Machine Learning-based Energy Aware Placement of Container over Virtual Machines

Rafael Albuquerque

A Thesis
in
Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Computer Science at
Concordia University
Montréal, Québec, Canada

July 2024

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Rafael Albuquerque**

Entitled: **Machine Learning-based Energy Aware Placement of Container over Virtual Machines**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

____ Chair
Dr. Lata Narayanan

____ Examiner
Dr. Dhrubajyoti Goswami

____ Examiner
Dr. Lata Narayanan

____ Thesis Supervisor
Dr. Brigitte Jaumard

____ Thesis Supervisor

Approved by ____
Dr. Charalambos Poullis, Graduate Program Director

July 29, 2024 ____________________________________
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Machine Learning-based Energy Aware Placement of Container over Virtual Machines

Rafael Albuquerque

The advent of 5G and the imminent arrival of Beyond 5G (B5G) have significantly increased demands on service providers. This exponential growth poses a challenge for 5G networks, since clients are currently offloading data to edge cloud servers to meet the connectivity and latency requirements. These servers must continually scale to meet increasing demands for CPU, memory, and storage, leading to significant energy consumption. Data centers account for 1.5% of global energy consumption and produce equally high greenhouse gas emissions. This trend will grow unless we find ways to improve efficiency.

Our work proposes a solution to these problems with an efficient placement algorithm backed by an accurate energy predictive model. This model helps by pre-emptively detecting the energy each machine will consume when future tasks are deployed. These predictive capabilities help the placement and reduce overall energy consumption. Our model uses Performance Monitoring Counters and various sensors, such as heat and fan speed, commonly found on Data Center machines, to increase its feature space and accuracy. Our work includes creating the model and integrating it with the energy-aware placement algorithm. Additionally, our method increased the performance and overall Quality of Service.

Our results show that our machine learning model, particularly using XGBoost, can reduce energy consumption and improve task completion times in realistic scenarios. Our experiments, tested on real servers with realistic loads, achieved good results without using stresses like stress-ng that generate unrealistic loads. Our model achieved an R^2 score of

91.2%, helping reduce energy consumption by 6% without changes to the cluster or the need for consolidation.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. Brigitte Jaumard for her dedicated support and guidance. I also want to thank all the support of Dr. Tristan Glatard and the team at GAIA Ericsson. I also would like to acknowledge Pierre Thibault for his exceptional support during this project. Finally, I want to thank my friends, family, as well as my partner Ana for their continued support during my Graduate Studies.

This research project was supported by a MITACS - Ericsson Internship and Concordia University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The advent of 5G and the imminent arrival of Beyond 5G (B5G) have brought wireless connectivity with enhanced speed, capacity, latency, and scalability. This advancement enables exciting applications such as augmented reality, autonomous vehicles, and an increased number of connected Internet of Things (IoT) devices. Managing the exponential growth in mobile data traffic remains one of the most challenging issues in 5G networks, currently addressed through data offloading to cloud servers [38].

The servers at the receiving end must continuously expand to accommodate the CPU, memory, and storage needs of their clients. As illustrated in Figure 1, this results in a significant increase in energy consumption over the coming years. With extensive resources at hand, the energy required to maintain them is substantial. Data centers (DCs) currently consume approximately 1-1.5% of global energy, contributing to 1% of all CO2 emissions worldwide. Despite some of this energy being sourced from renewable resources, the environmental impact remains significant.

In response, research into energy-reducing methods has led to efficiency improvements of 10-30% per year in recent years [36]. However, the relentless increase in consumption, coupled with our understanding of carbon's environmental effects, necessitates ongoing research into energy reduction. The data center industry continues to focus on optimizing computing sub-system resource management (e.g., sleep mode usage, VM and container scheduling, placement, and migration) to halt or slow down the growth in energy requirements.

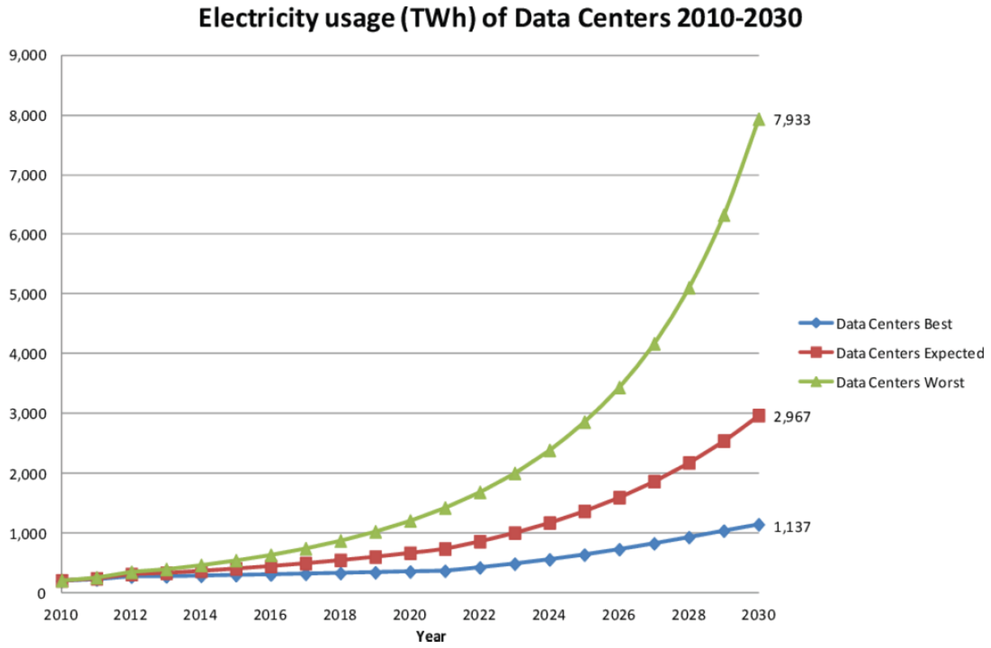


Figure 1: Extrapolation of electricity demand of DCs [17].

1.2 Background

Data Centers (DCs) are among the largest energy consumers in the computing industry by density. A medium-sized full-scale DC can host 100,000 Physical Machines (PMs) within 100,000 square feet[40]. Within these DCs, the compute system (e.g., server, networking, and storage) and the cooling subsystem are the most energy-intensive. When including the cooling system and supporting infrastructure, the energy consumption effectively doubles. This is because hotspots can significantly impact system performance, and cooling systems—comprised of radiators and compressors—often require an amount of power equivalent to that used for computing. Avoiding these hotspots is crucial, as they can cause permanent damage to the silicon. It is often recommended to implement software-side control in addition to relying on cooling systems.

Reducing energy consumption in DCs through optimization remains a critical area of research. This includes hardware changes as well as our focus: software operation and management optimization. Since most DCs today partition their PMs into smaller, easier-to-manage Virtual Machines (VMs), proper load assignment of workloads can enable energy savings, mitigate hotspot occurrences, and enhance overall performance. This thesis will focus on reducing energy consumption in container-on-VM deployments, as most workloads today are in the form of containers. The following sub-sections will provide background information on the virtualization systems used in this study, as well as an overview of the placement and orchestration of these virtual systems.

1.2.1 Virtualization systems

Virtual Machines (VMs) are among the most valuable technologies for Data Centers (DCs). The ability to partition compute resources such as CPU and RAM to provide a completely independent operating system (OS) enables DCs to scale to more powerful and denser machines without the risk of over-provisioning resources to clients. Additionally, VMs create isolated environments that run specific OSs within the PM, which is crucial for several reasons, most notably security. If an issue arises in one VM, the others remain isolated and protected.

This isolation becomes even more critical when technologies such as containers are utilized to enhance flexibility and speed. Containers, being more lightweight and OS-dependent, lack the security protections that VMs offer but excel in areas like speed and flexibility. By combining both technologies, it is possible to create clusters of sections separated by VMs for security, and within each VM, containers can be rapidly deployed to meet client needs.

As illustrated in Figure 2, a hypervisor allows the management of one or more VMs on a PM. This virtualization creates an independent OS with isolated components, similar to a real PM. Containers then provide fine-grained separation within the same VM, offering benefits such as rapid deployment and replicas management. The flexibility and security provided by these technologies, coupled with the rise of distributed software-oriented network architecture, have led to a surge in their utilization.

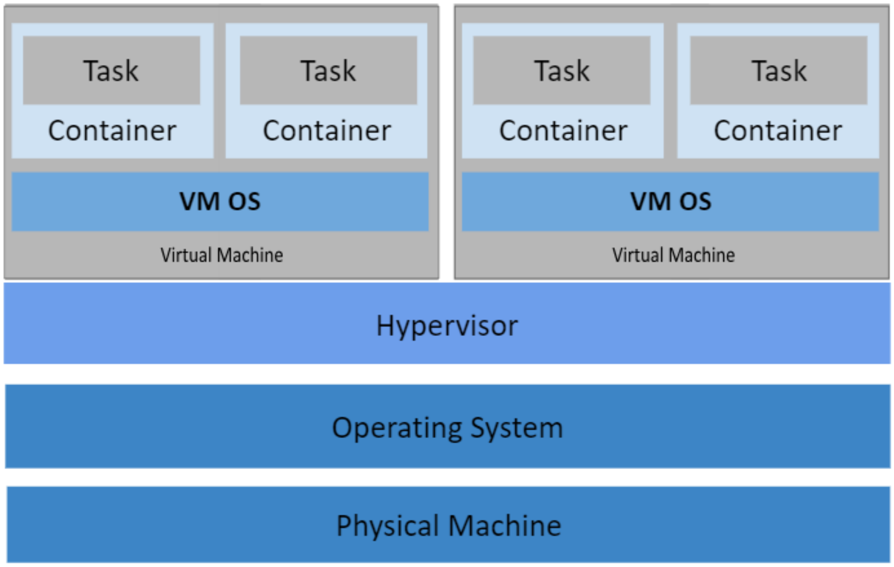


Figure 2: Virtualization layers on a Physical Machine

1.2.2 Placement and orchestration of virtual systems

The operation and management of these technologies consist of three main components: Scheduling, Placement, and Migration.

- Scheduling: This component is responsible for accepting deployment requests and determining the deployment time or order based on factors such as precedence graphs, hierarchies, or priorities. Scheduling ensures efficient deployment according to their importance or dependencies.

- Placement: Once the VMs or containers are ready for deployment, the placement component allocates them to specific hosts based on predefined policies or strategies. Effective placement optimizes resource utilization and ensures that the available computing resources are used efficiently.
- Migration: This component manages the migration of VMs from one PM to another or containers from one VM to another. Migration can be performed either while the workload is active (hot migration) or inactive (cold migration). Dynamic migration is essential for load balancing, maintenance, and minimizing downtime, ensuring that resources are utilized optimally and system performance is maintained.

These components work together to ensure efficient deployment, resource allocation, and system performance in a virtualized environment. One of the issues in the research on these problems is the risk management procedures, and since scheduling and migration usually have the highest chance of causing loss of Quality of Service (QoS) if done improperly, many studies avoid the topic. This is especifically true rue in the live transfer/migration since it requires significant network bandwidth and computing resources [6], and if done improperly could cause issues. As a result, this and many other studies choose to focus on the optimization of placement.

1.3 Project Definition: Container Placement for Energy Minimization

The goal of a container placement algorithm is to select the best host for container allocation to optimize multiple, very often conflicting objectives such as resource allocation, performance, energy efficiency, hotspot avoidance, and QoS. For the proper allocation, many tools, like analytical formulas or ML models are often used to assist the placement algorithm in optimizing this. Therefore, in this project, we aim to create an ML-assisted placement algorithm that utilizes sensors in the PMs present in data centers similar to the one done by Moocheet[26]. We aim to minimize energy consumption while maintaining QoS, ensuring proper resource allocation and high performance.

1.4 Key References

Data centers are typically designed to handle peak traffic to avoid Service Level Agreement (SLA) violations, overload, or hotspot conditions. Consequently, resources are frequently over-provisioned. Most researched methods try to reduce this energy consumption by reducing the number of active resources since the average utilization of Physical machines is only 12-18% of the DCs total capacity [40].

Some researchers advocate for consolidation strategies aimed at maximizing resource utilization and minimizing energy consumption by consolidating containers/VMs onto fewer PMs and powering down those that are not in use [4, 9, 20]. Moreover, the dynamic and unpredictable nature of cloud environments increases complexity. According to Helali *et al.* [15], uncertainty in resource provisioning poses a significant challenge due to fluctuating workloads and the difficulty in predicting exact resource needs. The volatility of VMs and containers makes accurate resource estimation challenging, complicating consolidation efforts. Providers often maintain safety margins to manage dynamic workloads, avoiding aggressive consolidation that could compromise service quality in the event of unexpected traffic spikes.

