that by using the continuous sampling mode, the HW guarantees a negligible uncertainty on the acquisition time of consecutive samples, ensuring correctness of the energy computation at a fine granularity [2]. Moreover, to make negligible both overhead and uncertainty introduced by the timestamp’s function call, we only generate timestamps at each flush of the kernel FIFO, and not for every sample (timestamps for every sample are then derived accordingly to the number of samples acquired).

2.2.3 Centralized Monitoring & Cluster Analytics

We exploit centralized monitoring, based on the open-source ExaMon monitoring platform [16,47], to carry out cluster-level analytics with data coming from multiple nodes. To send data from the distributed monitoring agents to the centralized monitoring unit, we adopted MQTT [40], which is a robust, lightweight and scalable protocol, already used for large-scale systems both in industry and academia (e.g., Amazon, Facebook, [40,47]). Figure 2.2 outlines its publish / subscribe
communication model, where the publishers (running in the embedded computers) send measurements to a broker, along with a topic that corresponds to the monitored metric (e.g., power consumption). The broker resides in the centralized monitoring unit together with the subscriber. The latter is used to filter and collect the data that is interested in, and store them in a highly scalable database, Apache Cassandra [48], through KairosDB [15], which is a plugin for time series. Data is finally exposed to the application layer, which consists of several applications, such as the Big Data engine Apache Spark [49], Grafana [47] for data visualization, and custom applications for data analytics. Notice that Apache Spark can access the data from both Cassandra (via kairosDB) and directly from the broker, enabling ML analytics both in streaming and batch mode. An overview of the whole centralized monitoring architecture is reported in Figure 2.7.
2.3 Experimental Results

This section starts by reporting the performance of the monitoring agents, along with the validation of the high-resolution power measurements from the point of view of accuracy and precision. It then presents a set of monitoring use-cases based on Fourier analysis to show the capability of the high-resolution monitoring in revealing fine grain computation activity. Finally, we report a campaign of ML inference benchmarks, based on ResNets running on the embedded monitoring platform, to show an example of the capability of DiG on carrying out edge ML analytics.

2.3.1 Monitoring Agents Performance

*Performance Measurements:* To provide completely out-of-band monitoring, we integrated our infrastructure in D.A.V.I.D.E. [3,14], taking advantage of its IBM out-of-band telemetry by collecting via Amester through the On Chip Controller (OCC) [10] 242 metrics per-component every 10s (*e.g.*, the performance of Core, Cache, Fan, etc.), and via IPMI 89 metrics per-component every 5s. All these measurements are sent to the centralized monitoring for cluster-level analytics, but can also be exploited on-board for real-time edge analysis on a target use-case (*e.g.*, detection of anomalies).

*Power Measurements:* To cover the full power consumption bandwidth at the node plug (*i.e.*, tens of microseconds, observed with a professional oscilloscope - *Keysight DS0X3054T* - attached to the power sensor) and at the same time to avoid overwhelming the CPU with the data acquisition routine, we tested several sampling rates (*Fs*) that we report in Figure 2.6.2. In particular, the best trade-off corresponds to a sampling rate of 800 kS/s per channel and HW averaging every 16 samples. This is equivalent to 50 kS/s (*i.e.*, 20 µs) and allows to obtain several benefits: (i) to cover the entire signal bandwidth, (ii) to keep the CPU load below 35% and (iii) to reach a precision below 1% (σ) of uncertainty (a.k.a. oversampling and averaging method [50]). It is noteworthy that this precision makes our system suitable for the most rigorous requirement needed to benchmark an HPC system in Top500 [38]. Finally, we send the measurements to the centralized
monitoring for cluster-level analytics at the rates of 1 s and 1 ms, while measurements at higher resolution can be analyzed directly at the edge.

**SW overhead:** The entire BBB CPU load of the monitoring SW stack (power and performance) is below 46%. In particular, performance monitoring requires roughly 11% and the power monitoring around 35%. We have run also some benchmarks to evaluate the computing capabilities of the embedded platform: we can perform (i) real-time Power Spectral Density (PSD) analysis of the high resolution power measurements (*e.g.*, useful for feature extraction [51]), in a time window of around 40 ms with roughly 7% of CPU usage, and (ii) ML inference via TensorFlow on these PSDs, implementing ResNet with 8 layers and channels \{8, 8, 16, 32\} and respecting a real-time constraint of 30 ms per spectrogram (a detailed analysis can be found in Section 2.3.4). Moreover, it must be noted that we did not use any optimization to run ML inference (*e.g.*, ARM NN SDK [45] or TensorFlow Light [52]), which means these results can be further improved.

**Synchronization:** To ensure accurate and precise timestamping of the measurements, we exploit PTP HW obtaining sub-microsecond synchronization (*i.e.*, smaller than the sampling period) across multiple nodes and embedded monitoring devices. Moreover, results in Chapter 3.4 show that with a proper setting of NTP, it is possible to reach synchronization with an uncertainty of a few microseconds in today’s DCs and HPC clusters.

**Scalability:** To benefit from a scalable interface to the centralized monitoring unit, we exploit Mosquitto [47] which is a Linux implementation of MQTT that consists of a single thread process. Our tests show that Mosquitto broker (*i.e.*, the bottleneck of the network) running in an Intel Xeon E5-2600 (Haswell) can handle up to 16 publishers that send data every millisecond using just 30% of a core, and of course, it is possible to increase the number of brokers if needed. In our current configuration of the monitoring system integrated in the D.A.V.I.D.E. HPC machine, we use one broker for all performance counters and three brokers (one per rack) for the power measurements, with no particular issue for the users/system admins of the DC since November 2017. Moreover, we tested MQTT with a similar configuration on all 516 compute nodes of GALILEO at CINECA, Italy (Intel Xeon
2.3. EXPERIMENTAL RESULTS

E5-2630v3 processors), proving that this interface is suitable for even larger scale systems.

2.3.2 Power Measurements Validation

To verify the accuracy and precision of the high resolution power measurements, we attached the DiG power sensing module to a dummy load (depicted in Figure 2.3) and calibrated the current and voltage sensors against a high-precision reference multimeter. In this way, we can also determine the conversion factors (offset and gain) for both voltage and current needed for computing power consumption in Watt. Figure 2.8 reports the results of the current measurements after calibration for both configurations, namely HE Sensor and current mirror plus shunt resistor. In particular, the x-axis corresponds to the different input loads that we applied to the power sensing module accordingly to the allowed current range (i.e., 0–100 A for the HE Sensor and 0–200 A for the shunt resistor), while the y-axis reports the current measured by DiG (dots) and a linear regression on the measurements (straight line). As can be seen from the plot, all the DiG measurements well match the input load, so the curve is linear across the full range (i.e., coefficient of determination \( R^2_{Shunt} = 0.9999 \) and \( R^2_{HE} = 0.9997 \)), except around zero, which is anyway not a problem as compute nodes never work in this low range of currents. We carried out the same procedure for the voltage measurements and obtained similar results.

After calibrating DiG, we can evaluate the precision on the power measurements (i.e., standard deviation - \( \sigma \) - and Coefficient of Variation - CV). With this goal, we can start by quantifying the precision of each ADC channel (current and voltage) independently, and use the propagation of uncertainty theorem for computing the uncertainty of the resulting power [53]. Indeed, given the measured current and voltage with uncertainties, \( I \pm \sigma_I \) and \( V \pm \sigma_V \) (where \( I \) and \( V \) correspond to the measured average value), under the assumption they are not correlated, the uncertainty on the power measurements is:

\[
\sigma_P \approx \sqrt{I^2\sigma_I^2 + V^2\sigma_V^2} \quad (2.1)
\]
Table 2.1 reports the resulting precision at 50 kS/s for three different server operating conditions: idle, medium load and maximum load (e.g., to give an idea in an Intel Xeon E5-2600 these conditions correspond to roughly 180 W, 600 W and 1200 W, respectively). The precision for around 68.3% of the power measurements (σ) is bounded between 1.73–3.96 W, for the minimum (idle) and maximum load, respectively, and increases of a factor of 3 when considering 99.7% of the samples (5.2–11.88 W for 3σ). Of course, the CV follows the opposite trend: it decreases when the power consumption increases (from 0.96% in idle to 0.33% for maximum workload).

The table reports also the precision when sampling at lower rates - by applying a SW average -, namely 25 kS/s, 1 kS/s and 1 S/s. As can be seen, σ drastically improves to a few watt precision already at 25 kS/s (even at the maximum load). With the goal to dynamically increase the DiG precision on the power measurements when required, the monitoring daemon can be set to dynamically switch to a lower sampling rate (e.g., to 25 kS/s) by averaging in SW the power samples if it is monitoring low currents for a certain time period. Thanks to this trade-off we can always keep the monitoring precision below a pre-set threshold (up to sub-watt precision), which makes DiG suitable
2.3. EXPERIMENTAL RESULTS

Table 2.1
DiG Precision based on dynamic SW average.

<table>
<thead>
<tr>
<th></th>
<th>Idle</th>
<th>Mid-Load</th>
<th>Max Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ (CV)</td>
<td>σ (CV)</td>
<td>σ (CV)</td>
<td></td>
</tr>
<tr>
<td>50 kS/s</td>
<td>1.73 W</td>
<td>2.58 W</td>
<td>3.96 W</td>
</tr>
<tr>
<td></td>
<td>(0.96%)</td>
<td>(0.43%)</td>
<td>(0.33%)</td>
</tr>
<tr>
<td>25 kS/s</td>
<td>0.5 W</td>
<td>1.26 W</td>
<td>2.28 W</td>
</tr>
<tr>
<td></td>
<td>(0.28%)</td>
<td>(0.21%)</td>
<td>(0.19%)</td>
</tr>
<tr>
<td>1 kS/s</td>
<td>0.47 W</td>
<td>1.14 W</td>
<td>2.16 W</td>
</tr>
<tr>
<td></td>
<td>(0.26%)</td>
<td>(0.19%)</td>
<td>(0.18%)</td>
</tr>
<tr>
<td>1 S/s</td>
<td>0.32 W</td>
<td>1.02 W</td>
<td>2.04 W</td>
</tr>
<tr>
<td></td>
<td>(0.18%)</td>
<td>(0.17%)</td>
<td>(0.17%)</td>
</tr>
</tbody>
</table>

to be used in production environments as a high-precision HPC energy monitoring solution.

2.3.3 Feature Extraction Benchmarking

This section aims at showing the capability of the high resolution monitoring in unveiling high-frequency components directly related to the computation activity. We exploit Fourier analysis as an example of feature extraction technique for time series that is suitable for deep learning algorithms (e.g., Deep Neural Networks - DNNs [51]). We will show in Chapter 5 how we use a similar approach for MD detection. Moreover, future works can extend this real-time analysis targeting specific use-cases (e.g., AD in workloads) and (i) exploit it, together with performance counters, as input data for DNN models running inference in the monitoring agents [51]; or (ii) just send it with lossy compression algorithms (data is sparse, as shown by the following tests) to the centralized monitoring for cluster-level analytics. We note that due to the limitations of SoA power monitoring support for compute nodes, up until now this kind of analysis could not be performed in production DCs.

We start the evaluation with a synthetic benchmark on the compute node, consisting of an Interrupt Service Routine (ISR) that we run every $\sim 10$ ms (roughly 100 Hz) with different sets of instructions. In particular, the first set of instructions corresponds to a duty cycle of 20% in power consumption (i.e., 2 ms of workload and 8 ms of sleep), while the second set to 50%. Figure 2.9 shows the PSD for the two cases, computed in a time window of 40 ms. According to Fourier
analysis, the set of instructions with duty cycle 20% (top) shows the fundamental at around 100 Hz plus all its harmonics, while the one at 50% (bottom) correctly reports only the fundamental and the odd harmonics (even harmonics are not completely null due to the not exact 50% duty cycle). This example shows that our monitoring can really capture spectral properties of different workloads in execution.

The second set of benchmarks, reported in Figure 2.10, aims at demonstrating distinctive frequency-domain signatures of real bottlenecks (e.g., CPU or memory limitations) and scientific applications. Goal of this test is not to analyze in depth the reasons behind the peaks, but instead to show that different patterns emerge in the power spectrum with different workloads, which can be used as input features for DNN algorithms.

In particular, comparing the first four plots we can clearly see four different patterns (peaks highlighted with dark/light circles to indicate stronger/weaker magnitude): the first plot portrays the PSD of the compute node in idle and reveals five main peaks (dark red circles) plus other weaker peaks (light red circles) spread in the entire bandwidth 0–12 kHz (richer activity in 0–4 kHz); notice that

Figure 2.9: PSD of an ISR at 100 Hz with different sets of instructions.
these main peaks persist also in all other benchmarks; the second and third plot depicts respectively a memory bound and a CPU bound synthetic benchmark, where the former is bound in the SDRAM, while the latter is stuck in the CPU ‘front-end’ process (i.e., phase where instructions are fetched and decoded into operations - it differs from the CPU ‘back-end’ process where instead the required computation is performed); these two benchmarks report a rich activity up to 6 kHz and are almost flat for frequencies above; moreover, they can be clearly distinguished from their pattern (three main peaks in the tested memory bound application vs. ten main peaks in the CPU bound). It is noteworthy that, while measuring only a power consumption coarse-grain value would not be enough to discern any difference between the two bottlenecks, DiG (w.r.t. SoA monitoring systems) can detect spectral components associated to different usage of architectural resources in the two benchmarks. Finally, the fourth plot shows a real scientific application (i.e., Quantum Espresso - QE [54]) and reports four main peaks more with respect to idle (dark yellow circles) and rich activity in all the spectrum.

