Is the *powersave* governor really saving power?

Darong Huang*, Luis Costero† and David Atienza*
*Embedded Systems Laboratory (ESL), EPFL, Switzerland, †Dpto. of Computer Architecture and Automatics, UCM, Spain
Email: *{darong.huang, david.atienza}@epfl.ch, †lcostero@ucm.es

***Abstract*—A frequency scaling governor is critical for the performance management of cloud servers, as it enhances energy efficiency and helps to control operational temperatures, thereby ensuring system reliability. However, our in-depth analysis of the application's performance and Dynamic Voltage and Frequency Scaling (DVFS) actions, alongside assessments of server power consumption and operating temperature, indicates that existing Linux scaling governors often fall into non-optimal DVFS strategies, especially for cloud applications with varying workloads and requests. This shortfall comes from the misleading CPU load metrics, which fail to accurately capture the applications' true performance requirements and demands. In this context, we introduce a novel scaling governor named GreenDVFS. First, it identifies the optimal frequencies for the application in a range of workload scenarios. Optimal frequencies are used to maintain application performance, reduce server power consumption, and maintain a balanced operating temperature in different workload scenarios. Furthermore, we design a long short-term memory (LSTM)-based time series methodology to detect the real-time workloads of cloud applications accurately and timely. Building on these foundations, the proposed method takes optimal DVFS actions, tailored for cloud applications under different workload conditions, to optimize performance, energy efficiency, and temperature. The experimental results highlight the effectiveness of the proposed GreenDVFS, with up to 18% savings in energy consumption and a 30% decrease in operational temperature by comparing against the default Linux governor, all while not compromising the application's performance. Such improvements help to optimize cloud computing operations for enhanced efficiency and sustainability.**

I. Introduction

Data centers have experienced remarkable growth in recent decades, thanks to their substantial advantages in security, flexibility, and cost-effectiveness. Thus fueling the migration of more end-users and applications to the cloud. With such a trend, the global expenditure on cloud services is projected to increase by more than 20% in 2023, reaching approximately $600 billion [1]. Data center energy consumption is skyrocketing along with its rapid growth. A recent study [2] estimates that by 2030, data centers will contribute to approximately 3 to 13% of the total electricity consumption of the world. Consequently, improving the energy efficiency of data centers has been a critical concern for both industry and academia.

An approximate power breakdown of a computing server shows that multi-core processors consume more than 56% of the total energy [3] of the whole system. Therefore, one effective means of controlling server energy consumption is to reduce the microprocessor's power consumption. It is well known that CPU power consumption increases at a cubic rate or higher when boosting server performance by increasing the operation frequency [4]. This is why the CPU frequency scaling has been integrated into the modern operating system, enabling the automatic adjustment of the CPU frequency in response to system load, thereby conserving power and enhancing performance.

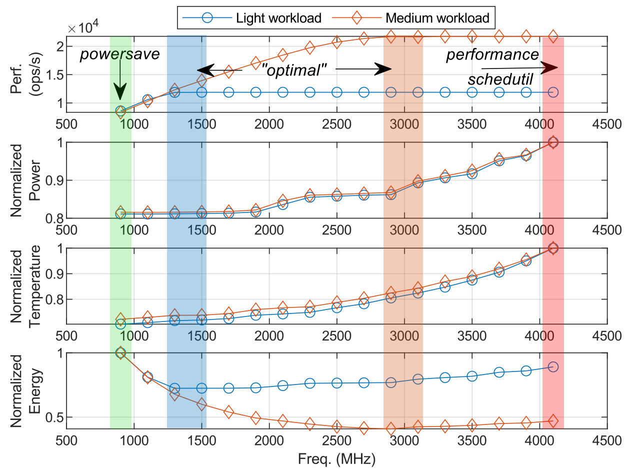


Fig. 1: Profiling results for the Data Serving benchmark under two different workload levels across different DVFS points.

CPU frequency scaling is achieved through various strategies, commonly known as scaling governors in Linux [5]. For instance, the *powersave* governor enforces to run the CPU at the minimum available frequency. The *performance* governor, on the contrary, maintains a constant maximum frequency. Additionally, there are governors that can dynamically adjust the frequency based on system load. One such governor is *schedutil*. It uses the per-entity load tracking technique [6] to estimate CPU utilization and compute the optimal CPU frequency to apply in accordance with this metric.

Despite the availability of these governors, we have observed that emerging cloud applications often do not run optimally on the servers in terms of energy efficiency. To show this fact, we perform a profiling of the Data Serving benchmark from the CloudSuite benchmark suite [7], a benchmark that loads one of the most popular NoSQL databases, Cassandra, using Yahoo! Cloud Serving Benchmark (YCSB) to simulate a representative NoSQL database application in the cloud.

We begin by profiling the benchmark under a medium workload, covering all available frequencies by manually fixing the DVFS level of the server. Subsequently, we present both application and server statistics, namely performance, normalized power, temperature, and energy consumption, across various frequency levels in Fig. 1 as the orange line with diamond markers. Initially, the performance increases with frequency level, but it reaches a plateau at around 3000MHz. However, the power and temperature continue to rise with the frequency increase as the application utilizes the server's full capacity. Consequently, we can conclude that one *"optimal"* scaling governor would maintain the frequency at 3000MHz to minimize power consumption without sacrificing performance. Then, we evaluate the scaling governors available in the Linux kernel, marking their runtime CPU frequency selections in the figure. The *powersave* and *performance* governors operate at the minimum and maximum CPU frequencies, as expected. The default Linux governor, *schedutil*, also runs the application at the maximum frequency as the *performance* governor. In conclusion, none of the Linux kernel's scaling governors are optimal for this scenario to minimize power consumption without sacrificing performance. This is highlighted by the energy consumption analysis in the figure's final subplot, showing that existing governors fail to achieve minimum energy consumption. In particular, the *powersave* governor, despite its name, is the most energy-intensive option. Furthermore, as the workload level decreases to a light level, depicted by the blue line with circle markers in Fig. 1, the optimal frequency of the application shifts to a lower range, approximately 1500MHz, to meet its decreasing demands. However, the default Linux governor, *schedutil*, consistently sets the maximum CPU frequency, regardless of fluctuations in the runtime workload. This leads us to conclude that the current scaling governors fall short in selecting the optimal operating frequency for cloud applications and lack sensitivity to the dynamic workload variations inherent in these applications.