Several studies, including those by Hag *et al.* [14], Farah *et al.* [11], Ran *et al.* [30], and Al-Moalmi *et al.* [2], tackle these challenges with proactive VM consolidation strategies . These approaches leverage prediction models to forecast future resource requirements of VMs and PMs using historical data. By anticipating both current and future resource utilization, these methods enable more effective placement and migration decisions, ultimately improving efficiency and reducing the likelihood of resource bottlenecks during consolidation. These methods are an improvement over the initial ones by trying to reduce the risk of SLA breaks with predictive measures. However, these works are still incapable of completely removing the possibility of SLA, which is why safety margins are implemented. Issues such as the margin of error in predictive models or special events could generate a bottleneck, causing a massive loss of revenue due to the penalty.

For the proactive techniques, one of the main tools would be the capability of predicting the power the machine will consume in the future, especially when creating an energy aware algorithm. The work done by Fan *et al.* [10] was a classical solution introducing the choice of analytical formulas that could predict energy consumed given the resources utilized. Later, work such as the one by Zhang *et al.* [41] went a step further into the area with more accurate methods while also creating a model for prediction for containers on VM instead of only prediction of PMs.

The work done by Moocheet *et al.* [26] is our main inspiration and our work is trying to continue their work by changing focus from a VM on bare metal deployment to a Container on VM. We also aim to raise the bar for testing by utilizing real loads instead of artificial stressors and by creating a variety of loads to test our model on.

1.5 Our Contributions

Our major contributions are summarized as follows;

1. We implement predictive Machine Learning (ML) models to enhance the accuracy of power consumption prediction. This is achieved with an improved feature space that incorporates real-time data from the PMs' embedded heat sensors, internal fan

speed, and Performance Monitoring Counters (PMCs) instead of only the latter.

2. We develop a proactive, energy-aware container Placement algorithm that can reduce energy consumption while ensuring QoS above levels achieved with consolidation.

3. We performed a series of experiments on a real, private cluster of Physical Machines (PMs). These experiments provide a comprehensive evaluation of all Machine Learning (ML) models, as well as our implemented container-on-VM placement algorithm.

Up until this work, to the best of our knowledge, there is very little work that considers the temperature of components, PMCs and utilizes Machine Learning models in the context of power prediction, and container placement for VMs. Moreover, there is even less research that does this in a real data center instead of using artificial simulations and restraining from the usage of stressors.

1.6 Plan of the Thesis

This thesis is organized as follows. In Chapter 1, we presented the motivation and background of our study. Chapter 2 goes through the environment used in all experiments as well as the traffic load generation and the data collection process. Chapter 3.1 provides the methodology, experiments and results achieved with the machine learning model, as well as the energy savings algorithm. Chapter 3.1 includes the manuscript submitted to an international peer-reviewed journal for potential publication, which contains all key results generated. Finally, we conclude this thesis and discuss future work in Chapter 4.

Chapter 2

Experiment Environment

This chapter outlines our experimental setup used for the experiments of the research paper presented in chapter 3.1. Within our experimental setup, we will cover the load utilized for the experiments. Finally, we will do an overview of the data collection process done by the controller.

2.1 Cluster Architecture

All experiments have been conducted in a private data center on a cluster of 9 HP ProLiant BL460c G8 physical machines (PMs) mounted on an HP C7000 chassis. One of these PMs was setup to work as VM's hypervisor , the container orchestrator as well as the node that ran our main scripts, therefore it was not used as a worker node. As described in Table 1, each of those PMs consists of 2 processors of 8 cores, 16 threads, and 128 GB RAM.

Chassis	HP C7000 × 1
Blade/PM	ProLiant BL460c Gen8. × 8
Processor	Intel(R) Xeon(R), 8 core, 16 threads, 2.70GHz. × 2
Memory	128GB RAM (DIMM DDR3). × 2
Disk	HDD 900GB. × 2

Table 1: Cluster & Physical Machine's Description.

Since our experiment focuses on container-on-VM deployment, it involves a two-layer system, as illustrated in Figure 2 . The first layer is the hypervisor/Virtual Machine level, followed by the container level. In this section, we will sequentially address each layer.

2.1.1 VM level

Our testbed utilizes OpenStack as the cloud platform for managing Virtual Machines (VMs) and adheres to a three-node architecture, consisting of three systems: the controller node, compute node, and storage node [28] (refer to Figure 3).

The controller node serves as the manager, running most of the OpenStack services and providing API, scheduling, and other shared services for the cluster. Eight PMs function as compute nodes, where VM instances, also known as Nova compute instances, are deployed. Although there is an additional node operating as a storage node, its utilization and function are abstracted from our problem, as it functions as storage for the worker VMs.

Furthermore, our controller node has SSH access to each PM and VM, allowing it to upload any necessary scripts.

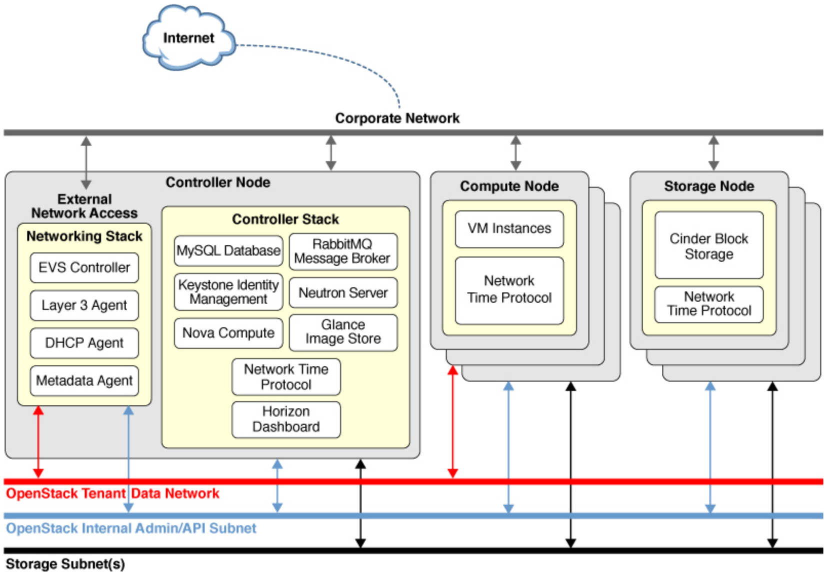


Figure 3: Three-Node Architecture of OpenStack [28]

Table 2: Example of configuration values for the container load

5G services	CPU	RAM	IO
Cloud training	65% × 4	4 GB	0 Gb/s
Gaming	49% × 2	2 GB	1 Gb/s
Data Base	55% × 3	8 GB	2 Gb/

2.1.2 Container level

Our testbed utilizes Kubernetes as the container orchestrator for all Docker containers used in our experiments, following a container-on-VM architecture. This architecture is configured such that any VM instance is recognized as a standard worker node by Kubernetes and can serve as a target for container deployment. With eight physical machines (PMs) hosting a total of three VMs each, we had a total of 24 VMs functioning as Kubernetes worker nodes.

2.2 Experiments

2.2.1 Load emulation

For our load emulation, we created our own containers from: (i) a dockerfile, (ii) a kubernetes yaml file, (iii) a bash script, and (iv) the python scripts. Example of the configuration of container can be seen in table 2, and each resource is defined as follow:

- The Dockerfile is used by Docker to build the container itself, providing rules and configurations for the image construction. These rules and configurations include the installation of the Python version, dependencies, copy of files that the container will run, and setting up all environment variables that the container will use. These environment variables are crucial as they define the workload and the level of utilization for CPU, RAM, and I/O, thereby determining the differentiation of all container loads created.

- The Kubernetes YAML file is responsible for configuring all Kubernetes settings for the deployed container. In our case, this file specifies the VM on which the container will be deployed, the resources required by the chosen type of container, and the amount of work and utilization for the PMCs. Since the latter must be passed to the Python script, it is written in the environment variables, which will be read by the Bash script upon initialization.

- The Bash script acts as the initializer since the container must execute one file as a starting point when it is deployed. This executable reads the environment variable values provided by the YAML file. It initializes all three Python scripts, one for each resource, and assigns each the respective configuration variable.

- Finally, each Python script receives the input of the amount of work and utilization it will handle for its respective task. For instance, the CPU stressing script manages CPU stress, and so forth.

We wanted all of our containers to perform a specific number of operations before terminating, rather than using time-based load, as the latter would replicate normal stressors such as Stress-ng [34]. For this purpose, we defined one "main" resource as the defining factor for each container (e.g., Cloud Training is defined by a specific amount of CPU work). This main resource would receive a limited amount of work to complete before the container would finish its operation. Meanwhile, the other resources would operate indefinitely until the main resource's task was completed. This approach simulates scenarios where containers require different amounts of time to finish depending on their respective resource availability.

2.2.2 Data collection

For data collection, the process is conducted in parallel from each worker to the controller. The procedure operates as follows: During initialization, the control node connects via SSH to every PM and initiates a data collection script for the PMCs and IPMI sensor values. This script, located on the worker node, periodically samples the PMCs and IPMI sensor data and reports back by writing the new data to a file on the master node, corresponding to that PM. This file is exclusively written by that PM and only read by the master node. For the VMs, the process is similar, but only the collection of PMCs is performed, as the sensors are not available in a virtual environment. Following this process, whenever data collection is needed for a specific VM, the master script merges the data collected on that VM with the data from the PM hosting it, resulting in the complete data sample for a given time.

The secondary data collection in this work involves logging occurrences in the Kubernetes environment, such as the start and end times of containers. Since this data is only available from the master node, no script upload is necessary; instead, a single thread on the controller oversees and logs the events. This script maintains a constant connection to the Kubernetes API, and whenever an event labelled 'START' or 'END' is triggered, the script logs the timestamp, the IP location where the container started or ended, the container name, and the event name.

These two logs were sufficient to generate a training and testing dataset used in training the ML models, as the exact deployment times could be traced back in the measurements file. The process of generating the training data from this will be further explained in Section 3.4.1.

Chapter 3

Energy Aware Placement of Containers over Virtual Machines

This chapter has been submitted as a Journal paper titled "Energy Aware Placement of Containers over Virtual Machines" written by R. Albuquerque, B. Jaumard, and P. Thibault.

3.1 Abstract

The advent of 5G and the upcoming arrival of 6G have significantly increased required cloud computing resources, especially for the edge cloud servers to meet the connectivity and latency requirements.

Many studies attempt to address these energy concerns in ways that seem very interesting but are not always viable for real deployments, such as tight consolidation techniques without safety margins for uncertainty in traffic prediction. These implementations show increased reductions in consumption but lack practicality for deployment in large data centers.

This study proposes an energy-aware machine learning placement algorithm that uses sensor data from Physical Machines (PMs) in data centers to predict energy consumption

and optimize container placement.

Computational experiments were conducted on a testbed with 5G realistic traffic scenarios, without the use of stresses such as stress-ng that generate artificial traffic loads with respect to compute resource utilization. Our results demonstrate that the proposed machine learning models, particularly the XGBoost one, can reduce energy consumption and improve task completion times, without the use of an explicit consolidation strategy. Indeed, in the context of container placement, our model obtained R^2 score results of 91.2%, which reduced energy consumption by 3% without the need to modify the cluster or consolidate it.

The advent of 5G and soon 6G brought wireless connectivity with enhanced speed, capacity, latency, and connectivity, enabling applications such as augmented reality, autonomous vehicles, and more connected IoT devices. Managing the exponential growth in mobile data traffic is a significant challenge in 5G networks, currently addressed through data offloading to (edge) cloud servers [38].

The servers at the receiving end must continuously grow to accommodate the CPU, memory, and storage needs of clients, leading to substantial energy consumption. Data centers (DCs) use around 1-1.5% of global energy and contribute to 1% of global CO2 emissions. Despite some energy coming from green sources, there is an ongoing need for energy-reducing methods to improve efficiency, achieving 10-30% improvements per year in recent years [36].

A popular method of minimizing energy is to reduce the number of active resources as much as possible since, in practice, the average utilization is often only 12-18% of the total capacity [40]. Service providers must maintain sufficient resources for peak demand, resulting in idle resources during non-peak times. This creates opportunities for energy-efficient task deployment by distributing traffic to minimize average consumption across hosts.

An accurate energy forecasting model is essential for successful energy-aware task placement and scheduling. Such a model helps identify which machine will have a smaller increase in energy consumption after deploying a container. Current solutions often use purely analytical models, suitable for small data centers but inadequate for global-scale data centers, since their linear aspect could overlook components like memory or I/O and external effects like component temperature.