With the next three plots in Figure 2.10, we demonstrate that we can appreciate the activity of short regions of code. More in detail, we report the PSD of the compute node when it is running sets of instructions at a desired frequency and duty cycle (i.e., pulse train of instructions where we alternate high load computation phases with idle phases). The black circles in the plots highlight we can capture the activity of SW routines running roughly every 150 µs, 110 µs and 90 µs, with a respective duration of 75 µs, 55 µs and 45 µs (i.e., routines at 6.5 kHz, 9 kHz and 11 kHz with 50% duty cycle between idle and computation).

Finally, we report in the last three plots of Figure 2.10 two use-cases of AD in the compute nodes. In particular, the first plot corresponds to a case of misconfiguration, where we disabled in the system the dynamic tick. This is enabled by default in Linux OS to make the system more energy efficient (i.e., the kernel can save power when idle because it does not have to wake up regularly just to service the timer tick). Comparing the PSD of the system in idle with the dynamic tick enabled (first plot) with the one without dynamic tick we can clearly see the peak at 1 kHz (frequency of the static tick) and all its harmonics (at multiples of the fundamental) till 11 kHz. We envision
this kind of high resolution monitoring along with edge ML analytics to help on catching anomalies in next generation of DCs.

Figure 2.10: Example of PSD patterns of real bottlenecks and applications.

Another interesting scenario for detection of anomalies is related to the detection of cyber attacks. Network security is a crucial challenge in DCs and cloud infrastructures, to prevent attackers from getting access into the system and steal sensitive data or illegally use computing resources [55,56]. Before intruding into the system attackers need to
gather information about the target machine and its running services, and thus about vulnerabilities that can be exploited. This is called scanning phase, and one of the most popular tools for port scanning is *NMAP* [57].

The use-case scenario is an attacker that tries to collect information about the front-end node of DCs / clouds to get access into the local network. Thus we run NMAP from a remote computer (outside the local network) with the OS detection mode enabled, which means we want to understand open ports, running services and OS of the front-end node. The scanning attack requires around 10 seconds and the last two plots of Figure 2.10 show the PSD of the DiG power measurements when monitoring the front-end node. With the goal to show that different phases of NMAP correspond to different patterns of PSD, which are also different from the PSD of the system in idle, we report the attacked node at second 2 and second 8, on the first and second plot, respectively. As can be seen, results show 2 different patterns: the plot at second 2, with regards to plot at second 8, reports main peaks only up to 5 kHz and is almost flat for the frequencies above; instead, the plot at second 8 reports main peaks spread in the entire bandwidth (*i.e.*, 1, 2, 3, 3.5, 4, 7, 8, 9, 10.5 kHz). This pattern recognition analysis, based on high resolution power measurements, can be used in future works to exploit ML classifiers running on the edge and correlate this information with SoA signature-based IDS (*e.g.*, SNORT) to help on preventing from intrusions.

### 2.3.4 Edge ML Benchmarking

In this section we show the performance of ResNets [41] running on the embedded monitoring platform with different layers and sizes on a dataset of pre-computed PSDs of 2048 points in frequency each (*i.e.*, 4096 B per spectrogram, which correspond to a time window of 40 ms), to show an example of the capability of DiG on carrying out edge ML inference. In particular, we use TensorFlow inference compiled for ARM architecture to exploit the NEON SIMD accelerator.

Figure 2.11 reports the results of the benchmarks, where the x-axis corresponds to the chosen batch size (*i.e.*, number of test images per iteration) and the y-axis to the processing time per image (milliseconds). Results show that the best trade-off is between batch sizes 3 and 5,
which can give an improvement up to around 10 ms w.r.t. using the same ResNet with no batch mode enabled. For bigger batch sizes the improvement is negligible (in the best case up to 1 ms). Moreover, as a matter of comparison we have run a ResNet with 8 layers and channels \{8, 8, 16, 32\} both exploiting the NEON accelerator (gray triangles) and without using it (purple triangles), and results show a 3.8× improvement when using NEON, which is a relevant processing time when handling high resolution measurements and live analysis.

Finally, considering a time constraint of 40 ms for running real-time PSDs on the edge (4096 B per spectrogram), results suggest that ResNet with 8 layers and channels \{8, 8, 16, 32\} (and below), but also 14 layers and channels \{4, 4, 8, 16\} (and below), are suitable for running on high resolution power measurements, while larger ResNet size and layers can be used together with performance measurements acquired at a lower rate. It should be noted that in our benchmarks we did not use any optimized ML framework for our embedded platform, such as ARM NN SDK [45] or TensorFlow Light [52], which we believe would further improve these results.

Figure 2.11: ResNet benchmarking with TensorFlow on BBB.
2.4 Related Work

Existing off-the-shelf methods to measure power and performance of compute nodes in DCs 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 [37], while examples of out-of-band solutions are IBM Amester [10] and the two standards IPMI [39] and Redfish [58] (i.e., new protocol for managing DCs HW, that fixes the security vulnerabilities of IPMI [10]). All these built-in tools allow a fine grain per-component monitoring (i.e., based on HW performance counters and up to 1 s for IPMI, 1 ms for RAPL and 250 µs for Amester [2, 10, 37]), but not high-resolution power monitoring (i.e., covering the entire signal bandwidth - tens of microseconds).

Pushed by the growing interest in fine-grained power monitoring, industrial and academic researchers are providing custom solutions for DCs. Examples are HDEEM [2], PowerInsight [11] and HAEC [36]. The first two systems provide power consumption measurements up to millisecond time resolution, while the last one has a much more fine grain insight, with a sampling rate up to 500 kS/s. All these custom solutions focus on only monitoring the power consumption (i.e., no performance knowledge) and send all the measurements to a centralized monitoring unit for analysis. Thanks to them, new opportunities for research on energy efficiency and other challenges are now possible, but going toward high resolution measurements this kind of monitoring design entails scalability issues (e.g., as discussed in [36], HAEC is suitable for high resolution monitoring in just a node, but not for an entire cluster).

Comparison with SoA: In our system (i.e., DiG) we combined all these features to enable research on several challenges for analytics, automation and control of DCs, with a highly-flexible monitoring platform: (i) we work completely out-of-band (i.e., no impact/perturbation on the computing resources); (ii) we collect all performance counters and (iii) the full power bandwidth at the plug (i.e., sampling at 50 kS/s) (iv) with high precision (i.e., below 1% - σ); (v) we provide highly synchronized measurements (i.e., sub-microsecond) for a detailed correlation of the activities within the cluster; (vi) we leverage the monitoring between edge and a centralized
unit, by exploiting dedicated embedded computers to collect measurements (they have complete knowledge of the status of their node) and have the possibility to carry out both edge and cluster-level analytics; (vii) the system is scalable (thanks to our flexible design, based on edge monitoring agents and a robust and scalable protocol - MQTT - to the centralized monitoring, where we analyze data at a lower rate), (viii) technology agnostic (i.e., tested on ARM, Intel and IBM) and (ix) low cost (i.e., no motherboard redesign required).

2.5 Conclusion

This chapter reports on the design of a novel monitoring infrastructure - namely DiG - that enables real-time high-resolution profiling and analytics of DCs, for their automation and control. Main design choices include complete out-of-band monitoring of power and performance, with a dedicated embedded computer per node to perform edge analysis, and a custom power sensor at the plug for high-resolution and high-precision measurements. We report (i) architecture design choices of both HW and SW, (ii) monitoring platform performance, (iii) an extensive set of tests based on Fourier analysis to show the high resolution monitoring insights and (vi) a campaign of benchmarks of ML inference, running on the embedded computer, to provide an overview of the real-time edge analytics capabilities of DiG.
Fine-Grain Synchronization in Data Centers

Tight time synchronization is important to address several challenges in today and future DCs and HPC systems. Among the many, (i) co-scheduling techniques in parallel applications with sensitive bulk synchronous workloads (e.g., MPI_Barrier, MPI_Alltoall), (ii) performance analysis tools and (iii) autotuning strategies based on high-resolution power and performance monitoring systems, are three examples where synchronization of few microseconds is required [26]. Previous works in literature report custom solutions to reach this performance without incurring the extra cost of dedicated HW. On the other hand, the benefits of using robust standards that are widely supported by the community, such as NTP and PTP, are evident. With today’s SW and HW improvements of these two protocols and off-the-shelf integration in SoA HPC/DC servers (e.g., enhancements on synchronization SW daemons, built-in PTP HW support, high stability of commodity oscillators) no expensive extra HW is required anymore, but an evaluation of their performance in DC clusters is needed. Our results show NTP can reach on DC nodes an accuracy
of 2.6\,\mu s and a precision below 2.7\,\mu s, with negligible overhead. These values can be bounded below microseconds, with PTP and low-cost commercial off-the-shelf switches (no needs of Global Positioning System - GPS - antenna). Both protocols are also suitable for data’s timestamps in SoA out-of-band HPC / DC monitoring infrastructures, like DiG [1]. We validate their performance with two real use-cases, and quantify (i) synchronization’s drift over time, (ii) scalability and (iii) CPU overhead. Finally, we report SW settings and our low-cost network configuration to reach these high precision synchronization results.

3.1 Introduction

Time synchronization is a key factor for any distributed system [59]. This is also valid for DC / HPC clusters, wherein the trend toward increasing the node count to exploit application concurrency and parallelism, it becomes crucial to ensure a tight level of time agreement [20, 22, 23]. As reported in [23], this is needed for several reasons, whose the ultimate aim is to improve the application execution time and energy efficiency of the system (a.k.a. \emph{Time-to-Solution} - TtS - and \emph{Energy-to-Solution} - EtS). As an example, a fine-grain synchronization (order of few \mu s) is fundamental in (i) wall-clock runtime of parallel applications that exploit the Bulk Synchronous Parallelism (BSP), which is a parallel programming model where the progress of concurrent processes is driven by synchronization points (synchronization barriers). As shown in [60], by mean of a coordinated scheduling (co-scheduling) strategy, it is possible to achieve a speedup of 285\% for sensitive bulk synchronous workloads (\emph{i.e.}, with frequent synchronizing collectives based on Message Parallel Interface - MPI -, such as MPI\_Barrier or MPI\_Alltoall).

Other examples are (ii) code development / debugging and (iii) high-resolution monitoring [1, 23]. Works in [19–21] show that synchronization performance in the order of few microseconds is important to carry out an efficient analysis of applications’ performance, as otherwise would not be possible to achieve a correct ordering of the events in concurrent processes running on multiple nodes. Moreover, new generation of high-resolution monitoring systems for DCs / SCs
\subsection{Introduction}

(e.g., HDEEM, HAEC, DiG [1, 2, 12]) require a synchronization at least comparable to their sampling rate, to be able to correlate the measurements with application phases, and therefore to get a detailed profiling picture over time, needed for a better usage of the resources [30].

Time synchronization in distributed systems is normally supported by network synchronization protocols. In these protocols, all the nodes are kept synchronized against a time reference assumed as true time. In this context, accuracy refers to the amount of shift between the time reference and the mean of the time measurements ($\mu$), and precision to the standard deviation ($\sigma$). This is represented in Figure 3.1.

![Figure 3.1: Accuracy and precision of a set of measurements.](image)

Over the past years two standards have emerged and are widely adopted: NTP [61, 62] and PTP [63] (a.k.a. IEEE 1588). The former targets Wide Area Networks (WANs), where it typically achieves an accuracy of milliseconds, but - as documented in the standard [62] - it can reach much better performance within fast Local Area Networks (LANs). PTP instead targets LANs and can synchronize devices with sub-microsecond performance, thanks to dedicated HW support for synchronization.

In particular, within the context of HPC systems and DCs, NTP is today the most used protocol [23], but - as reported in a recent survey on leadership-class supercomputing centers [23] - it is usually set with default configurations, obtaining a synchronization accuracy that exceeds acceptable uncertainty requirements for many potential
applications (jitter of tens/hundreds of milliseconds). To achieve microsecond synchronization without incurring extra cost for dedicated HW support, such as with PTP, several works in literature proposed alternative SW solutions [19–22]. However, the advantages of using robust standards that are widely supported by the community- like NTP and PTP- are evident. In addition, (i) today’s HW/SW implementations improvements of these two protocols, along with (ii) the off-the-shelf integration of PTP HW in SoA DC servers and IoT platforms commonly used for embedded monitoring solutions, encouraged the motivation behind our work, as the performance of both protocols need to be evaluated with Commercial Off-The-Shelf (COTS) HW and a proper configuration for DC/SC clusters. A key concept in our study is that synchronization is performed within the LAN of the DC/HPC infrastructure. This allows working in a corner case for NTP, where it can achieve its best performance. Moreover, it is not necessary to achieve high accuracy with respect to the absolute time reference (e.g., by mean of an atomic clock or a GPS antenna), but instead that all devices are highly synchronized between them.