Therefore, in this work, we focus on developing an *optimal* governor, called GreenDVFS. It is capable of detecting various levels of workload experienced by the application and subsequently setting the optimal frequency for its operation. The primary contributions of this research encompass:

- The development of an automatic optimization methodology to choose the ideal operating frequency corresponding to the workload level of the application. Thus, optimizing the energy efficiency of the server without compromising its performance.
- The proposal of a time-series workload detection method for earlier detection of the runtime workload level for the application.
- A comprehensive analysis of improvements achieved by the proposed workload-driven scaling governor, GreenDVFS, in comparison to the state-of-the-art Linux scaling governors. The results demonstrate that GreenDVFS achieves up to an 18% saving in energy consumption and a 30% decrease in operational temperature by comparing against the default Linux governor, all while not compromising the application's performance.

The rest of this paper is organized as follows. Section II discusses the related work and background of this study. Section III describes the general design of the workload-driven scaling governor. The experimental setup and results are presented in Sections IV and V, respectively. Finally, Section VI concludes this study.

II. Related Work And Background

Frequency scaling techniques have played a crucial role in contemporary computing systems, ranging from mobile devices [8] to high-performance cloud servers [9]. Given that modern microprocessors contribute substantially to overall system power consumption [3], scaling governor stands out as an effective means to control microprocessor power consumption and improve its performance. This motivates us to investigate these frequency scaling techniques, both in cutting-edge industry practices and in academic research.

A. State-Of-The-Art Frequency Scaling Techniques

Linux scaling governors primarily rely on CPU utilization as a key metric to determine the appropriate frequency level. In the case of the *ondemand* governor [10], it calculates the CPU load by assessing the portion of time during which the CPU remains active rather than idle, thereby establishing the ratio of active time to total CPU time as an estimation of the load. With the introduction of the per-entity load tracking (PELT) mechanism in Linux 3.8 [11], a new approach to estimating CPU utilization emerged, enhancing the scheduler design and the efficiency of scaling governors. Consequently, the *schedutil* governor [5] strives for a more seamless integration with the Linux kernel scheduler, deriving its load estimation through the PELT mechanism.

However, the CPU utilization metric can be inaccurate and misleading, as demonstrated in [12]. Consequently, Linux governors can fall short in setting optimal frequency for cloud applications to optimize energy efficiency, as highlighted in the case study of the Introduction. To address this issue, researchers have proposed DVFS techniques that leverage metrics such as cycles per instruction and various hardware events to govern CPU frequency [12]–[14].

For high-performance computing, Ali et al. introduced an automated method that can bring the temperature of an overheated CPU back within the normal range by scaling down its CPU frequency [9]. Akram et al. offered a performance prediction method for multithreaded applications and optimized their energy efficiency while keeping performance within user-defined thresholds [15]. Additionally, Liu et al. introduced power capping techniques based on CPU and memory DVFS to improve system energy efficiency [16].

For specific applications such as transactional database systems in the cloud, Korkmaz et al. proposed a workload-aware CPU frequency scaling method [17]. On a different note, Liu et al. proposed an innovative approach by embedding DVFS control within a Java virtual machine, allowing the application to take the frequency for itself [18]. However, this requires deep reengineering within the application itself, which is not feasible in most cases.

With the emerging of machine learning (ML) techniques, several ML-based DVFS schemes have been proposed in

[19]–[21]. Despite these diverse approaches trying to address challenges on frequency scaling of cloud servers and applications, it remains challenging to optimize the server to its best energy efficiency state without comprising performance. This difficulty largely stems from inadequate knowledge of cloud application workloads. Consequently, this situation underscores the importance of workload detection in the cloud.

B. Workload Detection

Numerous works have focused on workload detection, proposing different prediction approaches. For example, a linear regression model is proposed in [22] to estimate the incoming workload and then scale the cloud configuration to adapt to the requirements of future workloads. In addition to regression-based methods, random forest-based workload detection methods are presented in [23], [24]. In [25], a support vector machine-based workload detection method is implemented to predict the workloads of the cloud server.

To improve prediction accuracy, more recent efforts have focused on machine learning (ML)-based approaches. For example, a cluster-based workload detection method is proposed in [26]. This method initially clusters tasks into several categories before predicting CPU and memory usage for task scheduling. Another contribution is the use of the long-short-term-memory (LSTM) technique for workload detection, as explored in [27]. Despite these advances, the dynamic nature of client requests presents a tremendous challenge in predicting cloud server workloads. According to [27], the lowest error rate in predictions remains at 18%. This highlights the need for an effective, application-focused, workload detection method in real-time cloud environments.

In summary, to the best of our knowledge, cloud applications continue to grapple with the low accuracy workload detection methods and the non-optimal DVFS actions from state-of-the-art frequency scaling governors. Consequently, in this study, our motivation is to introduce a novel workload-driven DVFS technique, named GreenDVFS, to enhance the energy efficiency of cloud servers while maintaining optimal application performance.