In this project, we aim to create an energy-aware ML container placement system using sensors in PMs, building on the work by Moocheet *et al.* [26]. These sensors, typically used for hotspot detection to prevent damage and downtime [13], can also enhance our predictive model by providing data on heat, fan speed, power, and system status. We aim to improve upon Moocheet's work by developing a model capable of predicting multiple tasks with finite running times in a container-on-VM scenario, rather than a bare metal one. This added virtualization increases complexity, as it requires extracting information such as exact CPU utilization and power consumption from the virtualized environment, thus addressing a less-studied deployment type.

This paper is organized as follows. Section 3.2 presents the literature review, including other works on which this work is based. In Section 3.3.1, we will describe the problem statement as well as show possible problems with other solutions. Then, Section 3.4 details the traffic used for our experiments, the containers as well as the testbed used for the experiments and its sensors. After that, Section 3.5 reviews the models used and compared in the energy forecasting part of the study. Finally, Sections 3.8 and 3.9 are reserved for the numerical results and our conclusions.

3.2 Literature Review

Cloud computing has become an integral part of how to access, store, and share information, with three main infrastructure types: Bare Metal, Virtual Machines (VMs), and

Containers, which are not mutually exclusive as both VMs and containers run on top of bare metal servers, while containers can also be deployed inside VMs. The key differentiator between containers and Virtual Machines is that VMs virtualize an entire machine down to the hardware layers and containers only virtualize software layers above the operating system level.

Even though both types of applications are different, they do have some things in common. The requirement for an energy predictor is probably the biggest one, and is one of the main tools for good placement of virtualized elements. Most solutions nowadays require the ability to at least estimate the power used after the deployment of a given load. Therefore in this literature review, we will go over a few of the specific methods and improvements done by the literature for energy prediction in physical hosts as well as virtual hosts before moving on to placement. It is worth noting that most of these works do not fit perfectly into our solution since their method predicts the power consumed P_{t_1} at the time of feature sample t_1. However, the energy aware placement problem requires to predict P_{t_2} given the features of t_1, P_{t_1} and the container L or the expected increase in metrics by the load L : L_{CPU}, L_{RAM}, L_{IO}, ... Therefore they were used for inspiration but adaptations and small changes would need to be done.

3.2.1 Power prediction for physical machines

Several researches were done on PMs and one of the main classical examples is Fan *et al.* [10] which created a simple linear regressor that could track the relationship between the CPU and the power consumed. Many other implementations follow that idea [18] since the CPU is the main consumer, its utilization alone is somewhat linearly correlated to the power, plus the idle consumption as baseline [29].

One main advantage of these methods is the simplicity of it all being small and easy to implement, however, many works fail to notice that most DCs are also providers for a variety of services, including non-CPU intensive tasks[16]. Therefore, in a given distribution of container/VMs deployed, it is possible that the usage of other resources would be a limiting factor for a given PM instead of CPU utilization. Furthermore, these models may occasionally employ small adjustments, such as the calibration method used by Fan *et al.* [10], to reduce error. This approach sacrifices simplicity without achieving the full accuracy of a non-linear model.

Other works took into consideration components other than the CPU, leading to more complex, non-linear models. The work of Liang *et al.* [23] for example, included the energy consumed by the memory in their solution.

The integration of temperature into the power prediction model is also another addition to the feature space that can bring benefits and is something not yet fully explored. The work done by Rezaei *et al.* [32] exposed the effects of temperature in energy consumption, as well as created a model that improves the prediction of the baseline model by Fan *et al.* [10]. However, it has to be noted that temperature plays a role only when becoming rather high by the standards of data center rooms.

3.2.2 Power prediction for virtualized systems

For non-physical systems commonly used in DCs, such as VMs or Containers, the difficulty of power prediction increases considerably since there will now be an extra layer of virtualization. Furthermore, the usage of new kernels like the ones in VMs could change the power profile of the PM. VMs will also share underlying hardware resources, making changes to it in a way that different PMs would not, like increasing the temperature of components. Methods have been developed to deal with this increased complexity, such as the work done by Bohra *et al.* [5] with their host energy partitioning method. Many of these methods use the consumed energy P_{t_1} alongside the resource usage of the virtualized environment to distribute the energy usage accordingly for each VM. There are other similarly complex works, such as the one by Zhang *et al.* [41], that define a container as a sum of processes in the VM or PM which then measure the power consumed by those processes. These work sometimes manages to go down to the most basic level of definition of VMs in an attempt to very accurately estimate the power consumed only by that virtual system. These would be able to track in real time the energy consumed by the containers.

3.2.3 Container and virtual machine placement

Numerous studies have extensively investigated the performance implications of containerized tasks both on VMs and on bare metal platforms [21, 12, 3]. They consistently demonstrate that VM-based deployments offer greater flexibility and isolation, albeit at the expense of higher CPU and memory overhead due to increased virtualization. However, as shown by the research of Sultan *et al.* [35] modern DCs must accommodate both deployments, as relying solely on bare metal deployments can pose security risks [31].

On that note, other works tried to utilize the increased flexibility provided by the added virtualization to better allocate and balance load as a method to increase consolidation performances, such as the work done by Kaur *et al.* [19] and Al-Moalmi *et al.* [2]. These studies follow the trend of many others attempting to optimize the placement of containers and VMs for future increased consolidation, reducing the number of idle/powered-off machines as much as possible. While these methods are valid ways to reduce power consumption overall, they raise some issues, such as the possibility of causing overload, that need to be considered in real-world deployments which will be discussed in more detail in Section 3.3.3.

Some works try to reduce problems and risks taken during consolidation by employing proactive measures in the system, using technologies for power or resource predictions to anticipate any problems that could occur, such as the works done by Hag *et al.* [14], Farah *et al.* [11], and Ran *et al.* [30]. These methods do reduce risk but are still not completely safe since they always try to optimize for better and tighter consolidation.

3.2.4 Concluding remarks of the literature review

Each publication discussed here has both advantages and disadvantages, focusing on new components to enhance power prediction and placement/scheduling. However, none have integrated all these technologies into a single solution. Our goal is to develop a dynamic container-on-VM sensor-driven placement agent utilizing a data-driven model instead of analytical formulas for power prediction. This model will account for component utilization and external factors like temperature to optimize performance in real-world scenarios and will be tested on a test-bed, similar to the work of Moocheet *et al.* [26]. However, in contrast to the work mentioned, we aim to create a system closer to real-life operation, where tasks have finite running times and varying loads, all without the need for manual tuning of parameters per machine or tasks.

3.3 Problem statement: energy aware container placement

We will discuss here the detailed problem statement of our study as well as our motivations and consequences one could encounter when trying to implement consolidation methods as mentioned in Section 3.2.3.

3.3.1 Problem statement

Our goal is to study the load placement problem where we minimize the overall energy consumed in a cluster without loss of QoS. Each load is defined as a container with CPU, RAM and I/O requirements for a respective Python script running doing some work. Finally, each Physical Machine in the DC will be split into VMs with different pre-defined sizes just like it would be in a production application. For this goal, we also want to set a higher standard for evaluating our placement method, and with the use of real load instead of artificial stressors. Deployment and testing in a real data center (test bench) will provide an additional advantage in testing our methods. In the process of creating an agent that can correctly place a container load while reducing energy, an energy predictor is needed. We chose to use an ML-based model because we believe that classical linear models do not adequately capture the complexity of energy aware placement.

3.3.2 Motivation

Each Data Center is unique, especially when considering large data centers. That is because technology is always evolving, and DCs constantly replace old parts with new ones [1]. This creates an environment that is hard to simulate accurately, with lots of components with different efficiencies and specific quirks. An argument against purely analytical formulas and simulations is that they will give unrealistic results if they are not properly tuned and adjusted for each PM. Since in real-world scenarios it is likely that continuous changes will occur in the DC, this tuning could be avoided by using models that learn over time the specific parameters of these internal factors affecting the PMs. There are also other external factors, like the temperature of adjacent machines, the current leakage of that silicon chips, and even the air itself coming from the intake that can affect the temperature of components and their performance [24]. These are some of the reasons why PMs have small differences between them [25].

In one of our experiments, the same load was executed on two different PMs with states as close as possible. Figure 4 shows the differences in consumption and performance when running the same load on two different PMs of the same data center. From Figure 4, we trace the energy profiles of two PMs of the same DC, and observe that there is a notable difference in the energy profiles of the machines, and that it is interesting to exploit this difference to reduce energy. Therefore, using data-driven models, defined from the cluster

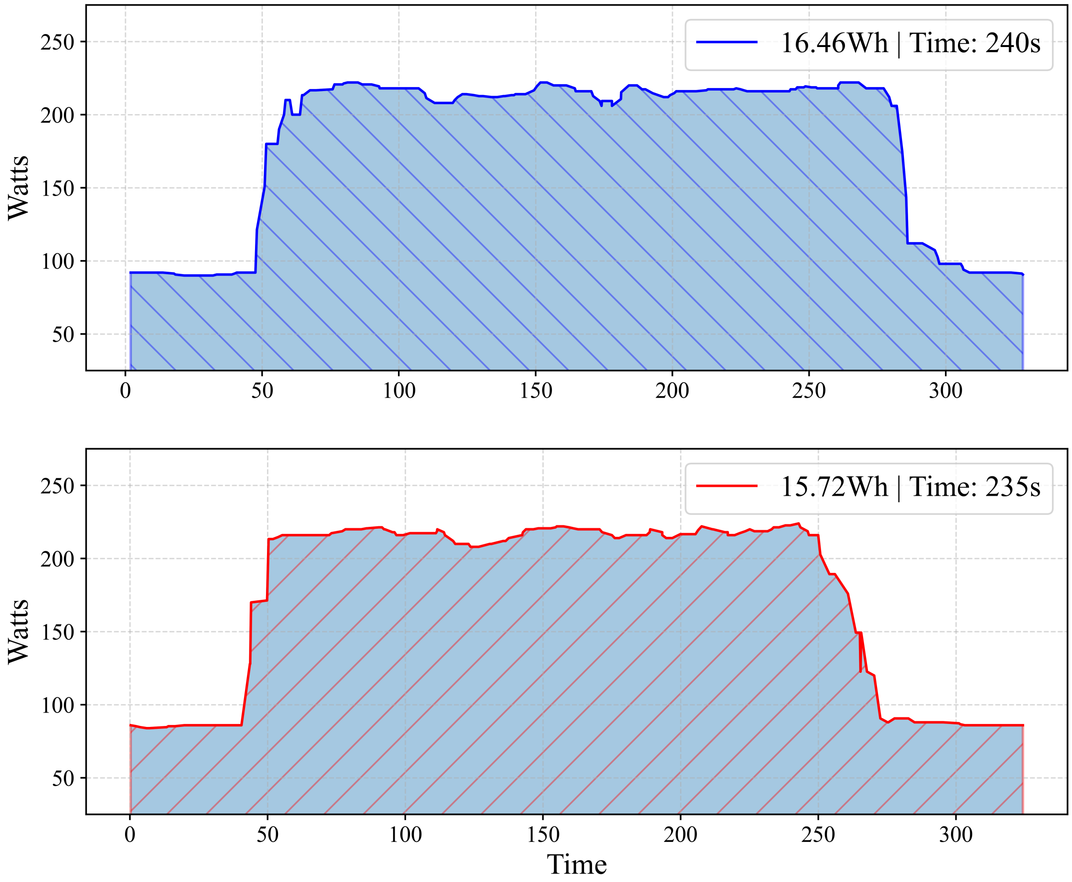


Figure 4: Comparison of different PMs with same configuration and load

itself, it is possible to obtain accurate energy consumption values and learn to predict the peculiarities of each PM. Finally, with the expected energy consumption and efficiency after deploying a given container in each of the possible VMs, we expect to obtain reduced energy values.

3.3.3 Consolidation methods

Some researchers advocate consolidation strategies aimed at maximizing resource utilization and minimizing energy consumption by consolidating containers/VMs onto as few

PMs as possible and powering off those that are not in use [4, 9, 20]. Specifically, consolidation works by shutting down the maximum possible number of PMs and deploying any task arriving into those available PMs, and once those hosts reach a specific level, a new PM is deployed.

A perfect consolidation scenario will deploy a new machine at the same time the previous set of machines is full. Once tasks are completed, any future task will prioritize filling already deployed PMs, and if the overall load decreases, subsequent PMs may be deactivated again. However, such an approach often neglects critical factors such as component temperature dynamics [24] which could cause damage and overheating, leading to throttling and loss of performance. Another downside that cannot be easily quantified would be the issue of completely shutting down PMs and the risk of possible loss of Quality of Service (QoS) if the new needed PMs are not deployed in time.