**Contribution:** In this chapter, we evaluate the NTP/PTP performance on a real DC in production (i.e., D.A.V.I.D.E. [3]), including its IoT-based monitoring infrastructure (i.e., DiG [1]) and networking devices (i.e., switches). Major contributions of this work are:

1. the demonstration that with a proper configuration, both synchronization protocols - NTP and PTP - can achieve the target synchronization of sub-milliseconds needed for many potential applications in today’s DCs/SCs. This is done through extensive measurements of synchronization’s performance (accuracy and precision) in a SoA HPC cluster and its out-of-band monitoring infrastructure. Moreover, we carry out a validation of the results in two real use-case scenarios. We will show that NTP can achieve in computing/front-end nodes an accuracy below 2.6μs and a precision below 2.7μs. Moreover, we observed that 99% of the measurements stay within 8.9μs of skew from the time reference. Regarding PTP, we measured in the computing/front-end nodes an accuracy within 0.2μs and a precision within 0.8μs. Furthermore, 99% of the measurements skew from the time reference of
3.2. **DATA CENTERS AND FINE GRAIN SYNCH**

no more than 2.2µs. It is noteworthy that it is possible to achieve this performance with low-cost COTS HW, which generally is already built-in in new SCs (no extra cost is required). Finally, we observed that the synchronization performance (accuracy and precision) achievable in the IoT devices used in DiG (best-in-class DC/SC high-resolution monitoring system) is below their monitoring sampling period (20µs), thus is suitable to correlate the monitored metrics with applications running in the compute nodes;

2. a detailed investigation of synchronization drift, scalability and CPU overhead of the protocols’ SW daemons distributed in the cluster devices;

3. a comprehensive description of the network topology and SW configuration used to achieve our performance in D.A.V.I.D.E., to be helpful for other leadership-class supercomputing centers.

Moreover, we note that the synchronization results obtained in this chapter are of interest not only for HPC systems and DCs, but also for any other distributed system based on COTS HW, where is important a fine grain synchronization within LAN (e.g., cloud, smart grids, and Wireless Sensor Networks - WSN - based on IoT computers in industrial environments [64, 65]).

**Outline:** We start by introducing in Section 3.2 the importance of fine grain synchronization in DC/HPC clusters, taking as an example D.A.V.I.D.E. and its SoA monitoring infrastructure DiG. In Section 3.3 we outline the related work. We continue in Section 3.4 by summarizing some background information on NTP and PTP, and report in Section 3.5 our performance measurements results, along with an *how-to* guide that outlines our network topology and configuration. Finally, we conclude the chapter in Section 3.6.

### 3.2 Data Centers and Fine Grain Synch

Data centers are nowadays remarkably increasing in complexity, and effective methods for improving their efficiency are at the basis of their growth. This is even more evident in high-performance computing,
where on the race toward exascale computing, energy efficiency is garnering a key rule [2, 14]. With this aim, industrial and academic researchers are focusing on the development of new HW technologies along with a more efficient usage of the resources.

At the basis of a better usage of the resources, techniques for exploiting more efficient concurrency on applications that run on many cores per chips (i.e., CPUs/Accelerators) and multiple nodes per clusters are of crucial importance [22, 23]. Already several works in literature show that an efficient concurrency, which in ultimate leads to improvements in TtS and EtS, can be achieved by (i) coscheduling techniques [23, 60], (ii) optimized coding - which requires application’s performance analysis tools for debugging [19–21, 23] -, and (iii) runtime autotuning techniques, based on power and performance monitoring, to promptly react to workloads changing and events (e.g., by mean of Power Capping and Dynamic Voltage and Frequency Scaling - DVFS; Approximate Computing; advanced cooling control policies) [1, 12, 30, 66, 67]. All these strategies require a proper level of synchronization within the DC / HPC cluster. In particular, coscheduling and performance analysis tools demand a synchronization accuracy in the order of few microseconds (4–7 μs [19–21]), while runtime autotuning, based on power and performance measurements, needs a synchronization accuracy lower than the sampling period of the monitoring, to be able to correlate application phases and events with physical and architectural metrics.

Figure 3.2 shows an example of a SoA energy-efficient HPC cluster, namely D.A.V.I.D.E. [3, 14], and its high-resolution monitoring infrastructure, DiG [1]. As reported in Chapter 2, by means of DiG we can collect power and performance of the several cluster components, from different heterogeneous sensors, in real-time and 24/7. However, to meet the accuracy synchronization constraints required by the aforementioned optimization strategies, it is essential that the clocks within compute nodes and monitoring devices (BBB) are tightly synchronized. These clocks can then be used for coscheduling strategies in applications running on multiple nodes, and to take timestamps of both (a) application phases / events from the compute nodes, and (b) measurements acquired by the monitoring devices. In particular, Figure 3.3 depicts an example where these constraints are not met during the system performance monitoring:
3.3 Related Work

Widely used approaches to address time agreement in distributed systems consist of network synchronization protocols, such as NTP and PTP [62,63]. The former is commonly used with default settings to synchronize clocks over WANs (synchronization every 1024 s), where it can reach an accuracy of milliseconds against the absolute time reference. The latter instead is used within LANs, for applications where high synchronization performance is required (sub-microsecond
accuracy against the absolute time, by mean of a GPS). However, PTP requires dedicated HW which only recently started to be integrated off-the-shelf in common servers and embedded computers.

To achieve in the past years sub-millisecond synchronization, without incurring extra cost for dedicated HW, prior works in literature proposed alternative solutions. One technique which targets events synchronization in HPC parallel applications consist of post-mortem ordering [19–21]. This approach is useful for event tracing, where application phases and their respective monitored metrics can be retroactively synchronized. However, it is not suitable for runtime synchronization, which is crucial to improve TtS and EtS of HPC applications [23].

Several studies propose different solutions to address this problem. In particular, Jones et al. [22] worked on a NTP-based synchronization scheme for MPI applications that provides time agreement up to 2.29µs of mean value (min −32.0µs, max 32.1µs, span 64.1µs). This solution is suitable for runtime synchronization of parallel applications (it exploits the PMPI interface of MPI), but not for autotuning techniques based on power and performance monitoring (i.e., where accurate timestamping of the measurements is required) [2,12,30]. Work in [68] shows how to achieve a runtime synchronization among computers.

Figure 3.3: Wrong correlation of events due to the high jitter in the clock synchronization of the monitoring devices.
3.3. RELATED WORK

within a LAN, up to ten microseconds by mean of a SW-based scheme called TSCclock. Even if this solution is suitable for several applications within distributed systems, the provided accuracy is still not enough for DC / SC synchronization constraints (i.e., few microseconds).

A more recent solution that provides higher synchronization performance is White Rabbit [69,70], which extends PTP with dedicated HW, firmware and SW, and can synchronize nodes with sub-nanosecond accuracy and picosecond precision. However, as it is over requirements for DC / SC synchronization’s constraints, also considering the extra HW needed, we focus our analysis only on NTP and PTP.

With today’s improvements on HW / SW technologies for network synchronization protocols (e.g., enhancements on SW daemons implementation, built-in HW support for PTP, high stability of commodity oscillator HW), a benchmarking study of NTP and PTP performance on SoA DC / SC clusters (including new generation of out-of-band high resolution monitoring systems) is missing, and as shown in Section 3.5 can reveal extremely useful results. To the best of our knowledge, only work in [71] carried out a similar evaluation, but only for PTP and targeting different requirements. In particular, they focused on distributed systems where high-performance synchronization against the absolute time reference was required. This is not needed to address DC / SC challenges (such as improving energy efficiency and execution time of applications) and, at the same time, can be an obstacle for supercomputing centers to install a GPS antenna or an atomic clock. Moreover, their evaluation included the measurements of network cables delay, to further refine their accuracy results when this is asymmetric between nodes. Such a delay is usually small (and often negligible) within the LAN of DCs / SCs [23], and considering large scale infrastructures, it would add a worthless extra effort to be measured. The goal of this analysis is to provide worst-case bounds on synchronization performance (i.e., including possible errors due to small asymmetric network drifts), suitable to be used off-the-shelf by researchers in common DCs / SCs.
3.4 Time Synchronization Protocols

In this section, we describe the key concepts of the two most used synchronization protocols, namely NTP and PTP. More in detail, we first introduce (a) their network topologies and (b) message synchronization patterns, and focus then on (c) the different timestamping methods and (d) main sources of jitter. According to the standard terminology, the terms client and server are usually used within the context of NTP, while slave and master within PTP. However, as there is no conceptual difference between them, to help in reading, we will often use the terms master and slave for both protocols (master corresponds to the server and slave to the client).

3.4.1 Network Topology

Figure 3.4 shows the typical network topologies of NTP and PTP. Both protocols use a hierarchical master-slave network. Regarding NTP, each level (stratum) can range from 0 to 15, where the zero represents the time reference (e.g., atomic clock, GPS), and lower strata are synchronized over the network to the respective upper layer (e.g., stratum “n” to stratum “n-1”). Moreover, the NTP algorithm sets the synchronization paths by a shortest-path spanning tree, and devices in the same stratum can also peer each other to stabilize the clock.

PTP uses a similar master-slave hierarchy within a LAN, which is managed by the Best Master Clock algorithm running on each node (it decides who is master and who the slave, based on best accuracy). According to the PTP specifications, end-nodes (Ordinary Clock - OC) are devices with one PTP port, which can be either master or slave. A particular OC, which is the main time reference for the whole PTP network, is the Grandmaster clock. Instead, networking-nodes (e.g., switches) can be of two kinds, Transparent Clock (TC) or Boundary Clock (BC). The goal of both is to compensate the jitter introduced by general networking devices. TC corrects it by measuring the time taken from a PTP packet to transit the device: it is “transparent” from the clock’s point of view, as it does not have PTP ports that act as master or slave for other nodes. Instead, this is done by the BC that has several PTP ports. As illustrated in Figure 3.4.2, in this
node one port (in a specific moment) is in a slave state, while the others are masters for one or more slaves.

### 3.4.2 Message Synchronization Pattern

Figure 3.5 depicts the message synchronization pattern of both protocols. The idea behind NTP is that each client regularly polls a cluster of servers to synchronize their clock by computing round-trip delay ($\delta$) and time offset ($\theta$). According to the standard, the latter is the time offset of the server relative to the client ($T_{\text{server}} - T_{\text{client}}$).

The message synchronization pattern involves only a pair of messages: an $\text{NTP}_{\text{Req}}$ from the client to the server (query) and backward an $\text{NTP}_{\text{Resp}}$ (reply). As soon as the server receives the query containing the client’s transmission timestamp $t_1$, it generates a reception timestamp $t_2$ and sends it back together with its transmission timestamp $t_3$ via the reply message. Therefore, the client notes its reply reception time $t_4$, and can finally compute both $\delta$ and $\theta$ by:

$$\delta = (t_4 - t_1) - (t_3 - t_2)$$  \hspace{1cm} (3.1)
\[ \theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \] (3.2)

where equation 3.2 is calculated:

- assuming a symmetric network delay between client (c) and server (s), \( \delta_{c\rightarrow s} = \delta_{s\rightarrow c} = \frac{\delta}{2} \),
- and by summing the server-client offset obtained by the NTP_Req message \( \theta = t_2 - (t_1 + \frac{\delta}{2}) \), and that obtained by the NTP_Resp message \( \theta = t_3 - (t_4 - \frac{\delta}{2}) \).

Equations 3.1 and 3.2 are also valid for PTP, where the one-way delay \( \frac{\delta}{2} \) is used instead. Similar to NTP, the idea is to continuously exchange messages between master and slave ports, to calculate offset and one-way delay. Figure 3.5.2 describes the synchronization message pattern using the delay request-response mechanism\(^1\). Four packets are involved: Sync, Follow_Up, Delay_Req and Delay_Resp. Sync and Delay_Req are event messages, as an accurate timestamp is generated at both transmission and receipt. The other two are general messages, as no timestamp is required. That means only the event messages contribute to the actual computation of \( \theta \) and \( \frac{\delta}{2} \), while the other

\(^1\)A second option would be the peer-to-peer delay mechanism, which is less flexible and thus left for possible future works.
two packets are used as support. Indeed, the master periodically sends Sync packets to the slaves, taking note of their transmission time $t_1$. These times are then delivered within the general messages Follow_Up. The reason behind this mechanism is that to include $t_1$ in the Sync packet itself without loosing in performance, dedicated HW is necessary (one-step clock). As soon as the slave receives the Sync message, it notes the reception time $t_2$ and conveys the Delay_Req packet to the master, writing down its transmission time $t_3$. In the last step, the slave receives from the master the Delay_Req reception time $t_4$ by the Delay_Resp message, and can finally measure both offset and mean propagation time.

Finally, few further considerations on the PTP synchronization pattern are essential to understand its network scalability, which will be later examined. The key concept is that the PTP communication is based on a multicast messaging model (or potentially a unicast messaging model). In other words, each PTP packet sent by any PTP port has a destination’s multicast address, which means that it will be received by all the PTP ports in the network segment. For packets that are specific per device (e.g., the Delay_Resp message, which is specific for the slave that delivered the Delay_Req), then the clock identity of the destination’s device is specified within the message. This allows PTP to reduce the network load on the master side.

### 3.4.3 Timestamping Methods

One of the main sources of jitter that directly affects the time synchronization accuracy is due to the uncertain processing time of the protocol stack during the transmission of the packets. Ideally, the packet is forwarded right after the timestamp is generated, with no related timing delay. However, in the real implementation, this delay depends on the layer of the protocol stack where the timestamp is done: the higher the distance from the physical layer, which corresponds to the real point where the packet is transmitted, the greater the jitter introduced in the timestamp. Figure 3.6 depicts the difference between ideal and real implementation, and the several points where the timestamp can be done.