III. GreenDVFS: Workload-Driven Dynamic Voltage Frequency Scaling

The proposed GreenDVFS workflow comprises three stages, as depicted in Fig. 2. In the initial stage, we perform profiling of cloud applications running on the server to identify the optimal workload-frequency pairs for a selected set of workload levels from its continuous workload space. This phase involves the profiling of application performance data, server's power consumption, and temperature under varying CPU frequencies. By integrating this data with the proposed optimizer, we can determine the optimal runtime frequency based on the proposed cost function, thus establishing the "best workload level-frequency pairs" for a set representative workload levels of the application.

The second stage involves training a neural network for workload detection. We propose to use the server metrics,

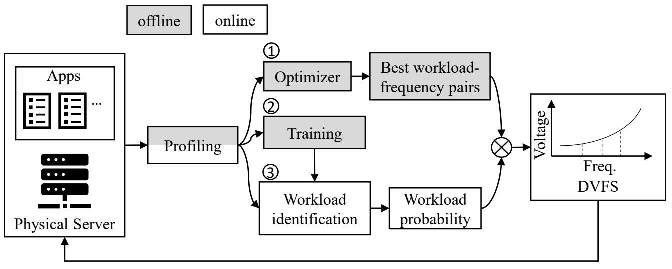


Fig. 2: The workflow of proposed GreenDVFS methodology.

which include data from hardware performance monitoring units (PMUs), network statistics, and memory usage information. Then, we employ a long short-term memory (LSTM) network for accurate time-series workload detection.

Upon completion of the two offline stages, we transition to the online workload-driven DVFS phase. At runtime, we can utilize the monitored server metrics and trained LSTM to predict the current workload status and generate a probability distribution for the current workload level. Combining this information with the knowledge of the best workload-frequency pairs obtained in the initial stage, our method performs DVFS for the server, aiming to achieve optimal energy efficiency without sacrificing application performance.

We discuss the GreenDVFS in detail below.

A. Optimizer for The Best Workload-Frequency Pairs

As demonstrated in the Introduction, there is an ideal operational frequency point for the application at a given workload level, a point that state-of-the-art solutions struggle to locate. Hence, we propose an optimizer to identify this optimal frequency for a determined workload wkl_i as follows:

$$\begin{aligned} \max_{f} \quad & L(f) = \alpha \cdot \mathit{Perf}(f) - \beta \cdot \mathit{Power}(f) - \gamma \cdot \mathit{Temp}(f) \\ & \alpha, \beta, \gamma \in [0,1], \quad \alpha + \beta + \gamma = 1 \end{aligned} \tag{1}$$

where Perf, Power, and Temp represent the performance of the application, the power consumption, and the operating temperature, respectively, at a specific workload wkl_i. These metrics are dynamic variables influenced by changes in CPU frequency at runtime f. In addition, we introduce three coefficient factors, denoted as α, β, and γ. Thanks to these coefficient factors, we can adjust the impact of different aspects, such as performance, power consumption, and temperature, on the loss function $L(f)$. So, given the profiling results for the application under a particular workload level, wkl_i, the optimal frequency f_{wkl_i} can be derived as $\max L(f)|_{wkl_i} \rightarrow f_{wkl_i}$.

Subsequently, for a selected set of workload levels, we can formulate the optimal frequency pairs f_{wkls} as follows:

$$\begin{bmatrix} \max & L(f)|_{wkl_1} \\ \max & L(f)|_{wkl_2} \\ & \cdots \\ \max & L(f)|_{wkl_N} \end{bmatrix} \rightarrow \begin{bmatrix} f_{wkl_1} \\ f_{wkl_2} \\ \cdots \\ f_{wkl_N} \end{bmatrix} \tag{2}$$

Please note that the application's workload space is continuous and covers a broad range, while the CPU frequency

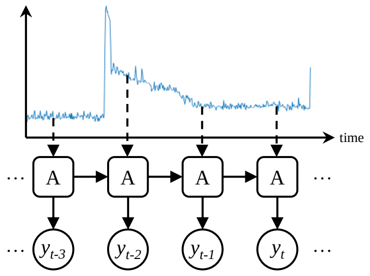


Fig. 3: The workflow of time-series workload detection. The neural network, denoted as "A", takes the sampled metrics and its own hidden state from the previous time step as the input and outputs y_t, representing the probabilities of the application operating under different workloads.

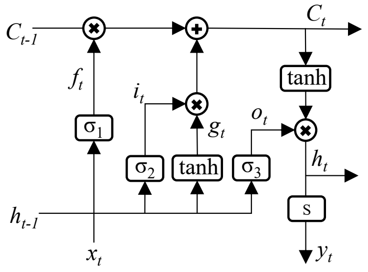


Fig. 4: The diagram of the LSTM cell used in this study.

space is discrete. As a result, one only needs to select a finite number of workloads, denoted N in the above equation, for each application, covering all of the frequency spectrum.

B. Workload Detection with LSTM

The absence of real-time workload and performance data for the frequency scaling governor leads to non-optimal operational CPU frequencies during runtime [12]. This can result in higher power consumption and increased server temperatures, as demonstrated in the Introduction. To tackle this challenge, we propose a machine learning-based method for workload detection in cloud applications without necessitating any modifications to the system or applications. The illustration of this approach is shown in Fig. 3.

In the figure, the application operates on the server under a specific workload. System profiling allows us to acquire real-time metrics from the system and application, like instructions per second (IPS), network packages, etc. To illustrate our proposal more easily, the blue trace in the figure represents one metric of all metrics sampled over time. Initially, we use the sampled metric data as input for a neural network (denoted "A" in the figure) to estimate the probabilities of the application operating under different workloads, denoted y_{t-3}. As time progresses, the neural network continuously incorporates new hardware metrics as input, also utilizing its previous state to make updated predictions.