Additionally, the dynamic and unpredictable nature of cloud environments adds to the complexity. Uncertainty in resource provisioning, as described by Helali *et al.* [15], poses a significant challenge due to the highly fluctuating nature of workloads and the difficulty in predicting exact resource needs. The volatility of workloads makes resources difficult to estimate accurately, which complicates consolidation efforts. Providers often prefer maintaining safety margins to handle dynamic workloads, avoiding aggressive consolidation that might compromise service quality if unexpected traffic arrives.

While consolidation is a valid approach, one must consider the scenario where it is not applicable, or where it has already been used up to a safety limit, and there is a desire to optimize the usage of remaining resources. Therefore for this research, we are interested in focusing on the reduction of energy without the use of shutdowns or sleeps. For this, as a baseline, we will use a modified version of the consolidation method where the minimum idle consumption of the machine will still be taken into consideration even when underutilized.

3.4 Traffic load and testbed

3.4.1 Traffic load

The 5G networks that have emerged in recent years, and the upcoming 6G, rely heavily on cloud services for many functionalities, including the execution of virtual network functions. In this study, we focused on the computing resource requirements of traffic from 5G services, as it uses a significant fraction of allocated resources and energy and is expected to continue to grow [22].

Building containers that perform 5G virtual network function tasks is challenging and no data is available. However, we can emulate these tasks by creating containers that require representative computing resources to host the virtual network functions on the servers. Examples include streaming servers, which typically require different resources (CPU or memory) than gaming servers.

We designed a traffic load based on a mix of 5G services, each with its own resources requirements, as described in Table 3. Each container was configured with three Python scripts, each generating load on different components: CPU, RAM, I/O, as shown in Figure 5. By adjusting the values in each script, we can simulate various tasks.

For these experiments, we ensured that the tasks were not simulated using a stress factor such as stress-ng, so as not to mask variations in efficiency or performance between the different machines (PMs). Indeed, a stressor constantly strives to maintain a constant percentage of load on the machines for the same amount of time, thereby masking their inherent differences. In our results, in Section 3.8, we will show differences in completion time caused by placement. These differences would be different with the use of artificial stressors such as stress-ng.

The number of processors is indicated in the table 3 and it depends on the number of cores allocated to the container, and since the nature of the CPU script is multithreaded, it

Table 3: Characteristics of the compute resources of the different 5G services

5G services	CPU	RAM	IO
Streaming (big)	36% × 3	8 GB	4 Gb/s
Streaming (small)	40% × 3	8 GB	4 Gb/s
Cloud training	65% × 4	4 GB	0 Gb/s
Website hosting	40% × 2	2 GB	1 Gb/s
Gaming	49% × 2	2 GB	1 Gb/s
Database	55% × 3	8 GB	2 Gb/s

uses the entire number of cores. Therefore if the CPU is 36% X 3 it means that it uses 3 cores.

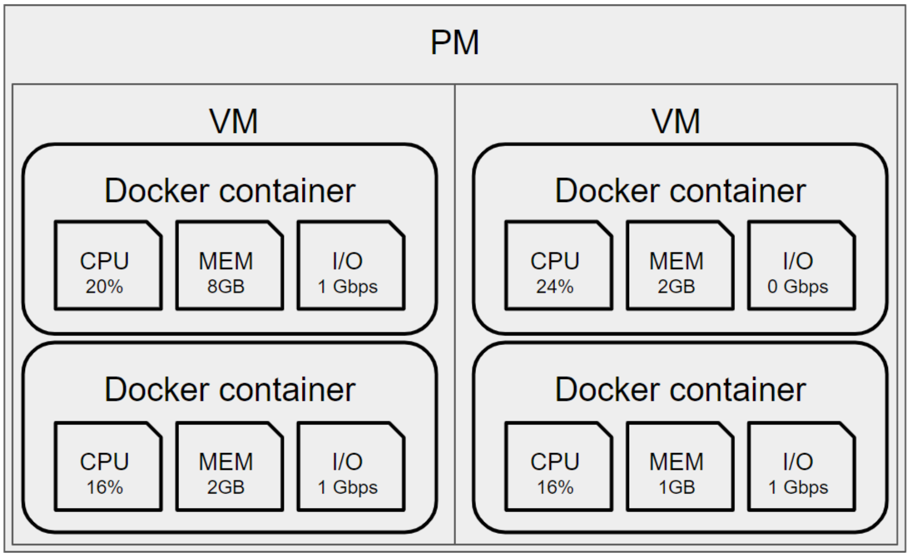


Figure 5: Two typical examples of VMs of our present architecture in a HP ProLiant BL460c G8

3.4.2 Testbed

Due to limited access to nodes in a live large data center, we developed a testbed in a private data center with 9 dedicated HP ProLiant BL460c G8 servers. This setup ensured no extra activities interfered with our experiments, which included the kernel, dependencies, and data collection algorithms. Each Physical Machine (PM) comes with embedded sensors and a power meter, capable of collecting two categories of data: Performance Monitoring Counters (PMCs) and sensor data.

PMCs are frequently used by administrators to monitor and control system performance, focusing on metrics like CPU utilization, RAM memory access rate, and hard disk I/O utilization. We used PSUtil [33] for the collection of PMCs, since it can do real-time data collection for logging and inference purposes.

Today data centers use multiple sensors to monitor and manage PMs, preventing high temperatures that can damage equipment and increase power consumption due to silicon leakage current. We chose Intelligent Platform Management Interface (IPMI) to collect these sensor values, granting us real-time temperature data and fan speeds, enhancing our model's accuracy. Figures 6 and 7 illustrate sensor locations and their readings.

Kubernetes managed container orchestration, while OpenStack handled VM deployment and management. hosts were defined as VMs with two configurations: small (8 CPUs, 16 GB RAM) and large (double the resources). Eight PMs served as compute nodes, with one as the controller node for data collection and VM/container management.

We implemented the proposed energy-aware container placement algorithm in Python 3.12, utilizing the Kubernetes library and other supportive libraries. The script managed machine communication, worker status messages, and hosted the machine learning model for predicting and deciding container placement. Our setup also leveraged multiple other APIs to control container placement, VM management, and worker statuses like SSH, PSUtil, and IPMI.

3.5 Data collection and training

We discuss here the process done to generate the data that would be used for the ML model. On subsection 3.5.1 we explain the work behind the container generation, its deployment and how the real-time sensor and PMC collection happened. Then, in subsection 3.5.2 we

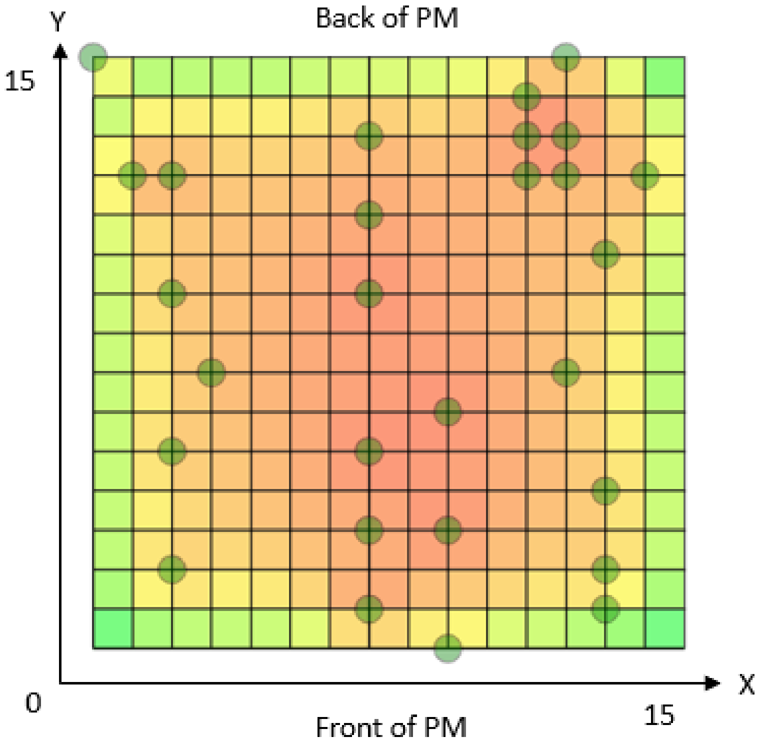
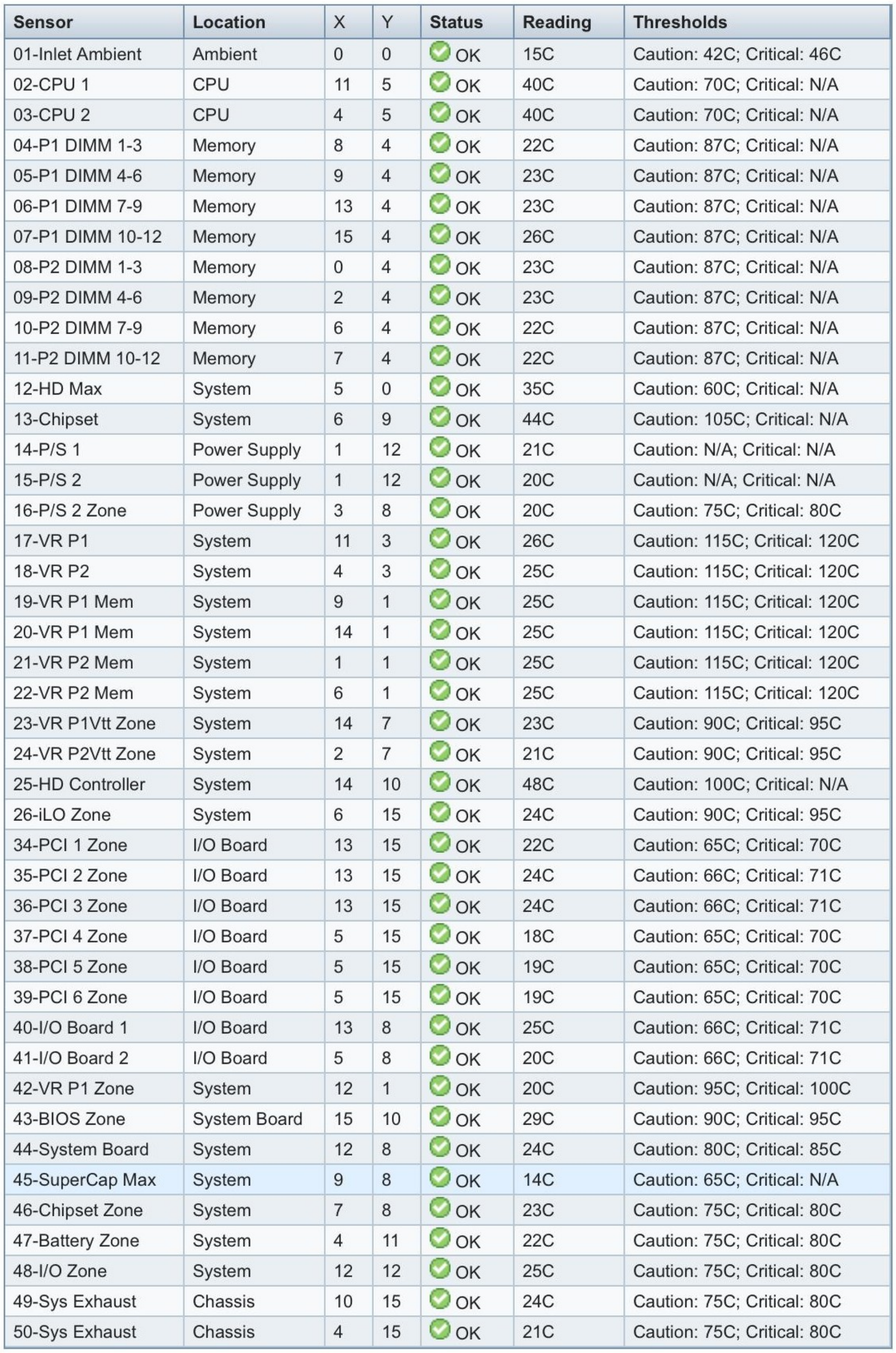


Figure 6: 2D heat map of temperature based on sensors location of a typical HP ProLiant BL460c G8