The two possibilities are the SW timestamp and the HW timestamp. The latter is available only for PTP HW-enabled devices and is actually
one of the main advantages that PTP has over NTP. Indeed, the SW timestamp is the least accurate option, as it is based on a SW clock that runs in the kernel, namely the system clock. It can be generated when a packet reaches the application layer or, to reduce the delay in the protocol stack, in the device driver. The system clock works with timer interrupts and keeps the time by reading the CPU register which counts the number of clock cycles since the last reboot (e.g., the Time Stamp Counter - TSC - on Intel x86 processors). The system clock has not to be confused with the battery-powered clock, also known as Real-Time Clock, and present in most Linux devices. This clock, indeed, is used only to keep track of the time when the system is turned off and later to initialize the system clock at boot time. In Linux systems, NTP is usually implemented by a daemon running in user space, the ntpd, which continually updates the system clock. The kernel will then correct the real-time clock drift, usually at a lower frequency.
To minimize the jitter introduced by the processing time of the ISO/OSI layers [72], PTP introduces the idea of HW timestamping support. As described in Figure 3.6, the PTP HW Clock (PHC) subsystem, takes advantage of the Media Independent Interface (MII) to detect PTP frames and provide timestamps with an accuracy close to the physical layer. A widely used implementation for PTP is the Linux PTP Project [13], which involves two user-space applications, namely ptp4l and phc2sys. Both take advantage of the kernel support for the PHC subsystem by using the clock_gettime family of system calls. While ptp4l is the actual implementation of PTP (it implements both boundary and ordinary clocks), phc2sys is used to synchronize the PHC to the system clock.

3.4.4 Possible Sources of Jitter

In the following, we summarize the main sources of jitter that affect the time synchronization performance of the two protocols. Of course, the main difference is the method used to timestamp, as the HW timestamping reduces the processing time delay to the OSI physical layer. Therefore, remaining sources of jitter are mainly attributed to [69, 73]:

1. the delay on the physical link, which includes
   
   (a) the physical channel (i.e., asymmetry of the network propagation delay in the two directions),
   
   (b) the use of general networking devices. In this case, PTP can be improved by PTP HW-enabled networking devices, while NTP by replacing general switches with more recent models that can be used as NTP time server (reducing the number of hops to the clients).
   
   (c) for PTP only, the physical distance between the PHY layer and the PHC within the device.

2. the HW properties of the clock (i.e., rate and stability of the oscillator, which result in a limited HW resolution and precision of the timestamps, respectively). More in detail,
(a) for NTP, the rate and the stability of the oscillator used for the system clock interrupt timer.  

(b) for PTP, the rate and the stability of the PHC.

### 3.5 Experimental Results

In this section, we evaluate the performance of NTP and PTP on a SoA HPC cluster, namely D.A.V.I.D.E. [3], in terms of (i) accuracy, (ii) precision, (iii) worst-case bound and (iv) scalability. Measurements are performed both on the compute nodes and on the out-of-band monitoring system of the cluster (DiG [1,14], depicted in Figure 3.2). To interconnect all devices, we used the **Cisco IE 3000** [74], a low-cost COTS switch, suitable for industrial installations and PTP HW-enabled.

![HPC Synchronization Testbed](image)

**Figure 3.7**: Testbed used for NTP / PTP performance evaluation on compute nodes of D.A.V.I.D.E. and its out-of-band monitoring system (DiG).

Figure 3.7 (top) illustrates the testbed and synchronization flow of the SW daemons. Regarding NTP, we used ntpd to synchronize the system clock of the slaves directly to the system clock of the master node (NTP server), while for PTP we used ptp4l to adjust the PHC of the slaves to the master PHC and exploited phc2sys to constantly...
update the respective system clocks (on both master and slaves). Basing on the results obtained in [13], we tuned their synchronization messages rate to the optimal working point, which corresponds to 0.125 Hz (8 s) of NTP polling rate, and 1 Hz and 12 Hz for ptp4l and phc2sys, respectively. Moreover, for PTP we configured the switch in transparent mode as it provides a better degree of accuracy [74]. We remind that in the target scenario (DC/SC clusters) it is not necessary to achieve a high level of accuracy with regards to the absolute time reference (e.g., atomic clock or a GPS), but instead that all devices - BBB and compute nodes - are highly synchronized between them, reason why we can use internal servers as time reference and then synchronize these servers to the external world with lower accuracy.

In the following, we will first evaluate (a) the synchronization between compute nodes, also showing an example with a parallel application, and then (b) the synchronization among monitoring devices, where we also verify the correlation performance between the out-of-band measurements and an application running on the compute nodes. Finally, for both protocols (c) we will show the time drift when the nodes are synchronized with a larger polling period (worst performance), (d) quantify their scalability and CPU load on the devices, and (e) report a short how-to guide to setup the SW daemons and build the synchronization network to reach the performance reported in this chapter.

### 3.5.1 Synchronization among DC/SC nodes

To measure the synchronization between compute nodes, we developed a synthetic benchmark based on OpenMPI [75], with one MPI process per node, as depicted in Figure 3.7.a. The goal is to generate timestamps on a set of queried nodes after a triggering event, and observe the clock’s skew of the clients from the time reference. The benchmark first gets the hostname of the machine where is executed, then waits for an MPI.Barrier to align the timestamps generation events between nodes, and finally produces the timestamp using the clock_gettime function.

As in common SC installations the MPI inter-node latency for small messages is in the order of few microseconds (i.e., accuracy within 4–7 µs [19–21]), it is crucial that clock synchronization of compute nodes is in this range or below.
We run the benchmark to generate 10 thousand timestamps over several hours, on 4 nodes of D.A.V.I.D.E. synchronized first with NTP and then with PTP. Figure 3.8 (top) shows the histogram of the obtained master-slave offset, where the green line on the zero represents the time reference (NTP/PTP server), and the y-axis reports the relative number of occurrences (number of observations in bin / total number of observations). Results show a clear Gaussian trend, for both NTP and PTP. In particular, accuracy ($\mu$) and precision ($\sigma$) are below 2.6 $\mu$s and 2.7 $\mu$s in NTP, respectively, and below 0.2 $\mu$s and 0.8 $\mu$s in PTP. To observe how the percentage of samples skews from
the reference time, we report in Figure 3.8 (bottom) the Cumulative Distribution Function (CDF). NTP bounds 75% of the samples within 4.4 $\mu$s, while PTP within 0.8 $\mu$s. These values increase up to only 6.7 $\mu$s and 1.6 $\mu$s for 95% of the occurrences, and to a maximum of 8.9 $\mu$s and 2.2 $\mu$s for 99%. It should be noted that D.A.V.I.D.E. exploits an EDR InfiniBand connection (100 Gbit/s) between nodes with a minimal latency of around 0.5 $\mu$s [76–78]. As this latency affects the MPI triggering event to generate timestamps on the nodes, both synchronization protocols would perform even better than what we measured (we are providing worst-case bounds). In light of these results, the time agreement provided by both protocols (i.e., 2.6 $\mu$s of NTP accuracy, and 0.2 $\mu$s of PTP accuracy) is suitable to appreciate and correlate parallel application phases running on different nodes, as well as to correlate them with in-band system performance metrics timestamped up to few microseconds granularity.

In particular, the accuracy provided by NTP on compute nodes is an important off-the-shelf achievement for DCs/SCs, researchers and developers, as it can push the boundaries of SoA techniques for profiling, debugging, scheduling and maintenance, and thus for improvements on DC/SC systems and applications performance [19–23]. To show one of the benefits of such a fine grain synchronization, we report on a common scenario that can be found by HPC application developers and users, when they have to balance the workload among MPI processes. The example consists of a scientific parallel application, namely QE [54], which involves several MPI_Alltoall. Let’s assume the user wants to evaluate the unbalance in the communication phase of the application. QE involves several MPI_Alltoall on which all MPI processes wait on a barrier for the other processes to end their computation. We want to compute the time each MPI process takes to exit the barrier after the last process reaches it. We run QE on two compute nodes - one executing an MPI root and one an MPI slave - both synchronized to a time reference on a third node (i.e., NTP server).

Figure 3.9 (top) shows the offset of the MPI processes at the exit of the barrier, for each MPI_Alltoall call in the application. In particular, the blue curve reports the offset when the nodes are synchronized with the default NTP polling rate (i.e., 1024 s, which is the one commonly used): the offset is in the order of few milliseconds and is linearly improving over time up to 0.75 ms. This result can be misleading, as it
could mean there is an increasing communication unbalance (i.e., MPI slave always leaves the barrier first) and latency for which it would be worst to tweak the code to alleviate its cost.

However, this is not the case, as if we keep the same workload but invert the nodes that are executing QE (green trace in Figure 3.9 - top), we obtain exactly the opposite behavior, where MPI root always leaves the barrier first. This is a clear problem of clock’s synchronization and is much more challenging to understand when facing with thousands/millions of processes in large-scale DCs/SCs. In particular, the clock of one node is head of time with respect to the other node, and the linear improvement of the offset is due to the NTP adjustments over time (jitter in the order of milliseconds because we are using the default polling rate). As shown by the two curves \textit{NTP 8s} and \textit{NTP 8s - Inv}, when we repeat the previous tests synchronizing the nodes with best settings (i.e., NTP polling rate of 8s), MPI processes exit the barrier with an offset roughly bounded within 80µs (zoom of the two curves in Figure 3.9 - bottom). Due to the microseconds granularity of MPI processes, this fine grain synchronization - that can be obtained off-the-shelf - can easily and drastically improve application’s debugging and tuning, crucial to increase the efficiency of HPC systems.

\subsection*{3.5.2 Synchronization of the Out-of-Band Monitoring System}

Accurate synchronization of the D.A.V.I.D.E. out-of-band monitoring infrastructure (DiG) is crucial to get high-quality profiling of the whole HPC system and running applications. As the maximum sampling period of DiG is 20µs, it is important that its synchronization accuracy is in this range or below to be able to correlate out-of-band measurements between them (i.e., the measurements acquired for each server would be perfectly aligned) and with application phases running on compute nodes. To evaluate the DiG synchronization performance, we used the same testbed of before - same switch and SW daemons on the embedded monitoring devices -, but with an external source of interrupts to get the same triggering event on all BBB (i.e., computing platforms on which DiG is built upon). In particular, we used a square wave signal connected to the General Purpose Input / Output (GPIO)
3.5. EXPERIMENTAL RESULTS

pins of the BBB. The GPIOs are handled by a custom device driver: as soon as the square wave goes high, an ISR in each BBB catches the event and generates a timestamp. To take into account the jitter related to the ISR processing time, we used another GPIO connected to an oscilloscope (Keysight DS0X3054T, depicted in Figure 3.7.b). After the timestamp is generated, we raise up the second GPIO and compare the ISR’s jitter of the BBB master with the one of each BBB slave, obtaining the $\Delta_{ISR}$, as outlined in Figure 3.10. Thus, to evaluate the master-slave offset ($MS_{off}$), we used the following formula:

$$MS_{off} = TS_s - (TS_m + \Delta_{ISR})$$ (3.3)

where $TS_s$ and $TS_m$ correspond to the system clock timestamps of the BBB slave and master, respectively. Moreover, the offset $\Delta_{ISR}$ is always referred to the master and can be either positive or negative depending on which clock is head of time.

For the scope of this test, we synchronized 2 BBB slaves to a BBB Master, used as time reference, and generated 30 thousand timestamps. Figure 3.11 (top) shows the master-slave offset, where as before the green line on the zero represents the time reference and the y-axis the relative number of occurrences. Results of NTP show an almost Gaussian trend with an accuracy within 14 $\mu$s and a precision below 10.7 $\mu$s. The Gaussian curve becomes much more tight for PTP, where we see an accuracy below 0.1 $\mu$s and a precision within 0.68 $\mu$s. To observe how the percentage of slaves’ timestamps skew from the master, we report the CDF in Figure 3.11 (bottom): 75% of the occurrences stay within 20 $\mu$s for NTP and 0.64 $\mu$s for PTP. These values increase up to 32 $\mu$s (NTP) and 1.52 $\mu$s (PTP) for 95% of the samples, and up to 36 $\mu$s (NTP) and 2.23 $\mu$s (PTP) for 99%.

Comparing these results with those obtained in the compute nodes, we can observe that PTP slightly improves its performance, while NTP is slightly worse (probably due to a cheaper built-in oscillator). However, we can state that both protocols are suitable to keep aligned the out-of-band power and performance measurements of D.A.V.I.D.E..

To verify the correlation performance of the DiG’s measurements with an application running on the compute nodes, we developed a custom benchmark based on OpenMP [79]. The benchmark simply
CHAPTER 3. SYNCHRONIZATION

gets a timestamp and stresses all CPU cores, while the nodes and the embedded monitoring devices are synchronized with NTP (i.e., the same testbed as before, see Figure 3.7). The result is visible in Figure 3.12, where the blue curve corresponds to the power consumption monitored by DiG (sampling period of 20 µs, represented by the stars) and the red line to the timestamp generated by the application, which highlights the instant of rising edge: if we look at it, we can see it is well aligned with the power consumption trace. With this test, we can finally validate both synchronization protocols (with our SW setting) to achieve a fine-grain synchronization in SoA DC / HPC clusters.