To achieve this, we propose to use the long short-term memory (LSTM) neural network [28] to detect the workload of time series. The structure of the LSTM cell used in this work is presented in Fig. 4. At a given time t, the LSTM cell receives the measured metrics x_t. Since LSTMs are designed to handle time series data with their recurrent structure, these inputs also encompass cell memory C_{t-1} and the hidden state h_{t-1} of the preceding LSTM cell. The processing is defined as follows:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) & C_t &= f_t \cdot C_{t-1} + i_t \cdot g_t \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) & o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ g_t &= \tanh(W_g \cdot [h_{t-1}, x_t] + b_g) & h_t &= o_t \cdot \tanh(C_t) \end{aligned} \tag{3}$$

where σ and $\tanh$ denote the activation functions of the sigmoid and hyperbolic tangent for the layers. The symbols W and b represent the weights and biases associated with these layers. Specifically, the layer σ_1 is referred to as the "forget gate" because its output, f_t, enables the LSTM to keep or erase information from C_{t-1}. On the other hand, the layers σ_2 and $\tanh$, along with their outputs i_t and g_t decide the amount of new information added to C_{t-1} to form the new memory C_t. Finally, the hidden state h_t is produced, incorporating the information from the layer σ_3 and the updated memory C_t.

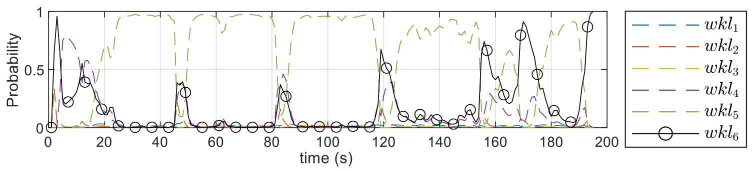
Then, the output layer takes the current hidden state h_t as input and outputs the probability distribution of different workloads as y_t by applying a *softmax* activation function, as specified in Eq. 4.

$$y_t = softmax(W_s \cdot h_t + b_s) \tag{4}$$

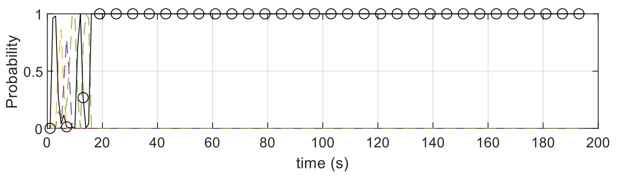
where $y_t \in \mathbb{R}^N$ represents a vector containing the probabilities assigned to various workload levels at time t. More precisely, we can express it as $y_t = [P_{wkl_1}, P_{wkl_2}, \cdots, P_{wkl_N}]$, where P_{wkl_i} denotes the probability associated with the current workload level labeled as wkl_i. As y_t represents a probability distribution, it satisfies $\sum_{i=1}^{N} P_{wkl_i} = 1$. By employing LSTM, we are able to perform time-series workload detection. However, a limitation in the current LSTM design lies in its training scheme, which primarily emphasizes accuracy only at the end of the trace, neglecting earlier stage accuracy. Indeed, if the LSTM only provides accurate results in the final phase of the application execution, it would not be useful in any way for the scaling governor.

To show this fact, Fig. 5(a) displays the results of a traditional LSTM trained for sequence workload detection. In the figure, the correct workload level is wkl_6, and the LSTM tries to predict the workload by giving the probabilities of each possible workload level per second. Ideally, the wkl_6 trace, depicted as a black line with circle markers, should have the maximum probability to indicate that the LSTM gives the correct prediction. However, with the existing training scheme, which emphasizes accuracy only at the end of the trace, it struggles to predict the correct workload before the end of execution, as demonstrated. The LSTM only provides accurate results around 200 seconds, i.e., at the end of the execution.

On the contrary, our objective is to attain accurate workload detections as early as possible to take frequency scaling via DVFS. Therefore, in this work, we introduce a training scheme specifically designed to address this issue and enable LSTM to provide accurate workload detections at the earliest possible



(a) Traditional LSTM



(b) LSTM trained with proposed training scheme

Fig. 5: Prediction results of trained LSTMs: (a) Detection of workloads from a traditional LSTM model trained for sequence classification. (b) Detection of workloads by an LSTM trained using the earlier proposed workload detection training scheme.

stage. The details of this training scheme are described in Algorithm 1. The first two lines represent standard initial procedures in machine learning training: model initialization and data splitting. Then, on line 3, the maximum trace length is saved as the variable len for the training data set. The algorithm then enters a training loop that continues as long as the current len is larger than the predefined threshold len_{th}. Inside this loop, the algorithm begins by pre-processing the data, truncating the samples in each trace with indices greater than len. Following this, the LSTM model is trained using the modified training dataset. After the first training iteration, the algorithm reduces the current trace length len by half (line 9), preparing it for the next iteration. This training enables the model to recognize patterns and relationships within truncated traces, enabling it to predict workload based on earlier, shorter data. Due to the proposed training scheme for early workload detection, the LSTM is trained to provide accurate workload detections in the early execution stages of the application, as illustrated in Fig. 5(b). After 18 seconds (less than 10% of the total execution time of the application), the proposed LSTM consistently provides precise workload-level predictions. This capability enables us to perform accurate and timely workload-driven DVFS actions.