Sensor	Location	X	Y	Status	Reading	Thresholds
01-Inlet Ambient	Ambient	0	0	✓ OK	15C	Caution: 42C; Critical: 46C
02-CPU 1	CPU	11	5	✓ OK	40C	Caution: 70C; Critical: N/A
03-CPU 2	CPU	4	5	✓ OK	40C	Caution: 70C; Critical: N/A
04-P1 DIMM 1-3	Memory	8	4	✓ OK	22C	Caution: 87C; Critical: N/A
05-P1 DIMM 4-6	Memory	9	4	✓ OK	23C	Caution: 87C; Critical: N/A
06-P1 DIMM 7-9	Memory	13	4	✓ OK	23C	Caution: 87C; Critical: N/A
07-P1 DIMM 10-12	Memory	15	4	✓ OK	26C	Caution: 87C; Critical: N/A
08-P2 DIMM 1-3	Memory	0	4	✓ OK	23C	Caution: 87C; Critical: N/A
09-P2 DIMM 4-6	Memory	2	4	✓ OK	23C	Caution: 87C; Critical: N/A
10-P2 DIMM 7-9	Memory	6	4	✓ OK	22C	Caution: 87C; Critical: N/A
11-P2 DIMM 10-12	Memory	7	4	✓ OK	22C	Caution: 87C; Critical: N/A
12-HD Max	System	5	0	✓ OK	35C	Caution: 60C; Critical: N/A
13-Chipset	System	6	9	✓ OK	44C	Caution: 105C; Critical: N/A
14-P/S 1	Power Supply	1	12	✓ OK	21C	Caution: N/A; Critical: N/A
15-P/S 2	Power Supply	1	12	✓ OK	20C	Caution: N/A; Critical: N/A
16-P/S 2 Zone	Power Supply	3	8	✓ OK	20C	Caution: 75C; Critical: 80C
17-VR P1	System	11	3	✓ OK	26C	Caution: 115C; Critical: 120C
18-VR P2	System	4	3	✓ OK	25C	Caution: 115C; Critical: 120C
19-VR P1 Mem	System	9	1	✓ OK	25C	Caution: 115C; Critical: 120C
20-VR P1 Mem	System	14	1	✓ OK	25C	Caution: 115C; Critical: 120C
21-VR P2 Mem	System	1	1	✓ OK	25C	Caution: 115C; Critical: 120C
22-VR P2 Mem	System	6	1	✓ OK	25C	Caution: 115C; Critical: 120C
23-VR P1Vtt Zone	System	14	7	✓ OK	23C	Caution: 90C; Critical: 95C
24-VR P2Vtt Zone	System	2	7	✓ OK	21C	Caution: 90C; Critical: 95C
25-HD Controller	System	14	10	✓ OK	48C	Caution: 100C; Critical: N/A
26-iLO Zone	System	6	15	✓ OK	24C	Caution: 90C; Critical: 95C
34-PCI 1 Zone	I/O Board	13	15	✓ OK	22C	Caution: 65C; Critical: 70C
35-PCI 2 Zone	I/O Board	13	15	✓ OK	24C	Caution: 66C; Critical: 71C
36-PCI 3 Zone	I/O Board	13	15	✓ OK	24C	Caution: 66C; Critical: 71C
37-PCI 4 Zone	I/O Board	5	15	✓ OK	18C	Caution: 65C; Critical: 70C
38-PCI 5 Zone	I/O Board	5	15	✓ OK	19C	Caution: 65C; Critical: 70C
39-PCI 6 Zone	I/O Board	5	15	✓ OK	19C	Caution: 65C; Critical: 70C
40-I/O Board 1	I/O Board	13	8	✓ OK	25C	Caution: 66C; Critical: 71C
41-I/O Board 2	I/O Board	5	8	✓ OK	20C	Caution: 66C; Critical: 71C
42-VR P1 Zone	System	12	1	✓ OK	20C	Caution: 95C; Critical: 100C
43-BIOS Zone	System Board	15	10	✓ OK	29C	Caution: 90C; Critical: 95C
44-System Board	System	12	8	✓ OK	24C	Caution: 80C; Critical: 85C
45-SuperCap Max	System	9	8	✓ OK	14C	Caution: 65C; Critical: N/A
46-Chipset Zone	System	7	8	✓ OK	23C	Caution: 75C; Critical: 80C
47-Battery Zone	System	4	11	✓ OK	22C	Caution: 75C; Critical: 80C
48-I/O Zone	System	12	12	✓ OK	25C	Caution: 75C; Critical: 80C
49-Sys Exhaust	Chassis	10	15	✓ OK	24C	Caution: 75C; Critical: 80C
50-Sys Exhaust	Chassis	4	15	✓ OK	21C	Caution: 75C; Critical: 80C

Figure 7: Sensor description and reading examples

explain the pre-processing done before the data was ready to be fed into the models. And finally, in subsection 3.5.3 we explain the training process that will be applied to all of the ML models explained in the following section 3.6.

3.5.1 Data collection

The data collection process is crucial for training the ML model. Initially, we used a pseudo-random generator to create containers with specified start times and compute resource requirements (RAM, CPU, I/O). PSUtil and IPMI continuously logged the status of the machines (PMs) and containers. Simultaneously, a Python daemon script interacted with the Kubernetes API to collect all events (e.g., new or ending containers) within the namespace. This allowed us to capture the exact start and end times for each container.

By cross-referencing these datasets, we could analyze how events affected PM resources, temperature, and power. The pseudo-random generator and data collection APIs enabled us to generate comprehensive data on PM behaviour during container deployment, covering various scenarios over time. For instance, we could examine resource usage when a "streaming" container is deployed on a host already running "gaming" and "cloud training" containers.

Algorithm 1 details the process of generating data from the measurement datasets (D_m) collected via IPMI and PSUtil, and the event dataset (D_k) from the Kubernetes daemon script. This process was repeated with varied configurations (e.g., different numbers of containers per hour, varying intervals between containers) to generate diverse data scenarios.

The process is also explained in Figure 8, where we can see how the sample is built, with information from before making up the features and the average power being the target. All the values on the before and after window are averaged to give us a more stable result since some sensors and power meters collect instantaneous values and can be quite

Algorithm 1 Data Generator

```
procedure DATA GEN($D_m, D_k$)
    Load IPMI and PMC measurements dataset $D_m$
    Load kubernetes container log dataset $D_k$
    $S \leftarrow$ Start of all containers in $D_k$
    $E \leftarrow$ End of all containers in $D_k$
    for $host$ in $hosts$ do
        $S_h, E_h \leftarrow$ filter by host
        for $Container$ in $S_h$ do
            $Cont_t \leftarrow GetStartTime(Container)$
            if $CheckEvents(Cont_t - 45, Cont_t + 90)$ then
                Skip Iteration
            $before \leftarrow AllMeasurements(Cont_t - 45, Cont_t)$
            $after \leftarrow Power(Cont_t + 45, Cont_t + 90)$
            $state_{before} \leftarrow mean(before)$
            $power_{after} \leftarrow mean(after)$
            $SaveSample(state_{before}, Cont, power_{after})$
```

volatile. This process will generate the data used for the training of all the models.

3.5.2 Data pre-processing

Data processing is a common step when dealing with machine learning models, as they can easily be over-fitted, under-fitted, or simply fail to learn if the data they contain are not properly processed. If a model is not fed with proper data, it will generate inaccurate predictions. We therefore want to minimize noise and fluctuations, making it easier for the ML model to learn properly. Also, since our environment is dynamic and sometimes unstable (problems or delays in any of the APIs or ssh could affect data collection), we must clean the data whenever we have these malfunctions. To resolve these issues, we perform a process similar to scraping corrupted data during data collection, where if there is missing data or an impossible value, the sample will be removed. The data set was then scaled to standardize the features, this ensures that each feature contributes equally to the model, avoiding situations where big number would affect the model unfairly, and improving its

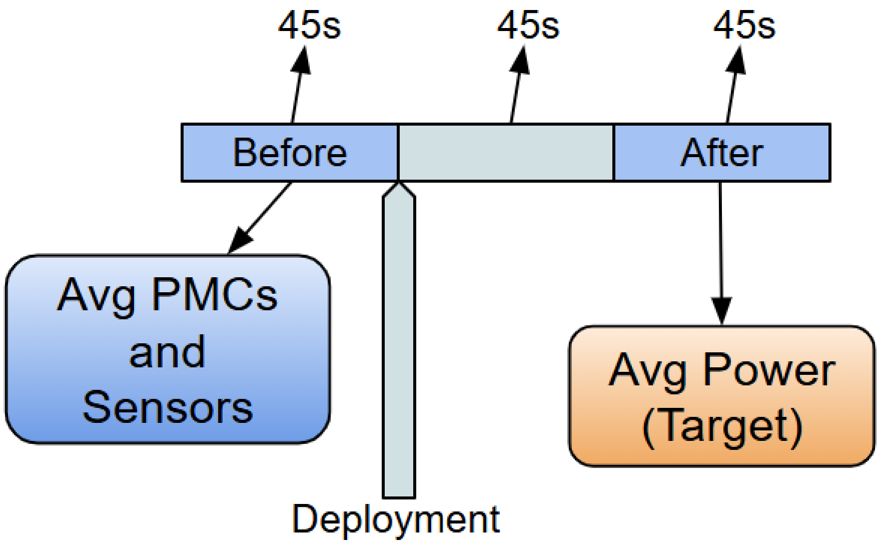


Figure 8: Figure of how a sample is built

performance and stability. The standardization formula is as follows:

$$z = \frac{x - \mu}{\sigma}, \tag{1}$$

where z is the standardized value, x is the original value, μ is the mean of the feature, and σ is the standard deviation of the feature. We also did a one-hot encoding for non-numeric features, such as the container that was added. One-hot encoding is a technique used to convert categorical variables into a binary (0 or 1) representation since machine learning models cannot accept non-numerical features.

3.5.3 Training

For the training process, the data collected through random placements was merged in a single data frame generating a training data of 6589 samples split between different destinations and categories of deployed containers. From this new dataset, 20% was split into the testing set, while the rest was kept for training. Since we also desired to better understand the effects of the increase feature space, we generated four subsets of this dataset:

1. Dataset C: A dataset where we only have CPU utilization information, as well as the current power meter, and the container deployed.
2. Dataset CM: A dataset where we have the information of dataset C, but also an added information of the memory utilization.
3. Dataset CMI: A dataset where we have all PMCs, with CPU, RAM, and I/O utilization, as well as the current power meter and the container deployed.
4. Dataset CMIS: This dataset is the complete dataset, with all the previous information as well as the sensor information collected(e.g. temperature sensors, fan speed ...).

After the dataset split, we train every chosen model (specified in the following section 3.6) with all variations of the dataset, with the exception of our analytical formula used as baseline, since this solution requires especific features.

3.6 Energy prediction models

The usual utilization of our models during the production is represented in Figure 9. The data is collected from both the sensors and the PMCs in the host, this is then fed into the model alongside the new load. The model will then generate a list of predictions for energy variations in all possible hosts and order them from lowest to highest. This will be used as the priority of deployment for the container placement algorithm.

However initially we must evaluate their performance, and we can do it artificially, using the full dataset generated randomly and feeding it for training and testing. The next subsections will go through every model used, as well as the analytical formula used for comparison.

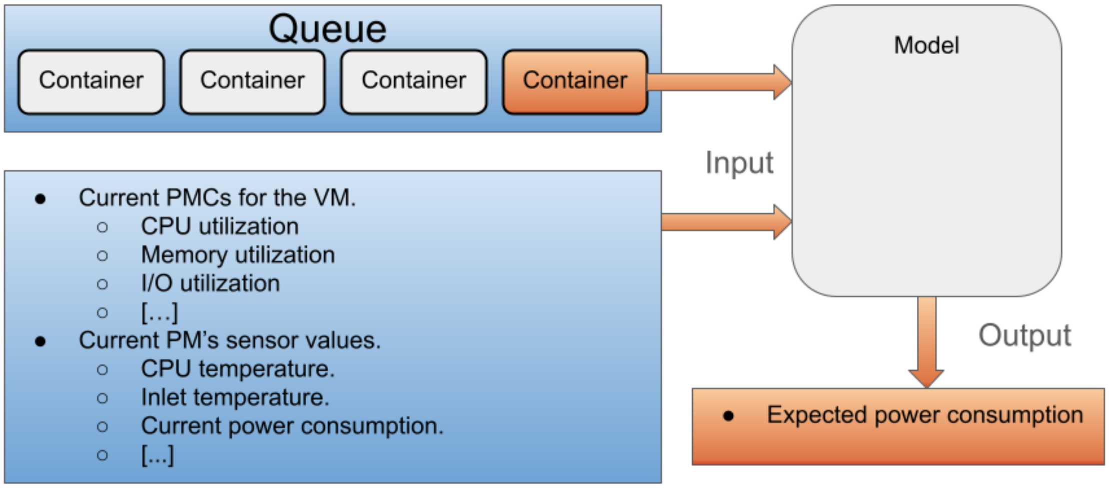


Figure 9: Model prediction architecture

3.6.1 Analytical formulas

The formula proposed by Fan *et al.* [10] was used by many studies, including very recent ones, as the baseline model and is written as follows.

$$P^{\text{Fan } et\ al.}(u) = P_{\text{idle}} + (2u - u^r)(P_{\text{busy}} - P_{\text{idle}}) \tag{2}$$

where *u* is the CPU utilization scaled to range (0, 1), $P(u)$ is the PM predicted power consumption at overall CPU utilization *u*, P_{idle} and P_{busy} are the power consumption of the PM at idle state and peak usage state, respectively. r is a calibration parameter to fit the model to minimize the forecast error. The formula (2) is a good starting point. However, there are more complex formulas that are still easy to calculate even if they include additional measurements, such as inlet temperature, resulting in increased accuracy without losing too much simplicity. The chosen one was the formulation proposed by Wang *et al.* [39], and is defined as follows:

$$P^{\text{Wang } \textit{et al.}}(u) = \mathrm{P}_{\text{idle}} + U_{\text{cpu}}(\mathrm{P}_{100\%} - \mathrm{P}_{\text{idle}}) + \Delta(T), \tag{3}$$
$$\text{where: } \Delta(T) = a_0 + a_1 T_{\text{inlet}} + a_2 T^2_{\text{inlet}}.$$

In this refined formula of Wang *et al.* [39], the inlet temperature is taken into account, and denoted by T_{inlet}. Temperature is implemented as a term added to the relative power used by the machine based on CPU usage multiplied by the possible increase in power usage under full load and idle status. Coefficients a_0, a_1, and a_2 are parameters fit through a process explained in the work of Wang *et al.* [39]. These variables are adjusted to minimize forecast error. In our experiments on our testbed, we obtained the values of 891,258 - 97.319 and 2.649 to minimize the error. Note that because our testbed was hosted within the IT department, we had very little control over the input temperature and had to do the parameter calculations with a smaller temperature range.