3.5.3 Linear Drift, Scalability and CPU load

Linear Drift. The synchronization performance previously presented are related to the best accuracy settings studied in [13]. As all the nodes in the network are based on free-running oscillators [70], decreasing the synchronization rate it follows in the growth of the master-slave time drift, which is bounded within two consecutive synchronization’s events. To give an idea of such a drift, we set on two nodes (master and slave) the maximum synchronization polling period for both NTP and PTP (i.e., ~36.4 h). We measured then the synchronization several times, over more than an hour. Results are reported in Figure 3.13, where we show in the y-axis the error bars (µ and σ) of the master-slave offset and in the x-axis the elapsed time since the first synchronization measurement. As can be seen, the drift due to the use of free-running oscillators is almost linear. Lastly, we note that in the two tests, the drift grows in a positive (NTP) or negative direction (PTP), and this depends on which clock was ahead of time during the initial synchronization.

Scalability. Looking at the message exchange patterns, we can quantify for both protocols how the master (bottleneck) scales with the number of connected devices. Regarding NTP, considering N clients, the master has to deal with 2N packets (a pair query-reply) per clock’s time update. In NTPv4, the query consists of 90 B (42 B for the header and 48 B for the payload), and the reply of 94 B (46 B for the header and 48 B for the payload) [62]. Therefore, with a polling period of 8 s the master data rate corresponds to 23 B/s per client.
3.5. EXPERIMENTAL RESULTS

About PTP, focusing on the Multicast communication model, the master has to handle $2 + 2N$ packets per clock’s time update. In other words, setting a polling period of 1s for all the synchronization messages, there is a fixed component of 180 B/s ($90 \text{ B/s for the Sync} + 90 \text{ B/s for the Follow Up}$), and a scalable component of 186 B/s per slave ($86 \text{ B/s for the Delay Req} + 100 \text{ B/s for the Delay Resp}$) [63]. It should be noted that PTP includes other general messages to handle the network: these messages are sent with lower frequency and mainly contribute to the fixed component, reason why they do not undermine the scalability. Such considerations are valid for network topologies with transparent clocks (or generic devices with no PTP support) between nodes. Indeed, by using boundary clocks, the master data rate would drastically decrease, as would correspond to master-slave point-to-point links.

Accordingly to the above considerations, both protocols are not critical in terms of scalability. As an example, considering a Fast Ethernet with a bit rate of 100 Mbit/s (e.g., 10 / 100 RJ45 on BBB), with the used settings an NTP server would use only ~0.000184% of the network bandwidth per slave, and a PTP master only ~0.001488%. In theory, using only the 10% of the bandwidth, an NTP server could handle up to ~54k nodes, while each PTP master port up to ~6.7k nodes. Moreover, using a Gigabit Ethernet, these values decrease of an order of magnitude, which in theory corresponds to handle up to ~540k nodes for each NTP server and ~67k nodes for each PTP master port.

**CPU Load.** To do not impact on the HPC cluster’s computing resources, it is important that the used configuration makes use of a low percentage of CPU. To evaluate the CPU load, we exploited the Linux *Top* program, which provides a dynamic real-time view of the running processes and system’s resources usage. Results show that ntpd slaves (which are the NTP daemons that run on the compute nodes and monitoring embedded devices) require a negligible overhead on the CPU (Top reports 0%). The same considerations are valid for PTP, were both daemons (ptp4l and phc2sys) have a negligible impact on the CPU of either compute nodes or BBB (always 0%, except phc2sys that on the BBB requires 0.7%).
3.5.4 How-To Guide - NTP/PTP Settings

In this final section, we report a short “how-to” guide, based on the experience gained in this work to achieve the presented synchronization performance. Figure 3.14 shows the network configuration with the respective polling rates of NTP and PTP. As to correlate parallel application phases and measurements, it is only important that all devices within the LAN are highly synchronized between them, and we can accept a lower accuracy to the absolute time (external world), we use few internal NTP servers that synchronize their system clock to a pool of external NTP servers (standard configuration of NTP) with the default polling rate of 1024 s. It must be noted that we use more than one internal server synchronized to the external world to achieve a more resilient configuration. All slave devices within the LAN are then synchronized with NTP polling rate of 8 s and default PTP rate of 1 s.

Table 3.1 summarizes the main settings for each daemon. In particular, for each NTP master within the LAN we show (i) how to set them to be the time reference for the slaves, and (ii) how to synchronize them to the pool of external servers to get the absolute time with lower accuracy. For ptp4l we used the default configuration, while for phc2sys we set the update rate to 12 Hz and the -w to wait for ptp4l to be in a synchronized state. Finally, during our synchronization tests we used (i) the do_gettimeofday() to acquire timestamps at kernel space in the embedded monitoring devices (i.e., BBB), and (ii) the clock_gettime() along with CLOCK_REALTIME to acquire timestamps at user space in the compute nodes.

3.6 Conclusion

High synchronization performance is at the basis of several challenges in today and future leadership-class computing centers. Among the many, (i) co-scheduling techniques in MPI applications with sensitive bulk synchronous workloads, (ii) performance analysis tools and (iii) autotuning techniques based on high-resolution power and performance monitoring systems, are three examples that require a few microseconds synchronization.
## 3.6. CONCLUSION

In this chapter we evaluated the two widely used network synchronization protocols, NTP and PTP (a.k.a. IEEE 1588), in a SoA HPC Cluster, namely D.A.V.I.D.E.. Our results show that NTP can reach an accuracy of 2.6 µs and a precision below 2.7 µs on compute nodes, while maintaining a negligible overhead. Such values can be bounded to sub-microseconds when using PTP and low-cost COTS switches. Moreover, we observed that 99% of the measurements skew from the time reference by not more than 8.9 µs on NTP, and 2.2 µs on PTP.

We then tested these protocols on embedded devices (i.e., Beaglebone Black) that are part of a SoA out-of-band power and performance monitoring infrastructure, namely DiG, and results show they are also suitable for data timestamping and correlation.

Finally, we quantify their (i) synchronization drift over time, (ii) scalability and (iii) CPU overhead, and present two real use-cases to validate their synchronization performance in the field. In light of the obtained results, we can state that both protocols provide a suitable degree of synchronization to address the aforementioned HPC challenges.

### Table 3.1
Summary of NTP / PTP settings.

<table>
<thead>
<tr>
<th>Daemon</th>
<th>Masters</th>
<th>Slaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntpd</td>
<td>- Synch to pool of servers (Default 1024s):</td>
<td>- Synch to LAN Masters (rate 8s):</td>
</tr>
<tr>
<td></td>
<td>pool IP-S0 iburst</td>
<td>server IP-M0 maxpoll 3</td>
</tr>
<tr>
<td></td>
<td>pool IP-Sn iburst</td>
<td>server IP-Mn maxpoll 3</td>
</tr>
<tr>
<td></td>
<td>- Set Node as LAN Master:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>server 127.127.1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fudge 127.127.1.0 stratum 1</td>
<td></td>
</tr>
<tr>
<td>ptp4l</td>
<td>- Default settings (rate 1s)</td>
<td>- Default settings (rate 1s)</td>
</tr>
<tr>
<td>phc2sys</td>
<td>- Rate 12 Hz</td>
<td>- Rate 12 Hz</td>
</tr>
<tr>
<td></td>
<td>- use -w, to wait until ptp4l is synchronized</td>
<td>- use -w, to wait until ptp4l is synchronized</td>
</tr>
</tbody>
</table>
Figure 3.9: (Top) Master-slave MPI_Alltoall offset, with NTP polling rate set at 1024s and at 8s; (Bottom) Zoom of the offset with NTP polling rate at 8s.
Figure 3.10: Oscilloscope’s window showing the delays between the triggering event and ISR processing time of each BBB \(i.e.,\) Delay_ISR_1 and Delay_ISR_2. 
CHAPTER 3. SYNCHRONIZATION

Figure 3.11: NTP / PTP performance on embedded monitoring devices: (top) master-slave offset; (bottom) CDF for worst-case bound.

Figure 3.12: OpenMP synthetic benchmark to validate the correlation’s performance between D.A.V.I.D.E.’s compute nodes and DiG.
Figure 3.13: Linear drift between the master and slave, due to free-running oscillators.
Figure 3.14: Proposed network configuration for HPC clusters.
Reliability is a cumbersome problem in HPC Systems and DCs evolution. During operation, several types of fault conditions or anomalies can arise, ranging from malfunctioning HW to improper configurations or imperfect SW. Currently, system administrator and final users have to discover it manually. Clearly, this approach does not scale to large scale SCs and facilities: automated methods to detect faults and unhealthy conditions are needed. In this chapter, we propose a novel method based on AE NNs trained to learn the normal behavior of a real in-production HPC system - i.e., D.A.V.I.D.E.- and it is based on coarse-grain measurements (i.e., data aggregated in 5 min intervals), which means it can exploit our centralized-AI infrastructure, namely ExaMon [16] on top of DiG [1]. Moreover, the method uses lightweight models; thus, the inference can also run on the edge of each compute node to increase scalability. We obtain a very good accuracy (i.e., values ranging between 90–95 %), and we also demonstrate that
CHAPTER 4. CENTRALIZED ANALYTICS

the approach can be deployed on DC/SC nodes without negatively affecting the computing units performance.

4.1 Introduction

Nowadays, SCs and large DCs are increasing in scale and number of components, with systems composed of thousands of computing units [18, 80] and represent an increasingly complex industrial plan. Therefore, there is a vast number of sources of possible faults, heterogeneous in their nature, ranging from HW malfunctions or misconfigurations to SW unwanted behaviors or bugs. For system administrators who strive to guarantee systems operating in optimal conditions, identifying faulty situations and anomalous behaviors is a daunting task.

An automated online AD system capable of satisfying real-time requirements would be a boon for facility managers. Currently, most of the SoA AD systems are based on the analysis of system logs or log messages generated by dedicated SW tools, often at OS level [81–83]. In this way, there is not a general and uniform detection system, and deploying a set of different tools with different requirements still requires a lot of effort by the system administrators. This fact curbs the number and types of identifiable fault conditions with log-based tools.

However, today’s SCs and DCs have HW components with sensors to monitor physical and architectural parameters [2, 14, 17, 47, 84]. The monitoring infrastructure periodically reads a set of metrics and collects them into a single gathering point, where data is usually stored into a database for post-analysis. Of course, as shown in Chapter 2, the monitoring granularity of these centralized infrastructures is a bottleneck when scaling to large-scale systems. On the other hand, they allow keeping track of the activity of the whole DC/SC over long time periods (e.g., months/years of measurement acquisitions). Therefore, it seems a sensible idea to use the collected data to look for possible unhealthy situations. For example, a possible approach relies on supervised ML techniques, where a classifier is trained to distinguish between healthy and abnormal behaviors. In recent years, some approaches went in this direction, showing promising results but with a somewhat limited scope. For instance, supervised methods need
a carefully prepared initial phase where the SC is injected with all the kinds of faults to be detected later. This is a crucial drawback because it does not encompass the occurrence of new, unseen anomalies.

**Contribution:** In this chapter, we propose a novel automated method for AD in HPC systems and DCs, based on semi-supervised learning with AEs. The method has a very good accuracy (around 90–95% of detection accuracy) and was tested in a real SC in production, namely D.A.V.I.D.E.. Moreover, the approach employs coarse-grain measurements (i.e., 5 min intervals) and very lightweight ML models, which means it can run on common DC/SC centralized monitoring infrastructures, exploiting historical measurements - stored in a database - for training over long time periods.

**Outline:** Section 4.2 highlights the related work. Section 4.3 reports on the proposed methodology for automated online AD in DC/SC, along with the description of the ML method, and its training and inference phase. The experimental results are described in Section 4.4. We conclude the chapter in Section 4.5.

## 4.2 Related Work

Tuncer *et al.* [85] tackles the issue of diagnosing performance variations in HPC systems. They collect several measurements through a monitoring infrastructure; a group of statistical features modeling the state of the SC is obtained from these features. The authors then train different ML algorithms to classify the behavior of the SC using the statistical features. The results are promising, outperforming previous SoA [86,87]. Baseman *et al.* [88] propose a similar technique for AD in HPC systems. They apply a general statistical technique called Classifier-Adjusted Density Estimation (CADE) to help the training of a supervised Random Forest classifier. The classifier decides the class - *i.e.*, normal, anomaly, etc. - of each data point (set of physical measurements).

These methods both belong to the supervised area: the training set must contain examples of all classes to be detected - *i.e.*, samples of normal and abnormal states - and must be unbiased (equal number of cases for each class). This fact has a consequence: a first phase is required to create a labeled dataset, and the SC must be injected with
faults. Furthermore, supervised classifiers only learn to identify the classes already seen at training time; unseen anomalies encountered at run-time cannot be properly detected by these methods. Our approach, thanks to the semi-supervised learning, resolves both these issues.

Dani et al. [89] describe, instead, an unsupervised technique for AD in HPC. Their work is very different from our approach since they consider only the console logs generated by compute nodes (no monitoring infrastructure). Their purpose is to distinguish records relative to faulty conditions from logs created by healthy nodes; the proposed approach uses the K-means clustering algorithm. Their work targets faults that can be recognized by a node itself and stored in log messages; this bounds the number and the types of detectable anomalies. Conversely, in our approach, we detect anomalies using the data gathered via a collection framework without the need for AD systems already deployed on compute nodes.