C. Workload-Driven DVFS

With the optimal workload-frequency pairs chosen for the application and the LSTM model trained for early workload detection, we can introduce the workload-driven DVFS in this section. During runtime, while the application is running on the server, profiling tools continuously monitor the server and application metrics. Subsequently, these metrics are input into the LSTM for workload detection.

As previously introduced, the output of the LSTM, denoted as y_t, is a vector containing probabilities associated with various workload levels: $y_t = [P_{wkl_1}, P_{wkl_2}, \cdots, P_{wkl_N}]$. Given this information, we can calculate the optimal frequency at runtime as:

$$f_s(t) = y_t \times f_{wkls} = \begin{bmatrix} P_{wkl_1} \\ P_{wkl_2} \\ \cdots \\ P_{wkl_N} \end{bmatrix}' \times \begin{bmatrix} f_{wkl_1} \\ f_{wkl_2} \\ \cdots \\ f_{wkl_N} \end{bmatrix} \tag{5}$$

where $f_s(t)$ is a scalar value representing the suggested frequency resulting from the proposed workload-driven DVFS technique. This value can be seen as an expectation of the operating frequency. Finally, we can configure the system to operate at the suggested frequency through DVFS actions.

Algorithm 1: Proposed training scheme for earlier workload detection.

```
Data: Dataset of sampled application traces
Result: Trained LSTM model for workload detection
1 Initialize an LSTM model;
2 Split the dataset into training and testing sets;
3 $len$ ← maximum length of traces in the training
    dataset;
4 $len_{th}$ ← the specified length threshold for stopping;
5 while $len > len_{th}$ do
      /* Discard samples in each trace
         with index exceeding $len$          */
6     for each trace in the dataset do
7         remove samples $trace[len : end]$;
8     Train the LSTM model on the training dataset;
9     $len = len/2$;
```

TABLE I: Server Configuration

Model	HPE ProLiant DL380 Gen11
CPU	2x Intel Xeon Gold 6448Y
Core count	128 (64 logical cores)
Max turbo freq.	4100MHz
Min freq.	800MHz
Number of P-States	34
RAM	256 GB DDR5
OS	Ubuntu 23.04 (Linux 6.2.0-26-generic)
Virtualization	KVM, QEMU, libvirt, Open vSwitch

IV. Experimental Setup

This section describes the implementation and evaluation of the proposed GreenDVFS, including server configuration, benchmarks, profiling metrics, and comparison methods.

A. Server Configuration

In this work, we employ an HPE ProLiant DL380 Gen11 server, equipped with two Intel Xeon Gold 6448Y CPUs and 256 GB RAM memory. This configuration aligns with the standard specifications commonly found in cloud servers. The server's operating system is Ubuntu 23.04, running on the Linux 6.2.0-26-generic kernel. For a complete breakdown of this server's specifications, please refer to Table I. To imitate real-world scenarios, and with the purpose of better isolation and to facilitate the profiling stage in this work, each application was run on a virtual machine. The virtualization solution is also included in the table.

TABLE II: Workload levels for different applications

	wkl_1	wkl_2	wkl_3	wkl_4	wkl_5	wkl_6
DS (Throughput)	2K	7K	12K	17K	22K	27K
RS (Connections)	1	6	15	30	60	-
WS (Clients)	20	50	100	200	300	350

TABLE III: Representative profiled metrics

Linux perf	
IPS	instructions per second
ctx	context switches
tpd_retiring	retire bound from top-down analysis
tpd_fe_bound	Top-down analysis, frontend bound
page_faults	page faults per second
turbostat	
avg_mhz	average CPU frequency
busy%	% of time the core is active
coretmp	core temperature
pkgtmp	pakage temperature
pkgwatt	watts consumed by the pakage
libvirt	
rx_bytes	received network bytes
tx_bytes	sent network bytes
rd_bytes	read bytes from block device
wr_bytes	write bytes to block device
flush	number of flush operations

B. Benchmarks

To comprehensively evaluate the system performance and align with the cloud scenarios investigated in this study, we have selected a range of representative cloud benchmarks. These benchmarks can be summarized as follows:

Data Serving (DS): The Data Serving benchmark, from CloudSuite [7], is based on Yahoo! Cloud Serving Benchmark (YCSB). This benchmark leverages YCSB to load and stress Cassandra, a widely used NoSQL database. This mimics the behavior of a representative database system in the cloud.

Redis (RS): Redis [29], an open-source in-memory data structure store system, serves multiple roles as a database, cache, message broker, and streaming engine. We employ the Redis benchmark utility, which came along with the Redis installation, to simulate the execution of commands by multiple clients concurrently generating queries.

Web Search (WS): The Web Search benchmark from CloudSuite [7] relies on the Apache Solr search engine framework. It emulates real-world clients sending requests to the index server, which maintains indexes of text and fields extracted from crawled websites.

In this work, we carefully selected the workload to comprehensively stress the system across a broad spectrum, ranging from low to high workloads that exceed the system's capacity. Following this rule, the specific workload levels for the benchmarks are detailed in Table II.

C. Profiling Metrics

We employ a combination of Linux perf, turbostat, and the libvirt API to gather and monitor server and application metrics. We sample a total of 73 metrics per second. Representative examples are included in Table III. For instance, one of the evaluation metrics used in this work is the CPU's energy consumption, which is obtained by integrating the total watts consumed by the package (pkgwatt) during the application's execution time. Additionally, temperature, another important metric, is determined by using the package temperature (pkgtmp) while the application is in operation.