An important aspect of the analytical formula is its reliance on CPU utilization to predict power consumption post-deployment. However, during inference, this information is unavailable, necessitating placement decisions without it. To address this we proceeded with the utilization of the expected increase in consumption by averaging all occurrences in our dataset.

3.6.2 Extreme Gradient Boosting (XGBoost)

XGBoost iteratively combines weak regression models, refining predictions while preventing overfitting through advanced regularization techniques. Renowned for its scalability, speed, and accuracy, XGBoost is widely applied across various domains[7]. Its popularity stems from its efficiency, delivering excellent results with fast parallel training due to its ensemble structure. Like other methods such as Random Forest, XGBoost effectively avoids overfitting.

3.6.3 Support Vector Regressor (SVR)

Support Vector Regressor (SVR) operates by finding the hyperplane that best fits the data, minimizing the error within a margin. This model uses the kernel trick to handle non-linear relationships, resulting in flexibility and robustness in predictions. SVR is a variation of SVM[8] and is also utilized in fields requiring efficient regression models. The reason for using it is to check that simple methods do not provide the best solution.

3.6.4 Multi-Layer Perceptron (MLP) Regressor

The MLPRegressor belongs to the category of feed-forward artificial neural networks.

Our MLP model is a fully connected network composed of an input layer of size 43 relative to all features used, followed by three hidden layers with a topology of (20,100,4), and an output layer of size one responsible for generating power consumption predictions. This specific number of neurons per layer was chosen after some hyper parameter optimization process while also taking into consideration the computation cost of an oversized network. Figure 10 shows our topology with reduced numbers of nodes per layer to make it more readable. Our choice of parameters was done after a hyper-parameter optimization process gave us some improved values.

For weight optimization during MLP model training, we employ adam. As for the activation function, we opt for the logistic activation function, also known as the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{4}$$

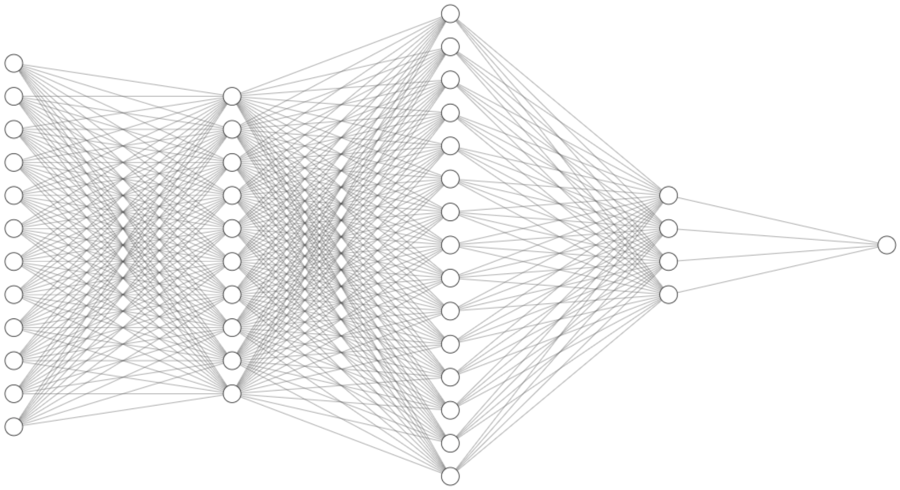


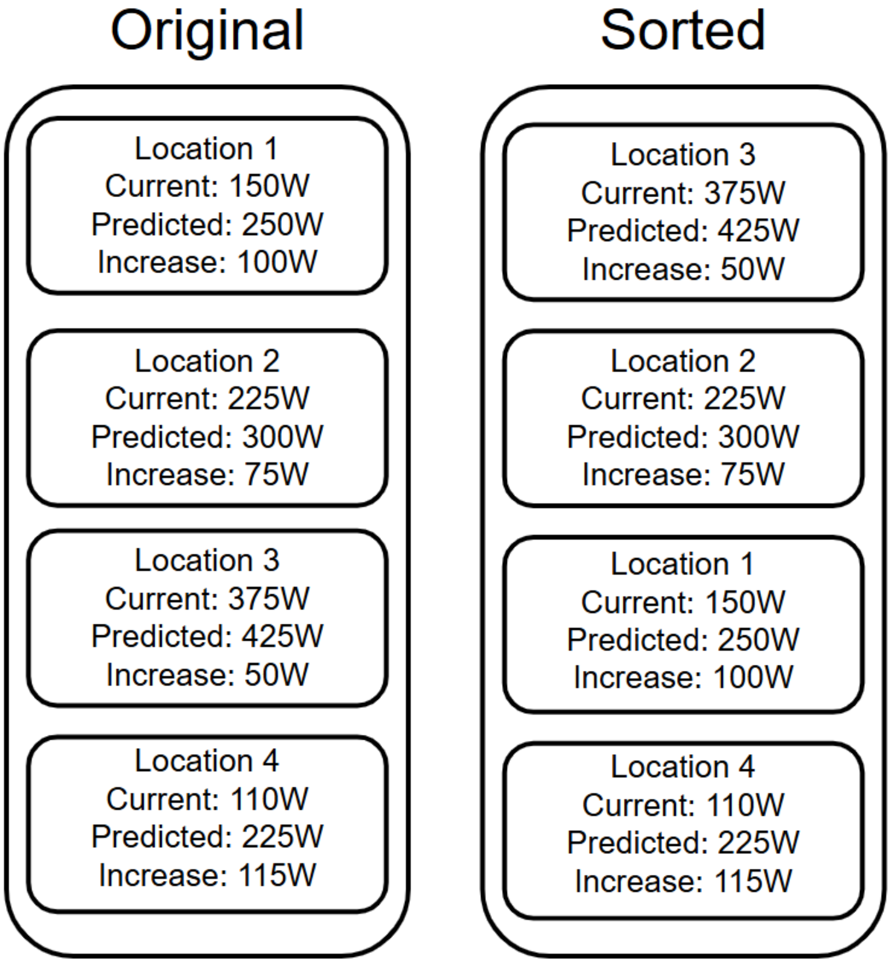
Figure 10: MLP topology

3.7 Containers on VMs/Bare metal placement algorithm

In this section, we go through our proposed algorithm for containers placement: it can be applied with only minor changes to both containers on VMs or containers bare metal. We first describe the machine learning model (Section 3.7.1), and then the consolidation algorithm that is used for performance evaluation and comparison (Section 3.7.2).

3.7.1 Placement with ML model

The core idea of our placement algorithm is to identify the PM efficiency discrepancies, as previously discussed and shown in Figure 4. The proposed energy-aware algorithm leverages a machine learning model to identify the optimal deployment location in terms of energy consumption. To do this, we estimate the energy impact of the incoming container, for all potential locations and compare their results. By sorting these results based on energy increase, we generate an ordered deployment list, as illustrated in Figure 11. The process of collecting the data can be optimized by programming the workers to constantly send to the controller their most up to date feature values, without the need for the controller to actively recover all this data. In the example proposed in Figure 11, if we assume a maximum power of 400 watts, then the best location would be location 2 since it is the location with the lowest increase that would be able to be provided by the power supply.



Original

Location 1
Current: 150W
Predicted: 250W
Increase: 100W

Location 2
Current: 225W
Predicted: 300W
Increase: 75W

Location 3
Current: 375W
Predicted: 425W
Increase: 50W

Location 4
Current: 110W
Predicted: 225W
Increase: 115W

Sorted

Location 3
Current: 375W
Predicted: 425W
Increase: 50W

Location 2
Current: 225W
Predicted: 300W
Increase: 75W

Location 1
Current: 150W
Predicted: 250W
Increase: 100W

Location 4
Current: 110W
Predicted: 225W
Increase: 115W

Figure 11: Example of preferences before and after sorted by power increase

In a context where there is a very high number of potential hosting PM/VM locations and the delay constraints are tight (e.g., in the order of few milliseconds in the edge cloud for the 5G fronthaul network), there might be a need to optimize this process. A solution could be to select only promising PMs, or, instead of making predictions only when containers come in, to make predictions periodically based on the information available at placement time, with the list sorted as much as possible to speed up the best available placement.

This ordered list offers two main benefits. Firstly, it allows us to select the deployment location with the lowest energy increase, ensuring efficient resource utilization. Secondly, it enables us to also detect the possibility of throttling. By comparing this predicted consumption with the machine's maximum power capacity, we can detect potential performance throttling due to insufficient power supply.

The proposed placement algorithm provides two significant advantages: throttle warnings and a priority list for deployment. This enables the creation of a straightforward algorithm for selecting the best deployment location, as detailed in algorithm 2. The first loop handles data collection and prediction for all machines. The second loop checks for cases where expected power consumption exceeds maximum capacity. In the rare cases where all options lead to machine overload, the first option in the priority list is selected.

Algorithm 2 Placement

```
procedure PLACEMENT($Cont, Max_p$)
    $Pred \leftarrow$ NewArray
    for $host$ in $hosts$ do
        $H_i \leftarrow$ Host Measurements
        $H_i.Append(Cont)$
        $Curr_i \leftarrow$ Host current Power
        $Pred_i \leftarrow model.Predict(H_i)$
        $Pred \leftarrow Pred.append((Pred_i, Curr_i, host))$
    $Pred \leftarrow Pred.SortBy(Pred_i - Curr_i)$
    $index \leftarrow 0$
    while $index < len(Pred)$ do
        if $Pred[index][predicted] < Max_p$ then
            return $Pred[Index]$
    return $Pred[0]$
```

3.7.2 Consolidation baseline

The primary difference between the ML-based approach and the consolidation method lies in how placements are determined. Instead of predicting and sorting a list to select placements, the consolidation based method creates a fixed list. This method prioritizes filling the first VM, and then moving to the next VM within the same PM until that PM is filled. Once a PM is filled, the process moves to the next PM. This strategy follows the first fit approach, which is to efficiently group containers onto as few machines as possible. Work of the literature has shown that best fit, first fit and worst fit have all their advantages and disadvantages and none of them is clearly outperforming the other ones [37], even with the energy concern [27].

3.8 Numerical results

We first present the results obtained from the training and testing of the energy consumption models detailed in Section 3.8.1. Next, we describe the experiment setup for the energy-aware placement experiments. Finally, in Section 3.8.3, we assess the energy savings achieved through the energy-aware placement of containers over VMs, along with the associated impacts on processing delays.

3.8.1 Energy prediction models

For the energy prediction system, the training data set was generated as explained in subsection 3.5.1. After conducting approximately 50 experiments, each spanning around 4 hours. For testing, we used RMSE (Root Mean Square Error) to quantify the error for each model. We also used the R^2 score as a secondary measure due to its flexibility. RMSE is defined as the square root of the mean of the squared prediction errors, providing a value that indicates the typical error magnitude, making it easier to understand the model's performance on our scale. R-squared R^2 is a coefficient of determination that measures the proportion of the variance in the dependent variable explained by the independent variables. This metric is generally helpful because R^2 does not require a specific range or scale to be meaningful;

its percentage value can be informative for any model. However, since we are analyzing wattage, RMSE remains very useful due to its ability to provide error magnitudes in the same units. Both metrics can be expressed as follows:

$$
\begin{aligned}
\text{RMSE} &= \sqrt{\frac{\left(\sum\limits_{i=1}^{N}(P_i - A_i)^2\right)}{N}} && (5) \\
R^2 &= \left(1 - \left(\frac{\sum\limits_{i=1}^{N}(A_i - P_i)^2}{\sum\limits_{i=1}^{N}(A_i - \bar{A})^2}\right)\right) \times 100 && (6)
\end{aligned}
$$

where N is the number of inferences done, A_i denotes the target, and P_i the predicted values for data point i, and $\bar{A}$ is the average of the real values. Table 4 shows the achieved values for each ML model, in this configuration, we experimented with the inclusion or removal of features to quantify the effects of increased data on the models explained in Section 3.6. This way it is easier to see the effect of using data regarding memory and IO utilization, as well as the effect of the sensors in the prediction model. The ML models are split into different categories as follows:

(i) Wang *et al.* [39] is the analytical formula that is being used as a comparison.