4.3 Automated Online AD: Methodology, Training and Inference

In this section, we propose a system-oriented methodology to automatically detect anomalies based on an ML model and relying on the data collected by DC/SC monitoring infrastructures. In particular, we test the feasibility of the approach on a real tier-1 SC in production, namely D.A.V.I.D.E. and its SoA monitoring infrastructure (i.e., DiG + ExaMon). Moreover, we describe then the ML method used to detect anomalies, along with its training and inference phase.

4.3.1 Methodology for Automated AD

The proposed scheme is depicted in Figure 4.1. The DC/SC compute nodes are endowed with embedded boards (i.e., BBB) that measure a set of metrics describing the system behavior. These measurements are sent via MQTT to a centralized monitoring unit (i.e., ExaMon, which runs on a Front-End node of D.A.V.I.D.E.) and stored in a distributed database for time series (i.e., KairosDB [90]), built on top of the NoSQL database Apache Cassandra [48]. This allows us to retrieve historical data easily and use them for training purposes. Moreover, we
4.3. **AUTOMATED ONLINE AD**

**Figure 4.1:** Centralized-AI methodology architecture.

Use further plugins that run directly in the centralized monitoring unit to collect at coarse-grain cluster-level info (e.g., cooling, environment temperature, job scheduler). Thanks to all these measurements, it is possible to distinguish between anomalous and normal conditions.

Data collected with the monitoring framework is fed to an ML model to train it to detect anomalies. During the training phase, the model is going to encounter only examples describing a system under normal conditions. After the training phase, the ML models can run inference on the centralized monitoring unit, or - if we want to scale to large-scale systems without overwhelming the computing resources of the centralized monitoring node - can be loaded on the embedded monitoring boards; when new measurements arrive, the trained ML model takes them as input and can identify anomalies, triggering an alarm for system admins.

### 4.3.2 Semi-supervised Learning Based on AEs

As an ML model to detect anomalies, we propose an approach based on a particular type of NN, namely an AE [91]. We exploit the series of measurements (features) describing the state of the DC / HPC system, and collected with our monitoring infrastructure DiG. Under healthy operating conditions, these features are connected by specific relationships (i.e., the power consumption directly depends on the workload, the temperature is related to the clock frequency, etc.). These correlations are no longer valid when the system enters an anomalous state. The main idea of our method relies on the AE capability
to learn the typical (normal) correlation between the measures and then consequently identify changes in this correlation that indicate an abnormal situation. This research avenue has been partially explored in recent years, although not in the HPC field [92–96]. We teach the AE the normal behavior of the compute nodes; after the training phase, the AE will be able to detect anomalous situations.

To understand how an AE can detect abnormal conditions, we have to look at its structure. An AE is a NN trained to copy its input $I$ to its output $O$. It has one or more internal layers $H$ that try to represent the data taken as input. An AE is split in two subparts: an encoding function $H = enc(I)$ and a decoding function that reconstructs the input $O = dec(H)$. Typically, AEs do not just learn the identity function $dec(enc(I)) = I$ but are designed not to create perfect copies, e.g., the dimension of the hidden layer can be smaller than the dimension of the input. Thus, the output of an AE is generally different from its input, and the difference between input and output is called reconstruction error.

We take advantage of the reconstruction error to detect anomalies. We train the AE with data corresponding to the normal state and minimize the reconstruction error; this error is called training error. In this way, it learns the normal correlations between the features from the monitoring infrastructure. After this first training phase, we feed the AE with new data unseen before – this is generally called inference in ML terminology – and we then observe the reconstruction error. If the new data is similar to the data used as input (if it respects the normal correlations), then the error will be small and comparable to the training error. If the new data correspond to an anomalous situation, the AE will struggle to perform the reconstruction, since the learned correlations do not hold. Hence we identify anomalies by observing large reconstruction errors (w.r.t. to the training error) during the inference phase.

### 4.3.3 Autoencoders Training

We create an AE for each node in the SC since, as we will describe in Section 4.4, dedicated models for each node outperform a single, generic model to be applied to all nodes. Each network is trained using node related data collected by DiG. In the training phase, we use only
4.3. AUTOMATED ONLINE AD

data corresponding to the normal state. The dataset is a collection of a couple of hundreds of metrics, ranging from core load, frequency and temperature, to node power consumption, room temperature, GPUs usage, cooling fans speed, etc. The metrics (also referred to as features) form the input set for the NNs.

For each node, we have a training set corresponding to 2 months of normal behavior (obtained in collaboration with system admins). Due to storage reason, the monitoring data stored in the database is not preserved for more than a week (notice that we send to the centralized monitoring unit of D.A.V.I.D.E. power measurements at 1 ms), hence for the training we use coarse-grain data, where the measurements are aggregated in five minute intervals. After collecting the raw data, we pre-process it (e.g., we remove data corresponding to periods where the monitoring system was not working properly) and normalize it. This preparation takes about 30 s. The final number of features is 166.

We adopted the same network topology for each node. After an empirical evaluation, we chose a sparse AE model [97], that proved to be the better option in terms of accuracy and computational demands, especially for training/inference times. The network is composed of three fully connected layers, an input layer, an output layer, and one hidden layer. As activation function for the neurons we use Rectified Linear Units [98] (ReLU); as regularization term (needed in sparse networks) we employ the L1-norm loss function [91, 99]. The input and the output layers have as many neurons as the number of the features (166), while the hidden-layer has ten times the number of features (thus 1660 in total).

Each AE is trained with data from its corresponding node; for the training, we use the Tesla P100 GPUs available in the SC nodes (each node trains its own AE). For the training phase, we use Adam algorithm [100], with mean absolute error as target loss. After some preliminary experiments, we opted for a batch size equal to 32 and 100 epochs. The training takes around 20 s for each AE. The training times and overhead are not a critical concern since the training phase takes place only once or at a very low rate (once every few months), and can be scheduled during maintenance periods. The framework used for the design and the training of the AEs is Keras [101] with TensorFlow for GPU [102] as a back-end.
4.3.4 Online Inference

After training the AEs, we can execute the inference online directly on the centralized monitoring unit, feeding the ML models with new data taken from the database. Moreover, as the models are lightweight, if we want to scale to very large DCs / SCs (e.g., thousand / millions of compute nodes), without impacting on the computing resources of the node dedicated for the centralized monitoring, it is possible to run the AE models also in the embedded monitoring boards of DiG. For this purpose, we installed TensorFlow on the BBB and took advantage of the NEON accelerator (SIMD architecture). On each BBB, we load the trained AE of the corresponding node; then, we feed it with new data coming directly from the sensors monitored by DiG. The computing time to process the set of 166 features in the BBB is just 11 ms, which is a negligible overhead considering the sampling rate of several minutes.

4.4 Experimental Results

We injected two different anomalies in the SC used as a test case: we changed the frequency governor configuration of the compute nodes (see [103] for details). Changing the frequency governor disrupts the relationships between core load and core frequency (and other related features, such as power consumption, etc.). This is a misconfiguration anomaly and should be detectable with our approach. The normal configuration is the default; we changed it to powersave and performance, in different periods of time\(^1\). To evaluate the trained AEs, we consider the reconstruction error. As described in Section 4.3.2, we expect to observe greater reconstruction errors during anomalous periods w.r.t. normal ones.

Our hypothesis is confirmed by quantitative and visual analysis of the reconstruction error. Figure 4.2 plots the reconstruction error (y-axis) obtained in a roughly two months period (x-axis), for a specific node. The reconstruction error trend is plotted with a light blue line; the gaps in the line represent periods when the node was idle and that

\(^1\)default: the frequency of a core directly depends on its load; powersave/performance: frequencies are forced to the lowest/highest possible value
4.4. EXPERIMENTAL RESULTS

Figure 4.2: Reconstruction error for a compute node of D.A.V.I.D.E.

have been removed from the dataset (thus, we exclude them from the reconstruction error). We have six anomalous periods, identified by colored highlights along the $x$-axis: during the first five (red lines), the frequency governor was set to powersave, while during the last one (blue), the governor was set to performance. It is possible to see that, on average, the reconstruction error is larger during anomalous periods compared to the normal ones. This observation is confirmed by a quantitative analysis performed on all nodes: the average normalized reconstruction error computed during normal periods (excluding the training set) is equal to 1.08, while the average error obtained during anomalies (grouping both types) is equal to 14.54. If we use a generic AE for all nodes (instead of a series of dedicated ones), the error for the anomalous periods decreases to 6.29 while the error for normal periods remains similar (1.01). To detect anomalies, we prefer reconstruction errors as large as possible with anomalous data, hence the set of dedicated models is better than the generic one.

Accuracy Evaluation. We use a threshold-based method to distinguish anomalies from normal states. Suppose we have a data point $i$ that contains features collected by the monitoring infrastructure. To classify it, we feed it to the trained AE: if the reconstruction error $E_i$ is greater than a threshold $\theta$, then the point is “abnormal”; otherwise, the data point is considered normal. As a threshold, we choose the $n$-th percentile of the error distribution of the normal dataset, where
$n$ is a value that depends on the specific AE/node. To find the best $n$ for each AE, we employed a simple generate-and-test search strategy; i.e., we performed experiments with a finite number of values (after a preliminary empirical evaluation) and then chose those guaranteeing the best results in term of classification accuracy. The fact that the best results are obtained by using a different $n$ for each node validates the choice of having multiple dedicated AEs rather than a generic one.

The classification accuracy is measured by the F-score [104] for each class, normal (N) and anomaly (A). F-score ranges between 0 and 1, with higher values indicating higher accuracy. In Table 4.1 we see the results. In the first column, there is the node name (we show the values for a subgroup of nodes). The remaining columns report the F-score values for three different $n$-th percentiles; there are two F-score values for each $n$-th percentile, one for the normal class (N) and one for the anomaly class (A). We can see that the F-score values are very good, highlighting the good accuracy of our approach, with an accuracy between 0.87 and 0.98.

### 4.5 Conclusion

In this chapter, we have presented an approach for automated AD in HPC environments and DCs. Our method leverages ML and coarse-grain measurements; thus, it can be deployed on common centralized monitoring infrastructures of DCs/SCs. Moreover, with a DiG-like monitoring infrastructure, it can also run real-time inference.
at the edge of the DC, allowing to scale also to very large DCs and HPC infrastructures. We use AEs trained to learn the normal behavior of each compute node based on its historical telemetry data of “good” behaviors. All the reported results are obtained in a real in-production SC (i.e., D.A.V.I.D.E.), during two months of activity of the D.A.V.I.D.E.’s scientific community research.
Chapter 5

ปาเอลลา: Edge-AI based Malware Detection in Data Centers

The increasing use of 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 AI powered edge computing is envisaged to be a promising direction. In this chapter, we focus on DCs and SCs, where a new generation of high-resolution monitoring systems is being deployed (e.g., HAEC [12], DiG), opening new opportunities for analysis like AD 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 MD, it runs on an out-of-band IoT-based monitoring system for DCs/SCs, and involves PSD of power measurements, along with AEs. Results are promising, with an F1-score close to 1, and a FA and MM rate close to 0%. We compare our method with SoA 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.

5.1 Introduction

In the era of the 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 [32,105–107]. 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 [108–110]. 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 methods based on AI running on edge devices (a.k.a edge AI or edge intelligence) are envisioned as a promising solution to address this challenge [111,112].

This is true also in the context of SCs and DCs, where a new generation of high-resolution monitoring systems is being deployed [1,12,25], pushing the research boundaries over new opportunities to support their automation, analytics, and control. In particular, DiG [1] 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. [3,14,15]). 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 (\emph{i.e.}, below microseconds) and a vertical integration in a scalable distributed data analytics framework (\emph{i.e.}, ExaMon [16]).

![Diagram](image)

**Figure 5.1**: 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 [9], while with current out-of-band monitoring solutions it is not possible to carry out on-board edge analytics [1]. As introduced in Section 2.1 - and also sketched in Figure 5.1.1 -, using an out-of-band monitoring with high-resolution measurements (\emph{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 SW tools that have to handle the measurements (in real-time and offline) [1]. Instead, with a DiG-like monitoring infrastructure (Figure 5.1.2) 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. As we will show in
this chapter, DiG provides a flexible platform to collect fine-grain measurements of the compute nodes activity, and prototype novel AD methods. In particular, we will show how a new approach involving AI and ML running on IoT monitoring devices, can allow increasing the cybersecurity of DCs, which is nowadays a high impact research topic [113–115]. Our edge AI approach targets real-time MD for DCs/SCs, and involves feature extraction analysis based on PSD estimation with the Welch method, along with an AE to identify patterns of malware. While similar approaches are already used in the context of speech recognition [116,117], 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 \textit{pAElla}, which stands for “Power-AutoEncoder-WeLch for anomaLy and Attacks”.

\textit{Contribution:}

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 (\textit{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 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 SW we used for the analysis [118]. The dataset includes 7 benchmarks (6 scientific applications
5.2. BACKGROUND ON MALWARE DETECTION

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 [3].

Outline: Section 5.2 introduces background information on performance counters based MD. Section 5.3 reports the related work. Section 5.4 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 5.5 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 chapter in Section 5.6.