D. Comparison Methods

In our study, we examine six state-of-the-art governors in the Linux kernel [5], each with distinct behaviors:

powersave: This governor consistently operates the CPU at its minimum frequency, prioritizing power consumption and operating temperature over performance.

performance: In contrast to *powersave*, this governor runs the CPU with the maximum frequency, aiming for performance.

ondemand: This dynamic governor adjusts the frequency based on the current CPU load, it rapidly scales to the highest frequency and reduces it as idle time increases, balancing both performance and energy efficiency.

conservative: Similarly to *ondemand* in its dynamic scaling based on CPU load, the *conservative* governor, however, increases frequency more gradually, offering a more cautious approach to frequency scaling.

schedutil: Utilizing the per-entity load tracking (PELT) mechanism, this scheduler-driven governor aligns CPU frequency with the Linux kernel scheduler's demands, often considered an advanced replacement for the *ondemand* and *conservative* governors. This governor is loaded by default for Linux 4.9.5.

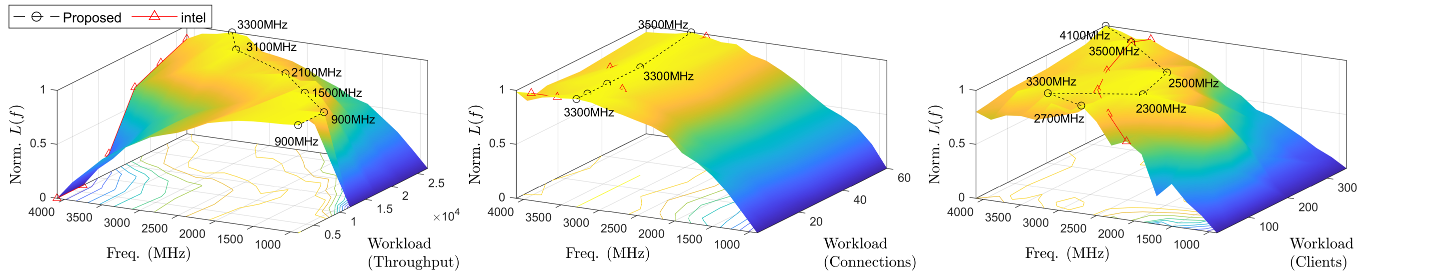
intel: The internal scaling governor for Intel CPUs when using the *intel_pstate* driver. It is similar to *schedutil*. However, it has the highest priority and is default loaded in servers with Intel CPUs. Therefore, we consider this governor as the baseline comparison method.

V. Experimental Results

The results of this study are described in four subsections. First, we introduce the effectiveness of the best frequency-workload pairs for different applications and workloads. Second, we evaluate the proposed LSTM-based technique for accurate and earlier workload detection. The following section assesses the proposed techniques, considering multiple aspects, including frequency, power consumption, temperature, and overall system performance. Lastly, we show that GreenDVFS is effective not only with known (trained) workloads, but also with unknown workloads that have not been seen or trained with previously.

A. DVFS Impact Analysis and Optimal Frequency-Workload Pairs

In the initial phase of our study, we performed an extensive profiling of applications under different workloads and frequencies. Our profiling efforts focus on assessing



(a) Data Serving (b) Redis (c) Web Search

Fig. 6: Normalized $L(f)$ values for three applications with dynamic workload configurations. The best frequency-workload pairs for each application are illustrated as a black dotted line with circle markers, in each subfigure. The DVFS decision of the *intel* governor is illustrated in the red line with triangle markers.

performance metrics such as application performance, server temperature, and power consumption. Subsequently, we collect the profiling data and feed them into our proposed optimizer, denoted as $L(f)$ in Eq. 1. Fig. 6 illustrates the 3D $L(f)$ plot for three different applications, where the X and Y axes represent workload and frequency, respectively, while the Z-axis indicates the normalized value of the loss function $L(f)$. The coefficient factors, specifically α, β, and γ, are set to 0.5, 0.3, and 0.2, respectively. These values are chosen through a heuristic approach to optimize application performance while reducing energy consumption and server operation temperature. Adjusting these parameters can result in varied optimization outcomes. For instance, decreasing the value of α while increasing the value of β and γ may lead to greater power and temperature reduction, however, at the expense of the application's performance level. The objective of this analysis is to maximize $L(f)$, thereby identifying the optimal frequencies for different workloads. We present the best frequency-workload pairs using black dashed lines with circles. The red line with triangle markers represents the DVFS decision from the *intel* governor, which demonstrates that the *intel* governor falls short in selecting optimal frequency in most cases.

Fig. 6 also provides valuable insights into the diverse characteristics of the different applications. For instance, in the case of Data Serving (Fig. 6(a)), higher frequencies are required as the workload increases. In contrast, Redis (Fig. 6(b)) consistently demands a stable CPU frequency to maintain its operations. The behavior of the Web Search application (Fig. 6(c)) is particularly complicated. When the workload is low, a higher frequency is optimal. However, as the workload increases, the optimal frequency initially decreases before again rising. This peculiar behavior can be explained by considering the P and C states of the CPU [30]. In scenarios of low workload and sparse requests, the CPU can operate at a higher frequency to complete tasks quickly and then transition into a longer power-saving sleep mode. As the workload increases, it becomes more efficient to maintain the CPU at a moderate frequency instead of frequently transitioning between active and sleep modes. Finally, when the workload reaches high levels, increasing the CPU frequency to meet the expected demands becomes necessary.