(ii) C-LR, C-MLP, C-SVR and C-XGB are the models that were trained using only CPU information.

(iii) CM-LR, CM-MLP, CM-SVR and CM-XGB are the models trained using CPU and Memory information.

(iv) CMI-LR, CMI-MLP, CMI-SVR and CMI-XGB are the models trained using CPU, Memory and IO information.

(v) CMIS-LR, CMIS-MLP, CMIS-SVR and CMIS-XGB are the models that were trained using all PMCs and sensors.

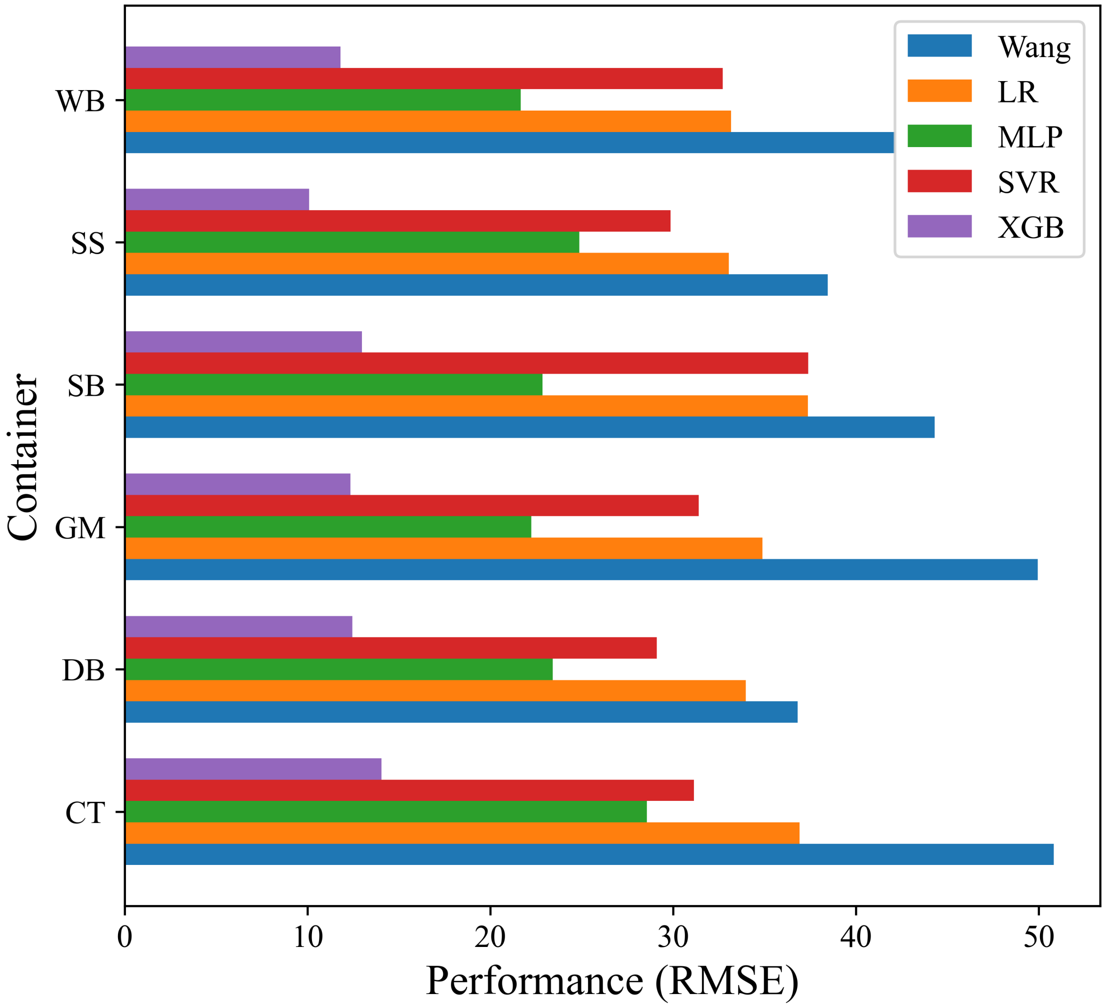


Figure 12: Model performance comparison

For the columns, each value represents a different load applied from all the services described in section 3 giving us a total of 6 services over all models.

From Table 4, it is evident that XGBoost achieved the highest accuracy across all categories with the greatest increase with the sensor data. The overall R^2 score was 91.2, with an RMSE of 12.8. Given our test bed, which includes realistic noise and real load, these values are quite impressive. The model consistently predicts energy consumption within 20 watts of the actual value, even in less favourable scenarios.

In contrast, the baseline model by Wang *et al.* [39] did not perform as expected in these

Table 4: Model Performance per load

Models		**CT**	**DB**	**GM**	**SB**	**SS**	**WS**
Wang *et al.* [39]		-8.4	18.7	-39.7	14.2	12.4	-10.1
C	LR	-11.6	-4.9	-20.3	-0.9	-9.8	-14.1
	MLP	21.8	27.5	15.5	38.3	43.5	42.6
	SVR	18.8	17.8	-9.8	-2.1	22.5	10.8
	XGB	86.4	88.4	92.9	93.5	90.6	89.6
CM	LR	-15.6	-6.1	-19.5	-3.1	-7.1	-17.7
	MLP	16.9	28.0	4.1	43.2	47.5	28.2
	SVR	20.5	21.4	-11.7	3.7	28.3	16.9
	XGB	86.4	88.2	90.5	89.3	91.7	87.3
CMI	LR	-14.5	3.5	-6.9	19.2	14.5	6.9
	MLP	39.4	66.7	71.1	77.7	66.4	73.3
	SVR	26.9	42.2	24.2	25.7	46.3	28.6
	XGB	81.1	87.7	90.6	90.4	89.9	89.1
CMIS	LR	25.9	7.7	21.5	33.6	34.8	32.7
	MLP	71.6	72.3	72.5	78.0	72.9	79.1
	SVR	46.3	36.1	34.8	31.0	50.6	34.1
	XGB	87.9	89.1	92.1	91.9	94.2	91.7

scenarios, prompting us to compare it against a classical linear regression model. This comparison clearly shows that a simple linear regression model fails to accurately predict the energy consumption of these hosts when the load is not easily predictable. Furthermore, there is a significant discrepancy in predicting different loads when using overly simplistic models like linear regression.

The values shown in Figure 12 represent the RMSE of wattage prediction after deploying a given container for the CMIS version of each model. This highlights the differences in task performance by the same model. We can see that more complex models can make accurate predictions for various loads, albeit with slight differences. In contrast, this discrepancy increases when using Linear Regression and the analytical formula, indicating a non-linear factor in power utilization.

Additional analyses were conducted based on CPU utilization before deployment. These were performed because, as utilization increases and power reaches its maximum capacity, the range of possible prediction values is reduced, and model accuracy is expected to improve. This trend is evident in Figure 13, where all models show a continuous reduction in error as CPU utilization increases.

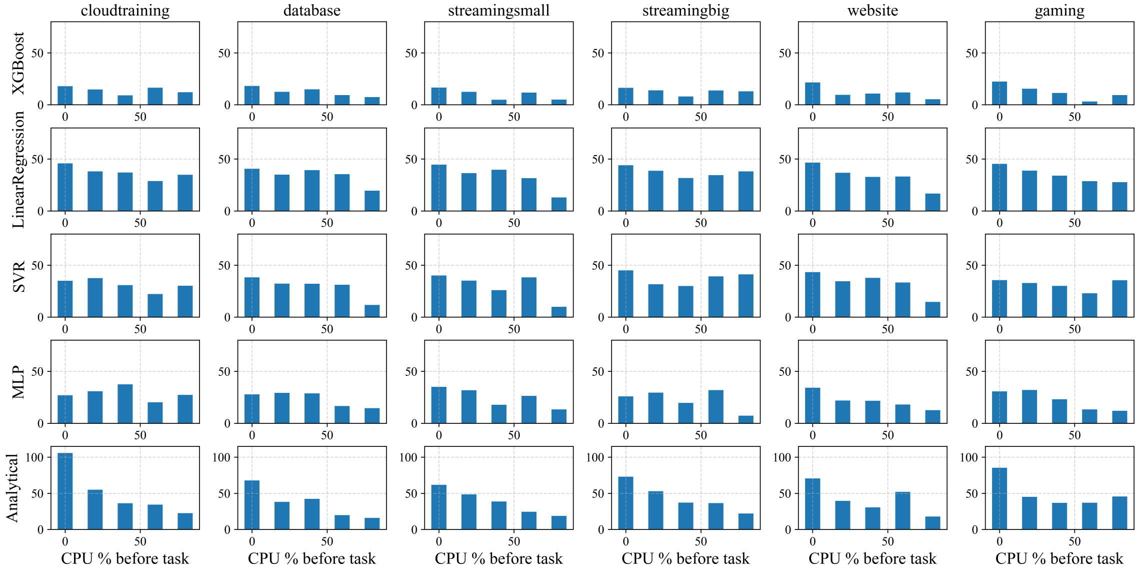


Figure 13: Model performance per percentage of utilization

3.8.2 Energy aware experiment setup

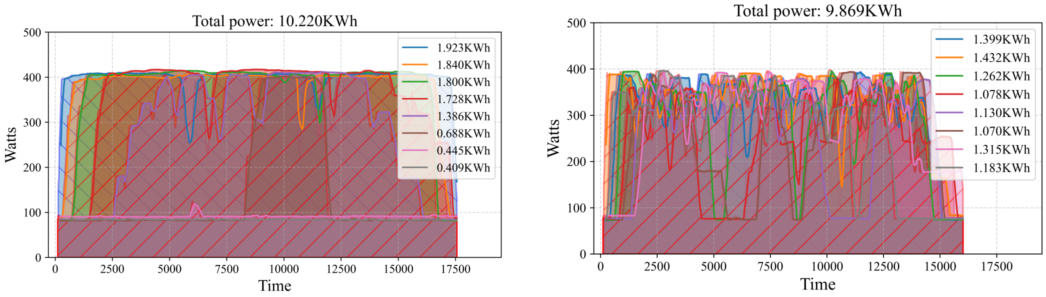
To evaluate the energy savings of the energy-aware placement algorithm, we used the modified consolidation method described in Section 3.3.3 as our baseline. This method is commonly chosen for its simplicity and effectiveness in reducing energy consumption. In addition to the baseline, we implemented our approach using the XGBoost model, which demonstrated greater accuracy as shown in Section 3.8.1.

For this experiment, there is an initial process of generating the list of containers to be deployed (identical process to the one explained in Subsection 3.5.1). When the experiment progresses to the second phase, where PSUtil and IPMI start logging data, this data is sent directly from each VM to the controller in real-time. From this point, instead of randomly placing containers on VMs, the algorithm follows its implementation to optimize container placement for energy reduction.

The ML-assisted algorithm operates as follows: When the time for deploying a container arrives, the algorithm retrieves the latest data from all potential host VMs. It then iterates over this data, one VM at a time, using the ML model to predict the power consumption after deploying the new container. This process generates a list of energy variations for each VM, which is ordered from the smallest to the largest increase in power consumption. The placement algorithm uses this list, along with Kubernetes metadata to avoid resource overload, to determine the optimal placement for the container.

For the Consolidation algorithm, the process differs significantly. Instead of iterating over the data, this algorithm generates a fixed list that prioritizes deployment on a predetermined sequence of machines. This approach effectively consolidates the load onto the initial set of machines whenever possible.

3.8.3 Benefits of energy aware placement of containers over VMs



(a) Consolidation strategy

(b) Smart placement

Figure 14: Energy consumption during trial

For energy savings, we conducted two experiments to evaluate the performance of the method of placement : *(i)* A classical consolidation technique; *(ii)* ML-assisted Placement.