5.2 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 [7]. 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 HW performance counters available in modern processors [5,7,119–121]. 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 [7].
There are mainly two ways to train the anomaly-based detector to catch malware: (i) training on malware signatures [119], 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 [7]. 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) [7,27]. 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 [122]). 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 [7]. However, as described in Section 5.1, 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 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.
5.3 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, [119] 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, [7] 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 5.1 - 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 [5,6,27,121] showed how to use performance counters to detect anomalies, while [55] showed how to use them for detecting covert cryptocurrency mining. However, very recent works in [123,124] 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, [125] presented a machine learning based approach to detect ransomware attacks by monitoring the power consumption of Android devices, or [126] 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 MD for general-purpose computers was proposed in [127]. In this paper, Bridges et al. use an unsupervised one-class AD ensemble, based on statistical features (*e.g.*, mean, variance, skewness and Kurtosis), with a power consumption sampling rate of 17ms. 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 [9], which can provide a time granularity up to 1 ms, while examples of out-of-band solutions are IBM Amester [10] and IPMI [39], which can provide a resolutions of 250 $\mu$s 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 [2,12], and DiG [1] is the best-in class SoA infrastructure that bridges the gap of very fine-grain monitoring (i.e., 20 $\mu$s 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 [27,128], the use of Fourier analysis together with NNs is a well-known procedure in speech recognition [116,117]. We use a similar approach for MD 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.

5.4 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.

5.4.1 Edge Real-Time Feature Extraction

For the Feature Extraction phase, we use the PSD with the Welch method [129]. 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 [129]. Moreover, PSDs already proved to be a robust feature extraction method to be used together with NNs, in applications such as speech recognition [116, 117].

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 1kHz, where we alternate high load computational phases with idle phases. Figure 5.2 shows the results of

![PSD of Pulse Train Instr. @1kHz, PSD_win=163.84ms (8192 samples) FFT_win=81.92ms (4096 samples)](image)

![PSD of Pulse Train Instr. @1kHz, PSD_win=163.84ms (8192 samples) FFT_win=40.96ms (2048 samples)](image)

![PSD of Pulse Train Instr. @1kHz, PSD_win=163.84ms (8192 samples) FFT_win=20.48ms (1024 samples)](image)

Figure 5.2: Comparison between PSDs of a Pulse Train of instructions at 1kHz for different FFT lengths.
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., \((FFT\texttt{\_samples}/2)+1\) - 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 5.5, 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.

### 5.4.2 Edge Malware Detection Inference

For the ML inference phase, we use an AE, which, as we explained in Chapter 4, it is a particular kind of NN suitable for AD [27,128]. 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. 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.

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 Section 5.5, 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 [91], while as activation functions for the
neurons we use the hyperbolic tangent (tanh) and the 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 the reconstruction error, we use the Mean Square Error (MSE) between input and output of the AE. Then we tag as anomalies all the PSDs where the reconstruction error is greater than \(T_E\). The second threshold, namely \(T_O\), 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.

### 5.4.3 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\(^1\), (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 5.3 highlights their implementation in the BBB.

**Phase 1:** The bottom layer represents the ADC HW 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 HW FIFO. By tuning the ADC sampling frequency (Fs\(_{ADC}\) w.r.t. Figure 5.3) and the HW averaging (AVG), it is possible to set the frequency (Fs) at which samples enter the SW layers. The best trade-off corresponds to a sampling rate (Fs\(_{ADC}\)) of 800 kS/s per channel - maximum rate when using two channels - and HW average every 16 samples. This is equivalent to 50 kS/s (i.e., Fs equals to 20 \(\mu\)s) and allows to (i) cover the entire signal bandwidth of the server power

\(^1\)Notice that we modified the embedded SW architecture presented in Chapter 2 for the needs of pAElla.
Figure 5.3: Implementation of pAElla in the IoT embedded systems of DiG.

consumption, and (ii) reach a measurement precision below 1\% (\(\sigma\)) of uncertainty (a.k.a. oversampling and averaging method [50]). 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 (12kB).

Phase 2: When a given number of samples is collected (we set it to 2048 samples, which correspond to 4kB in memory - 2B 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 (163.84ms) 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.

Phase 3: 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.

5.4.4 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 this 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 [130], 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 HW (e.g., large variations of material properties when moving to 10–7 nm chips [131]).
CHAPTER 5. EDGE ANALYTICS

5.5 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.

5.5.1 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 QE [54], HPLinpack (HPL) [132], High Performance Conjugate Gradients (HPCG) [133], Gromacs [134] and NAS Parallel Benchmarks (NPB) [135], 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) HW performance counters and (ii) PSDs computed from the high-resolution power measurements. In particular, we analyzed the performance counters suggested by Tang et al. [7], plus further metrics available with our open source monitoring tool, ExaMon [16]. We summarize a list of the selected metrics in Table 5.1 (per-core metrics) and Table 5.2 (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 5.2, 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 \textit{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 \texttt{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 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 (\textit{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 SW to analyze the data open source, at the following link [118].

### 5.5.2 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
CHAPTER 5. EDGE ANALYTICS

ML algorithms suitable for Anomaly and Malware Detection, namely oc-SVM [7], Isolation Forest (IF) [137] and AEs. For oc-SVM and IF, we use the Scikit-learn [138] implementation developed in python, while for the AE we use Keras [101] with TensorFlow [139] as back-end.

After an empirical evaluation, we adopted standardization (Scikit-learn StandardScaler) like in pAElla (settings described in Section 5.4.2), 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 [137]). 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 5.3 reports the results of our analysis in the test set. Namely, the (i) FA rate (= False Positive - FP - rate) - i.e., healthy benchmarks erroneously labeled as malware; (ii) the MM rate (= False Negative - FN - rate) - i.e., malware not detected; and (iii) the weighted F1-score [5] (best value at 1, and worst at 0), which measure the test accuracy based on the following formula:

\[
F1score = \frac{2TP \cdot W_M}{2TP \cdot W_M + FP \cdot W_H + FN \cdot W_M}
\]  

(5.1)

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 AD in SCs [5, 8], 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 [123, 124], when using tree-based models with performance counters for MD (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 5.3. 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 of 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.

5.5.3 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 [52] 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 × 50 kS/s per node × 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 kilobytes for training and a few bytes to send alarms when potential malware are detected.

Moreover, we highlight 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 5.1 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 this data to a centralized monitoring unit, with the consequent overhead in the network infrastructure (i.e., roughly 13.3 TS/s).
5.6 Conclusion

This chapter reports on novel method - namely pAElla- to increasing the cybersecurity of DCs/SCs, involving AI-powered edge computing. The method targets real-time MD running on an out-of-band IoT-based monitoring system for DCs, and involves feature extraction analysis based on PSD of power measurements, along with AE NNs. 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 [118]. 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 AD, opening new perspectives for future works.
### Table 5.1

**SELECTED PER-CORE PERF COUNTERS BASED ON SOA [7]**

<table>
<thead>
<tr>
<th>Raw Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>UOPS_RETIRE_D (Retired Micro-ops)</td>
</tr>
<tr>
<td>ICACHE (Instruction Cache misses)</td>
</tr>
<tr>
<td>LONGEST_LAT_CACHE (Core cacheable demand req. missed L3)</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_L3_HIT_RETIRE_D (Retired load uops in L3)</td>
</tr>
<tr>
<td>BR_MISP_RETIRE_D (Mispredicted branch instr. retired)</td>
</tr>
<tr>
<td>UOPS_ISSUE_D (Uops issued to Reservation Station - RS)</td>
</tr>
<tr>
<td>IDQ_UOPS_NOT_DELIVERED_D (Uops not deliv. to RAT per thread)</td>
</tr>
<tr>
<td>INT_MISC (Core Cycles the allocator was stalled)</td>
</tr>
<tr>
<td>BR_INST_RETIRE_D_NEAR_RETURN_D (Instr. retired direct near calls)</td>
</tr>
<tr>
<td>BR_INST_RETIRE_D_NEAR_CALL_D (Direct and indirect near call ret.)</td>
</tr>
<tr>
<td>BR_INST_EXEC_ALL_DIRECT_NEAR_CALL_D (Retired direct calls)</td>
</tr>
<tr>
<td>BR_INST_EXEC_TAKEN_INDIRECT_NEAR_CALL_D (Ret. indir. calls)</td>
</tr>
<tr>
<td>DTLB_STORE_MISSES_STLB_HIT_D (Store operations miss first TLB)</td>
</tr>
<tr>
<td>ARITH_DIVIDER_UOPS_D (Any uop executed by the Divider)</td>
</tr>
<tr>
<td>DTLB_LOAD_MISSES_STLB_HIT_D (Load operations miss first DTLB)</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRE_D_L3_MISS_D (Miss in last-level L3)</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRE_D_L2_MISS_D (Miss in mid-level L2)</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRE_D_L1_MISS_D (Ret. load uops miss in L1)</td>
</tr>
<tr>
<td>BR_MISP_EXEC_ALL_BRANCHES_D (Retired mispr conditional branch)</td>
</tr>
<tr>
<td>BR_MISP_EXEC_TAKEN_RETURN_NEAR_D (Retired mispr. indir. br.)</td>
</tr>
<tr>
<td>C3 (Clock cycles in C3 state)</td>
</tr>
<tr>
<td>C3res (C3 residency - Clock cycl. in C3 state between 2 sampling time)</td>
</tr>
<tr>
<td>C6 (Clock cycles in C6 state)</td>
</tr>
<tr>
<td>C6res (C6 residency - Clock cycl. in C6 state between 2 sampling time)</td>
</tr>
<tr>
<td>temp (Cores temperature)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Derived Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_core (per-core load)</td>
</tr>
<tr>
<td>IPS (Instructions per Second)</td>
</tr>
<tr>
<td>CPI (Cycles per Instruction)</td>
</tr>
</tbody>
</table>

**Note.** More info about the metrics are available in [7, 16, 136].
## Table 5.2

### Selected Per-CPU Perf Counters

<table>
<thead>
<tr>
<th>Raw Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2 (Clock cycles in C2 state)</td>
</tr>
<tr>
<td>C3 (Clock cycles in C3 state)</td>
</tr>
<tr>
<td>C6 (Clock cycles in C6 state)</td>
</tr>
<tr>
<td>C2res (C2 residency)</td>
</tr>
<tr>
<td>C3res (C3 residency)</td>
</tr>
<tr>
<td>C6res (C6 residency)</td>
</tr>
<tr>
<td>freq_ref (Core frequency)</td>
</tr>
<tr>
<td>erg_units (Energy units)</td>
</tr>
<tr>
<td>temp_pkg (Package temperature)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Derived Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>erg_dram (Energy DRAM consumed)</td>
</tr>
<tr>
<td>erg_pkg (Package Energy consumed)</td>
</tr>
</tbody>
</table>

*Note. More info in [16, 136].*
### Table 5.3

**Final results of the MD analysis and comparison with SoA approaches**

<table>
<thead>
<tr>
<th>ML Approach</th>
<th>Benchmark</th>
<th>Perf Metrics + Power Statistics @20ms</th>
<th>PSD of Power @20us</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False Alarm</td>
<td>Malware Miss</td>
<td>F1-score</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.042</td>
<td>0.645</td>
</tr>
<tr>
<td>oc-SVM</td>
<td>Idle</td>
<td>1.000</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>QE</td>
<td>1.000</td>
<td>0.095</td>
</tr>
<tr>
<td></td>
<td>HPL</td>
<td>1.000</td>
<td>0.600</td>
</tr>
<tr>
<td></td>
<td>HPCG</td>
<td>1.000</td>
<td>0.074</td>
</tr>
<tr>
<td></td>
<td>Gromacs</td>
<td>1.000</td>
<td>0.853</td>
</tr>
<tr>
<td></td>
<td>NPB_btC9</td>
<td>1.000</td>
<td>0.884</td>
</tr>
<tr>
<td></td>
<td>NPB_btC16</td>
<td>1.000</td>
<td>0.365</td>
</tr>
<tr>
<td>Overall</td>
<td>1.000</td>
<td>0.000</td>
<td>0.053</td>
</tr>
<tr>
<td>IF</td>
<td>Idle</td>
<td>0.000</td>
<td>0.137</td>
</tr>
<tr>
<td></td>
<td>QE</td>
<td>0.000</td>
<td>0.642</td>
</tr>
<tr>
<td></td>
<td>HPL</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>HPCG</td>
<td>0.000</td>
<td>0.874</td>
</tr>
<tr>
<td></td>
<td>Gromacs</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>NPB_btC9</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>NPB_btC16</td>
<td>0.667</td>
<td>0.238</td>
</tr>
<tr>
<td>Overall</td>
<td>0.238</td>
<td>0.244</td>
<td>0.758</td>
</tr>
<tr>
<td>AE</td>
<td>Idle</td>
<td>0.000</td>
<td>0.579</td>
</tr>
<tr>
<td></td>
<td>QE</td>
<td>0.000</td>
<td>0.937</td>
</tr>
<tr>
<td></td>
<td>HPL</td>
<td>0.000</td>
<td>0.947</td>
</tr>
<tr>
<td></td>
<td>HPCG</td>
<td>0.167</td>
<td>0.042</td>
</tr>
<tr>
<td></td>
<td>Gromacs</td>
<td>0.000</td>
<td>0.947</td>
</tr>
<tr>
<td></td>
<td>NPB_btC9</td>
<td>0.000</td>
<td>0.084</td>
</tr>
<tr>
<td></td>
<td>NPB_btC16</td>
<td>0.000</td>
<td>0.189</td>
</tr>
<tr>
<td>Overall</td>
<td>0.024</td>
<td>0.532</td>
<td>0.627</td>
</tr>
</tbody>
</table>

**Note.** For best results we want FA rate (healthy benchmarks seen as malware) and MM rate (malware not detected) close to 0, while F1-score close to 1. Results related to pAElla are highlighted in bold.
Chapter 6

Summary and Conclusion

Today, DCs and HPC systems have become extremely large and complex, and the necessity for novel methods to support their automation, analytics, and control is gaining significant attention. Based on this support, industrial and academic researchers are pushing for continuous enhancements in their monitoring infrastructures to provide more and more detailed information on the activity of the DC. However, with the increase in measurement resolution and the ensuing vast amount of data to handle, new challenges arise, such as bottlenecks on the network bandwidth, storage, and monitoring software tools.