TABLE IV: Results of earlier workload detection

App	MisPre. (#)	Accuracy (%)
Data Serving	16.3	94.1
Redis	4.2	97.7
Web Search	30.0	90.0

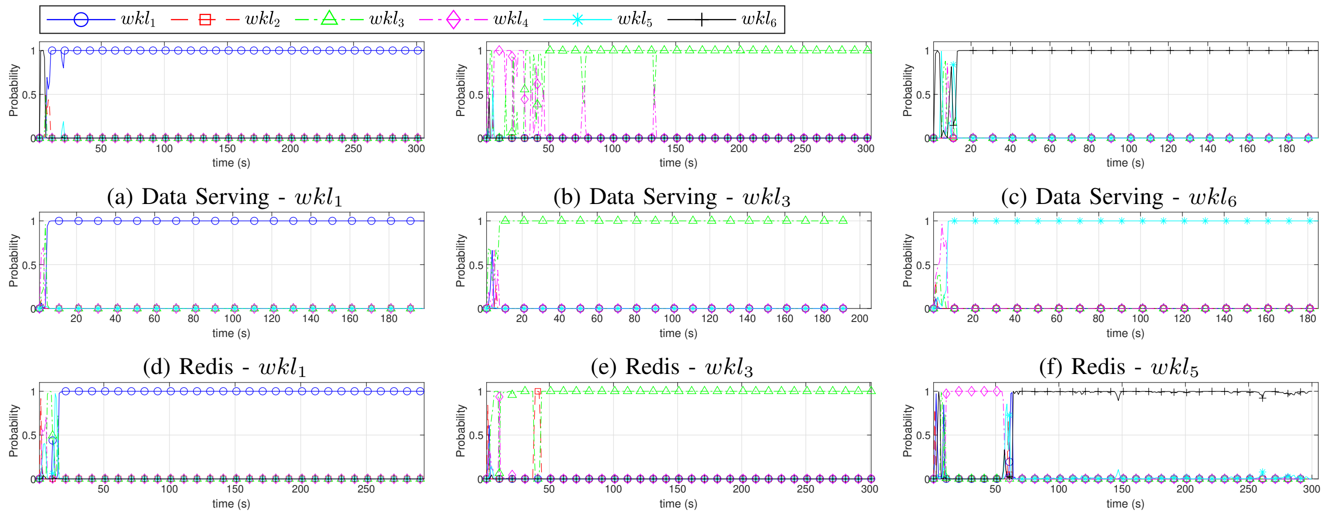
B. *Evaluation of LSTM-Based Workload Detection*

In this work, we train the LSTM model using 80% of the profiling data, allocate the remaining 20% for the test set. Fig. 7 illustrates the effectiveness of LSTM trained for each application in three distinct workloads, namely light, medium and heavy workloads.

More specifically, Fig. 7(a) demonstrates the behavior of our proposed LSTM-based method during the execution of the Data Serving under a light workload (wkl_1). Within a few seconds, our LSTM-based approach provides accurate and consistent workload detection results, offering precise guidance for DVFS actions. When the workload reaches its maximum level (wkl_6), the LSTM model still maintains its accuracy from the beginning. A similar trend is observed for the Redis, as depicted in Figs. 7(d-f).

In the case of Data Serving, at medium workload level (wkl_3 in Fig. 7(b)), the LSTM occasionally predicts either wkl_3 or wkl_4 before converging to wkl_3. This behavior arises because wkl_3 and wkl_4 initially exhibit similar characteristics, which requires additional data to decisively determine the actual workload, whether it is wkl_3 or wkl_4. This similarity in behavior is also observed in the Web Search. In the initial phase of this application, different workloads often display similar behaviors for initialization, making workload detection more challenging. However, it is important to note that the impact of such mispredictions between similar workload levels is demonstrated to be negligible in the subsequent overall results comparison section. The reason is that if two workloads share similar behavior within the same time slot, the same frequency level also optimizes for both during that time.

In different applications, we calculated the average number of mispredictions generated by our earlier workload detection method and collected the results in Table IV. In summary, our proposed method exhibits an average of fewer than 30 mispredictions, accounting for less than 10% of the total pre-



(a) Data Serving - wkl_1 (b) Data Serving - wkl_3 (c) Data Serving - wkl_6

(d) Redis - wkl_1 (e) Redis - wkl_3 (f) Redis - wkl_5

(g) Web Search - wkl_1 (h) Web Search - wkl_3 (i) Web Search - wkl_6

Fig. 7: The proposed LSTM-based method demonstrates effective workload detection across multiple application and workload scenarios, offering both accuracy and consistency in earlier workload detection.

dictions. Consequently, the accuracy of our workload detection consistently surpasses 90% for all applications.

C. *Overall Evaluation of GreenDVFS*

In this section, we evaluate the proposed GreenDVFS approach. These evaluation results are derived from the interaction among the various techniques proposed in this study, including the best frequency-workload pairs, the LSTM-based workload detection, and the collective impact of the workload-driven DVFS method. To facilitate the comparison, we first compare the proposed method with the baseline *intel* scaling governor across various aspects, as illustrated in Fig. 8. Subsequently, we present a comprehensive comparison with other Linux scaling governors in Table V.

In the case of Data Serving (Figs. 8(a)-(c)), *intel* consistently operates the CPU at the highest frequency level, leading to unnecessary high power consumption and temperatures. On the contrary, the proposed method adjusts the DVFS level according to workload detection results, offering a more adaptive and efficient control strategy.

In the context of Redis (from Figs. 8(d)-(f)), *intel* frequently fluctuates the CPU frequency, resulting in less stable performance. Conversely, our proposed method adapts the DVFS level based on workload detection results, ensuring a more consistent and reliable control approach. Therefore, the proposed method achieves both lower power consumption and operational temperature.

When it comes to Web Search (from Figs. 8(g)-(i)), our proposed method exhibits flexible adaptability in handling complex workload scenarios, enabling it to select the optimal DVFS level for the application under different workload levels, i.e., higher frequency for both light and heavy workloads, lower frequency for medium workload. In contrast, *intel* fails to provide such tailored optimization.