In Figure 14a, we observe the behaviour of the experiment under the consolidation policy. The most notable difference is the presence of hosts that are barely used, as indicated by areas remaining at idle levels. In contrast, Figure 14b shows that our algorithm utilizes different hosts to manage and reduce power consumption, waiting for temperature reductions and the completion of other containers.

A significant drawback of the consolidation approach is the potential for performance reduction due to job stacking on the same hosts, which is evident in the longer completion time for the experiment.

For comparison, two metrics must be considered: energy reduction and QoS maintainability. The energy used, as a key performance indicator (KPI), shows a consistent reduction of 2.5% to 3% with our method. When accounting for the energy required to cool the data centers, these savings effectively double, since cooling energy is proportional to the data center's energy utilization. This reduction holds immense potential for data centers, where even minor power savings can lead to substantial cost reductions and decreased CO2 emissions, potentially saving millions of dollars over time. Extrapolating these results, a small data center with 1,000 Physical Machines could save up to 87.75 KWh in overall usage. Savings would be even more significant in large data centers, which typically consume hundreds of MWh.

In addition to power consumption reduction, our method also reduces processing time, positively impacting QoS. While this improvement is difficult to quantify with a specific KPI, maintaining a fixed QoS is critical for service providers to avoid contractual fines due to non-compliance.

3.9 Conclusion and future work

We proposed an accurate method for container placement on VMs aimed at reducing power consumption through the use of machine learning models, which further achieved significant energy reductions.. Throughout our experiments, we demonstrated that our best model, XGBoost, attained high accuracy across diverse types of traffic loads and machine states. Indeed, the model achieved an R^2 score of 91.2% and an RMSE of 12.8. It was thoroughly tested on various machines, proving its ability to generalize and learn the underlying patterns. Additionally, the model maintained performance across different types of loads with varying resource utilization.

By leveraging our XGBoost model, we efficiently deployed containers, thereby reducing energy consumption. We conducted tests using realistic scenarios and mimicking real-world loads, created from practical operations rather than artificial stressors, ensuring a more accurate representation of real-life conditions. Additionally, we implemented the model on various classes of VMs to enhance the robustness and validation of our results. Our method achieved a consistent reduction in energy consumption.

Moreover, this approach improved task processing times by better allocating tasks and avoiding already overloaded machines. This not only increased performance and QoS but also maintained energy savings. Consequently, our implementation can be utilized for the placement of containerized tasks to minimize energy consumption in data centers' virtual or Physical Machines, using readily available features in modern data centers.

Building on the results of our proposed method for container placement on VMs, future research should focus on integrating real-time monitoring and adaptive learning into the XGBoost model for dynamic adjustments based on live data. This would optimize resource utilization and energy savings and could integrate well into migrations algorithms, another effective way to reduce energy consumption dynamically. Additionally, exploring advanced machine learning techniques, such as deep learning, could enhance prediction accuracy and adaptability. These models would better handle workload and hardware variability, leading to more efficient placement decisions. By pursuing these areas, research can develop more sophisticated and sustainable solutions for container placement in virtualized and cloud environments.

Chapter 4

Conclusion & Future Work

4.1 Conclusions

We propose an effective method for placing containers on VMs to reduce power consumption using machine learning models and algorithms. Our approach had two main objectives: developing an accurate model and reducing energy usage. We successfully achieved both, resulting in a model that provides above-average energy savings. Our experiments showed that, compared to traditional analytical solutions, our method offers superior performance in data center environments.

Our proposed models addressed these shortcomings by fully leveraging the feature space with sensors and performance monitoring counters (PMCs), creating a non-linear solution capable of learning system complexity. Indeed, our models significantly improved energy prediction for future load assignments. Our best model achieved an R^2 score of 91.2% and an RMSE of 12.8. This improved accuracy resulted in an energy saving of 3%, potentially doubling with the addition of cooling.

Furthermore, our research demonstrated several key findings that contribute to the realism and effectiveness of our approach: (i) By avoiding artificial stressors, we ensured that our results apply to real-world scenarios in terms of the use of computational resources.

(ii) Conducting comprehensive testing in a real data center validated the practical viability of our model. (iii) Implementing a non-consolidation method highlighted the benefits of dynamic resource allocation. Showing that consolidation methods are not the only way to reach an energy efficient VM/container placement (iv) Utilizing varied VM hosts and loads confirmed the robustness and adaptability of our approach across different conditions.

Our placement method consistently reduced energy consumption by a small but stable amount. Additionally, our approach enhanced task processing times by better-allocating tasks and avoiding already overloaded machines, thereby increasing performance and QoS while maintaining energy savings. When extrapolated to larger and more powerful data centers, our method has the potential to achieve annual savings of millions of dollars.

4.2 Future work

This study can propel future work for further improvements in multiple directions;

- Focusing on traffic prediction could significantly enhance the scheduling aspect of the environment. By accurately predicting future loads, we can better allocate and distribute resources, leading to more efficient load management and improved performance.

- Many of the works that inspired us tackled current energy prediction in each instance of containers or VMs, and, since our application required future energy consumption, we chose to simplify the problem and skip over this part, changing prediction only to the increase in energy of PM. Changing this into the original task and utilizing the "real-time" energy prediction would allow us to manage the cluster on a finer level, and enable migrations and many more works.

- The work can be extended to a more robust and complete framework of virtual system management for a more complete energy savings system. If proper changes are done, another work could tackle scheduling, migration as well as placement. The absence of dynamic migrations to optimize the cluster in case of unexpected changes could lead to untapped potential.

- The usage of a reinforcement learning method can be a great solution to this problem. The online states and reward possible setup from the energy consumption point of view seem to be a logical application to this kind of technology. The cost of training a reinforced learning model could potentially be high, however, if the work proposed in item I is achieved, it could simplify the environment setup for this kind of solution drastically since all the RL needs would be easily integrated.

Bibliography

[1] K. Ahmed, M. Alvarez, and M. Bollen. Characterizing failure and repair time of servers in a hyper-scale data center. In *IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, pages 660–664, The Hague, Netherlands, 2020.

[2] A. Al-Moalmi, J. Luo, A. Salah, K. Li, and L. Yin. A whale optimization system for energy-efficient container placement in data centers. *Expert Systems with Applications*, 164:113719, 2021.

[3] J. Baumgartner, C. Lillo, and S. Rumley. Performance losses with virtualization: Comparing bare metal to vms and containers. In A. Bienz, M. Weiland, M. Baboulin, and C. Kruse, editors, *High Performance Computing*, pages 107–120, Cham, 2023. Springer Nature Switzerland.

[4] A. Beloglazov and R. Buyya. OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds. *Concurrency and Computation: Practice and Experience*, 27(5):1310–1333, 2015.

[5] A. Bohra and V. Chaudhary. VMeter: Power modelling for virtualized clouds. In *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8, 2010.

[6] L. Caviglione, M. Gaggero, M. Paolucci, and R. Ronco. Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. *Soft Computing*, 25(19):12569–12588, 2021.

[7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[9] A. H. T. Dias, L. H. A. Correia, and N. Malheiros. A systematic literature review on virtual machine consolidation. *ACM Comput. Surv.*, 54(8), oct 2021.

[10] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH International Symposium on Computer Architecture (ISCA)*, 35(2):13–23, 2007.

[11] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen. Energy-aware VM consolidation in cloud data centers using utilization prediction model. *IEEE Transactions on Cloud Computing*, 7(2):524–536, 2019.

[12] S. Giallorenzo, J. Mauro, M. G. Poulsen, and F. Siroky. Virtualization costs: benchmarking containers and virtual machines against bare-metal. *SN Computer Science*, 2(5):404, 2021.

[13] S. Gill, S. Tuli, A. Toosi, F. Cuadrado, P. Garraghan, R. Bahsoon, H. Lutfiyya, R. Sakellariou, O. Rana, S. Dustdar, and R. Buyya. Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments. *Journal of Systems and Software*, 166(110596):234 – 248, April-June 2020.

[14] K. Haghshenas and S. Mohammadi. Prediction-based underutilized and destination host selection approaches for energy-efficient dynamic vm consolidation in data centers. *The Journal of Supercomputing*, pages 1–18, 2020.

[15] L. Helali and M. N. Omri. A survey of data center consolidation in cloud computing systems. *Computer Science Review*, 39:100366, 2021.

[16] A. Hylick, R. Sohan, A. Rice, and B. Jones. An analysis of hard drive energy consumption. In *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pages 1–10, 2008.

[17] S. Ilager and R. Buyya. Energy and thermal-aware resource management of cloud data centres: A taxonomy and future directions. *ArXiv*, abs/2107.02342, 2021.

[18] C. Jin, X. Bai, C. Yang, W. Mao, and X. Xu. A review of power consumption models of servers in data centers. *Applied Energy*, 265:114806, 2020.

[19] K. Kaur, S. Garg, G. Kaddoum, F. Gagnon, and D. N. K. Jayakody. Enlob: Energy and load balancing-driven container placement strategy for data centers. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2019.

[20] M. A. Khan, A. Paplinski, A. M. Khan, M. Murshed, and R. Buyya. *Dynamic Virtual Machine Consolidation Algorithms for Energy-Efficient Cloud Resource Management: A Review*, pages 135–165. Springer International Publishing, Cham, 2018.

[21] C. Kominos, N. Seyvet, and K. Vandikas. Bare-metal, virtual machines and containers in OpenStack. In *Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 36–43, 2017.

[22] N. Li, X. Xu, Q. Sun, J. Wu, Q. Zhang, G. Chi, C.-L. , and N. Sprecher. Transforming the 5G RAN with innovation: The confluence of cloud native and intelligence. *IEEE Access*, 11:4443–4454, 2023.

[23] B. Liang, X. Dong, Y. Wang, and X. Zhang. Memory-aware resource management algorithm for low-energy cloud data centers. *Future Generation Computer Systems*, 113:329–342, 2020.

[24] S. Madan and D. Antoniadis. Leakage current mechanisms in hydrogen-passivated fine-grain polycrystalline silicon on insulator mosfet's. *IEEE Transactions on Electron Devices*, 33(10):1518–1528, 1986.

[25] A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree. An empirical survey of performance and energy efficiency variation on intel processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, E2SC'17, New York, NY, USA, 2017. Association for Computing Machinery.

[26] N. Moocheet, B. Jaumard, P. Thibault, and L. Eleftheriadis. A sensor predictive model for power consumption using machine learning. In *IEEE International Conference on Cloud Networking (CLOUDNET)*, pages 238 – 246, Hoboken, NJ, USA, November 2023.

[27] T. K. Okada, A. De La Fuente Vigliotti, D. M. Batista, and A. Goldman vel Lejbman. Consolidation of VMs to improve energy efficiency in cloud computing environments. In *Brazilian Symposium on Computer Networks and Distributed Systems*, pages 150–158, 2015.

[28] Oracle. Installing and configuring openstack (kilo) in oracle® solaris, three-node architecture overview. `https://docs.oracle.com/cd/E65465_01/html/E61044/archover.html`.

[29] F. Quesnel, H. Mehta, and J.-M. Menaud. Estimating the power consumption of an idle virtual machine. In *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 268–275, 2013.

[30] M. Ranjbari and J. Torkestani. A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. *Journal of Parallel and Distributed Computing*, 113:55–62, 2018.

[31] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan. Security of os-level virtualization technologies: Technical report, 2014.

[32] M. Rezaei-Mayahi, M. Rezazad, and H. Sarbazi-Azad. Temperature-aware power consumption modeling in hyperscale cloud data centers. *Future Generation Computer Systems*, 94:130–139, 2019.

[33] G. Rodola. Psutil documentation, 2020.

[34] L. Stress. stress(1) - linux man page. `https://linux.die.net/man/1/stress`.

[35] S. Sultan, I. Ahmad, and T. Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996, 2019.

[36] D. Thangam et al. Impact of data centers on power consumption, climate change, and sustainability. In K. Kumar, V. Varadarajan, N. Nasser, and R. Poluru, editors, *Computational Intelligence for Green Cloud Computing and Digitial Waste Management*, chapter 4, pages 60 – 83. IGI Global Publishers, USA, March 2024.

[37] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos. A survey on data center networking for cloud computing. *Computer Networks*, 91:528–547, 2015.

[38] Y. Wang, D. Nörtershäuser, S. Le Masson, J.-M. Menaud, et al. An empirical study of power characterization approaches for servers. In *International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 1–6, 2019.

[39] Y. Wang, D. Nörtershäuser, S. Le Masson, J.-M. Menaud, et al. An empirical study of power characterization approaches for servers. In *International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 1–6, 2019.

[40] C. Zaloumis. Are your data centers keeping you from sustainability?, Feb 2024.

[41] X. Zhang, Z. Shen, B. Xia, Z. Liu, and Y. Li. Estimating power consumption of containers and virtual machines in data centers. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 288–293, 2020.