In this thesis, we have investigated several research questions toward the design of a new generation of DCs and HPC monitoring systems. In particular, we designed the first monitoring system - i.e., DiG- that enables real-time profiling and analytics, both at the edge and on a centralized unit, on SoA high-resolution power measurements and performance counters. We have then evaluated the performance of the two most used time synchronization protocols - i.e., NTP and PTP - on the D.A.V.I.D.E. HPC machine, proving that both are suitable for most requirements of HPC parallel applications (i.e., sub-millisecond synchronization), and for timestamping measurements of current SoA
DC monitoring infrastructures. Finally, we exploited the flexibility of DiG to analyze data both directly on the network edge and on a centralized unit, and to develop and implement two methods for anomaly/malware detection in DCs.

6.1 Overview of the Main Results

The main results and contributions can be summarized as follows.

High-Resolution DC Monitoring System

We have proposed and developed a high-resolution monitoring infrastructure for DCs and HPC systems \((i.e., \text{DiG})\). The design allows out-of-band monitoring, and real-time edge/centralized analytics. We exploit a custom power sensor at the plug to trace the power consumption at high resolution, covering the entire signal bandwidth (sampling up to \(20\,\mu s\)) with a measurement precision below 1% \((\sigma)\). The system allows to interface with existing out-of-band telemetry \((e.g., \text{Amester [10], IPMI [39]})\), but also with in-band built-in tools \((e.g., \text{RAPL [37]})\). All the measurements are synchronized at sub-microseconds precision to obtain a detailed picture over time of the nodes and cluster. We adopted a scalable and lightweight interface to the centralized monitoring \((i.e., \text{MQTT [40]})\) to support large-scale computing centers. The monitoring infrastructure is technology agnostic \((i.e., \text{already tested on Intel, ARM, and IBM})\) and low cost \((i.e., \text{the custom power sensor does not require any motherboard redesign})\). Moreover, the system is currently installed in a tier-1 SC already in production \((i.e., \text{D.A.V.I.D.E. [3]})\).

Evaluation of Time Synchronization Protocols on DCs

Tight time synchronization is an important requirement for DCs/SCs and their monitoring infrastructures. In this work, we have evaluated the performance of the two most used time synchronization protocols \(- i.e., \text{NTP and PTP - on a tier-1 green HPC system (i.e., D.A.V.I.D.E.)}.\) Our results show NTP can achieve in SC nodes an accuracy below \(2.6\,\mu s\) and a precision below \(2.7\,\mu s\). Moreover, we observed that 99% of the measurements stay within \(8.9\,\mu s\) of skew from the time reference.
Regarding PTP, we measured in the SC nodes an accuracy within 0.2 \( \mu s \), a precision within 0.8 \( \mu s \), and observed that 99\% of the measurements skew from the time reference of no more than 2.2 \( \mu s \). It is noteworthy that it is possible to achieve this performance with low-cost COTS HW, which is generally built-in in SoA DCs / SCs (no extra cost is required). Moreover, we observed that the synchronization performance (i.e., accuracy and precision) achievable in the IoT devices used in DiG (i.e., best-in-class high-resolution DC monitoring system) is below their monitoring sampling period (20 \( \mu s \)); thus it is suitable to correlate the monitored metrics with applications running in the compute nodes. Finally, we show that both scalability and CPU overhead are not a concern when using the proposed settings to obtain high synchronization performance with the two protocols.

**Online Anomaly Detection in DCs**

Reliability is a cumbersome problem in DCs and HPC Systems evolution. During operation, several types of fault conditions and anomalies can arise, ranging from malfunctioning HW to improper configurations or imperfect SW. Currently, system administrator and final users have to discover it manually. Certainly, this approach does not scale to large scale SCs and facilities: automated methods to detect faults and unhealthy conditions are needed. In this thesis, we exploit DiG to study and develop a novel automated approach for AD in DCs / SCs, based on semi-supervised learning with AEs. The method has a very good accuracy (around 90–95\% of detection accuracy) and was tested in a real SC in production, namely D.A.V.I.D.E.. Moreover, it employs coarse-grain measurements (i.e., 5 min intervals) and very lightweight ML models, which means it can run inference on both (i) common DC / SC centralized monitoring infrastructures or (ii) on edge (namely in the DiG embedded devices). Finally, we propose a method for online training, which is suitable to run in DCs / SCs in production.

**Malware Detection in DCs**

As in any domain involving computers, security is a crucial problem in DCs / SCs. Despite the existence of anti-virus software, the different ways malware can subvert them are constantly growing. To mitigate
this problem, we proposed a novel lightweight and scalable approach - we called it pAElla- that exploits AI powered edge computing and high-resolution power measurements for MD. The method involves PSD of power measurements and semi-supervised learning with AEs (i.e., training on healthy data). Results are promising, with an F1-score close to 1, and a FA and MM rate close to 0%. We compare our method with SoA 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 [118].

6.2 Outlook

In the following, we outline promising directions to advance the capabilities of high-resolution monitoring infrastructures in DCs / SCs, and further improve anomaly and malware detection methods running on these systems.

High-Resolution DC Monitoring & Synchronization

In this thesis, we proposed a new approach to improve current DC / SC monitoring solutions. However, some aspects still leave room for further improvements. While our solution involves a custom power sensor for monitoring the power consumption at the plug with high-resolution, it would be interesting to exploit open standards, such as the OpenPOWER architecture and the OpenBMC firmware to improve the way they are currently collecting data (e.g., the OpenBMC is currently pushing out the data based on IPMI protocol, which is well known for its latency issues), and thus the final per-component sampling granularity at which the measurements can be analyzed. Moreover, using embedded HW accelerators for more advanced ML on edge (e.g., the open HW project PULP [140]), and give access to this open HW platform to the DC / SC research community would be a further benefit. Finally, about synchronization, the accuracy and precision reached by PTP are by far meeting the current requirements of microsecond synchronization. However, if a finer time synchronization will be
required by future monitoring infrastructures, White Rabbit [69] - which extends PTP with dedicated open HW, firmware and SW, and can synchronize nodes with sub-nanosecond accuracy and picosecond precision - could be an alternative solution.

**Anomaly/Malware Detection in DCs**

In Chapter 4 and Chapter 5, we proposed two novel methods for automated online anomaly and malware detection in DCs, respectively. In particular, our methods exploit AE NNs with semi-supervised learning on healthy data. It would be interesting to test also other kinds of ML algorithms, which take into account also the time information of the time series, such as Long Short-Term Memory (LSTM) AEs. Also, more advanced automated ways to replace DC’s system admin’s intervention have to be investigated (e.g., in our case, when possible directly fix the anomaly issue, instead of just sending an alarm).

**Integration with DC/SC compute rooms’ monitoring**

A valuable improvement to the proposed high-resolution monitoring infrastructure would come from the integration with compute rooms’ monitoring. As an example, collecting metrics like environment temperature, humidity, air pressure, and vibrations, with WSNs spread in the DC/SC compute rooms, and use the same scalable interface used in DiG (i.e., MQTT) to the centralized monitoring unit would be a possible solution. In this direction, already several works proved it is possible to improve the overall performance and energy efficiency of the DC/HPC system [141,142]. Also, another interesting idea that needs to be further investigated is the use of thermal infrared array sensors to obtain insights into temperature distributions within the compute room and between the compute nodes.

**Closing the Loop Improving Perf. & Energy Efficiency**

Proceeding toward this path, we envision that such a new generation of monitoring infrastructures will open new opportunities also to improving the performance and energy efficiency of DCs and HPC systems. As proved in several works [30,143,144], methods for power
management and autotuning of applications, based on data collected by monitoring infrastructures, can drastically help to improve performance and energy efficiency. With this aim, we encourage further investigation in this direction.
## Appendix A

### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Anomaly Detection</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog-to-Digital Converter</td>
</tr>
<tr>
<td>AE</td>
<td>Autoencoder</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>BBB</td>
<td>Beaglebone Black</td>
</tr>
<tr>
<td>BC</td>
<td>Boundary Clock</td>
</tr>
<tr>
<td>BMC</td>
<td>Baseboard Management Controller</td>
</tr>
<tr>
<td>BSP</td>
<td>Bulk Synchronous Parallelis</td>
</tr>
<tr>
<td>CADE</td>
<td>Classifier-Adjusted Density Estimation</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Function</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-The-Shelf</td>
</tr>
<tr>
<td>CPI</td>
<td>Cycles per Instruction</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CV</td>
<td>Coefficient of Variation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>DC</td>
<td>Data Center</td>
</tr>
<tr>
<td>DC-DC</td>
<td>Direct current to Direct Current</td>
</tr>
<tr>
<td>DiG</td>
<td>Dwarf in a Giant</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>DVFS</td>
<td>Dynamic Voltage and Frequency Scaling</td>
</tr>
<tr>
<td>DWT</td>
<td>Discrete Wavelet Transform</td>
</tr>
<tr>
<td>EtS</td>
<td>Energy-to-Solution</td>
</tr>
<tr>
<td>FA</td>
<td>False Alarm</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>Flops</td>
<td>Floating point operations per second</td>
</tr>
<tr>
<td>FN</td>
<td>False Negative</td>
</tr>
<tr>
<td>FP</td>
<td>False Positive</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input / Output</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard disk drive</td>
</tr>
<tr>
<td>HE</td>
<td>Hall Effect</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>HPCG</td>
<td>High Performance Conjugate Gradients</td>
</tr>
<tr>
<td>HPL</td>
<td>HPLinpack</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection Systems</td>
</tr>
<tr>
<td>IF</td>
<td>Isolation Forest</td>
</tr>
<tr>
<td>IIO</td>
<td>Industrial I/O</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet-of-Things</td>
</tr>
<tr>
<td>IPS</td>
<td>Instructions per Second</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>MD</td>
<td>Malware Detection</td>
</tr>
<tr>
<td>MII</td>
<td>Media Independent Interface</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MM</td>
<td>Malware Miss</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>MQTT</td>
<td>MQ Telemetry Transport</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>NPB</td>
<td>NAS Parallel Benchmark</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>OC</td>
<td>Ordinary Clock</td>
</tr>
<tr>
<td>OCC</td>
<td>On Chip Controller</td>
</tr>
<tr>
<td>OCP</td>
<td>Open Compute Project</td>
</tr>
<tr>
<td>oc-SVM</td>
<td>one-class Support Vector Machine</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>pAElla</td>
<td>Power-AutoEncoder-WeLch for anomaly and Attacks</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PE</td>
<td>Processing elements</td>
</tr>
<tr>
<td>PHC</td>
<td>PTP Hardware Clock</td>
</tr>
<tr>
<td>PMU</td>
<td>Performance Monitoring Unit</td>
</tr>
<tr>
<td>PRU</td>
<td>Programmable real-time unit</td>
</tr>
<tr>
<td>PSD</td>
<td>Power Spectral Density</td>
</tr>
<tr>
<td>PSU</td>
<td>Power Supply Unit</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
</tr>
<tr>
<td>QE</td>
<td>Quantum Espresso</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>SAR</td>
<td>Successive Approximation Register</td>
</tr>
<tr>
<td>SC</td>
<td>Supercomputer</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Synchronous Dynamic Random Access Memory</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>SoA</td>
<td>State-of-the-Art</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TC</td>
<td>Transparent Clock</td>
</tr>
<tr>
<td>TCR</td>
<td>Temperature Coefficient of Resistance</td>
</tr>
<tr>
<td>TP</td>
<td>True Positives</td>
</tr>
<tr>
<td>TSC</td>
<td>Time Stamp Counter</td>
</tr>
<tr>
<td>TtS</td>
<td>Time-to-Solution</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Networks</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
<tr>
<td>WPT</td>
<td>Wavelet Packet Transform</td>
</tr>
</tbody>
</table>


[38] E. W. Group, Energy efficient high performance computing power measurement methodology (v.2.0 RC 1.0), 2017, https://eehpcwg.llnl.gov/assets/sc17_bof_methodology_2_0rc1.pdf.


[143] L. RIHA, A. BARTOLINI, and O. VYSOCKY, “Fine-grained application tuning on openpower hpc systems,” in *Proceedings*
of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering (PARENG), June 2019.

Curriculum Vitae

Antonio Libri received a M.Sc. degree with distinction in electrical engineering from the University of Genova, Italy, in 2013, with a thesis on Wireless Sensor Networks carried out at the 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.