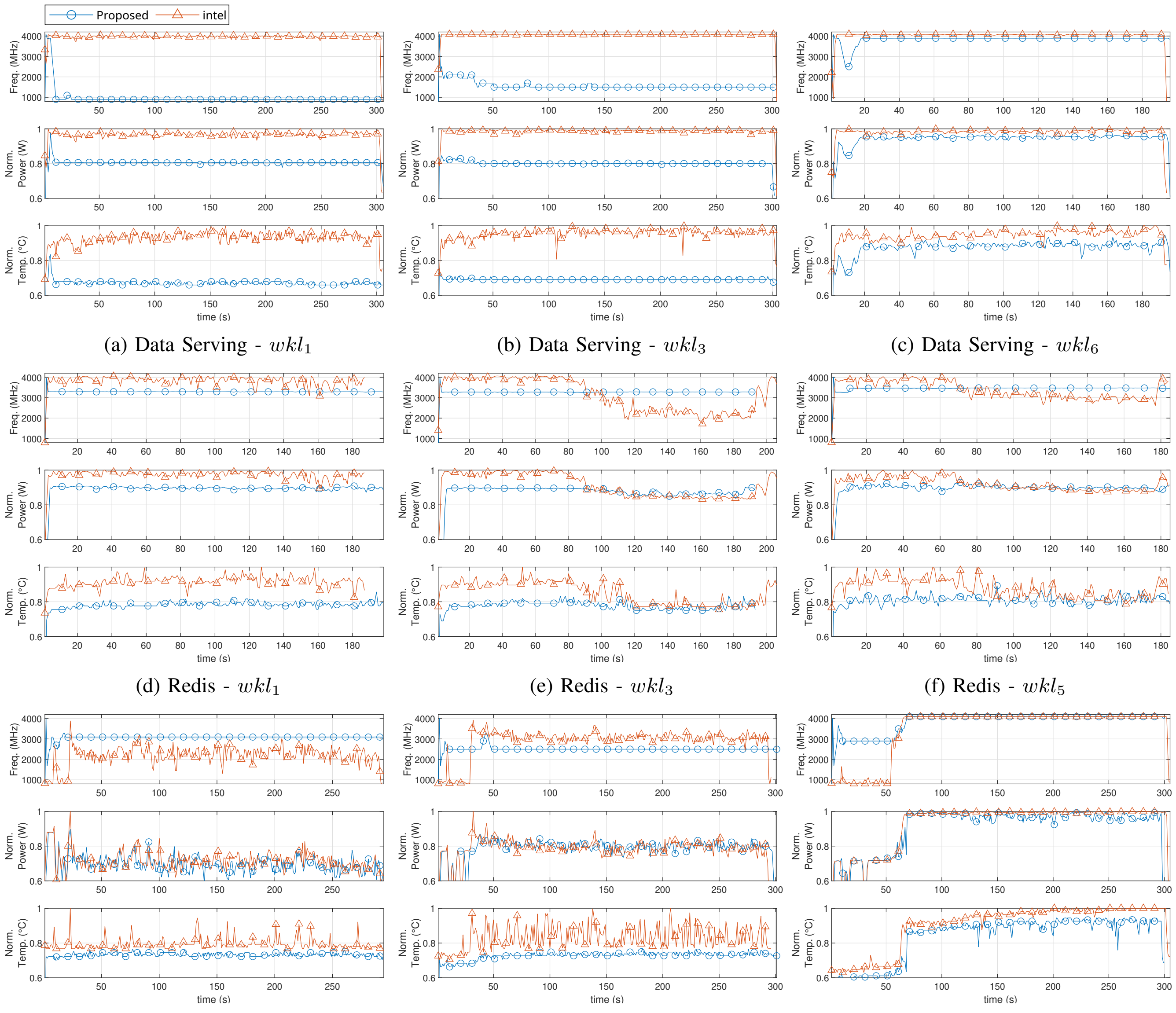
Moreover, the proposed workload-driven method offers a significant advantage in enhancing the energy efficiency of cloud servers without sacrificing application performance. This is evidenced by the comparative analysis detailed in Table V, which includes the proposed method and the other five Linux scaling governors: *powersave*, *ondemand*, *conservative*, *schedutil*, and *performance*. The improvement (%) of each different scaling governor is compared against the default *intel* governor of our test server.

In terms of performance, as expected, the *powersave* governor can lead to a significant performance degradation, up to 156%, while the *performance* governor can enhance performance by up to 4.9% compared to *intel*. The other governors exhibit similar trends, but do not exceed the performance of the *intel* governor. In contrast. the proposed workload-driven DVFS technique consistently outperforms the *intel* governor thanks to its ability to find the optimal frequency point for the application. Regarding power consumption, *powersave* has the lowest consumption, with reductions of up to 22.8%. Close behind is our *proposed* method, which can reduce power usage by up to 16.7%.

The energy metric, which represents the cumulative power consumption over time, reveals that *powersave* can unexpectedly increase energy usage by over 65.9% due to extended CPU run times at slower speeds. Consequently, the *powersave* governor does not necessarily translate to energy savings. On the contrary, the *performance* governor shows little energy savings in certain scenarios by completing tasks more quickly, allowing the CPU to enter a low-power state earlier. However, our proposed method stands out as the best, demonstrating substantial CPU energy savings of up to 18.3% due to the optimized DVFS action.

Temperature control is another forte of the proposed method. By implementing more steady control and power reduction strategies, GreenDVFS can maintain temperatures up to 30% lower than the default *intel* governor.

In conclusion, the proposed method not only preserves ap-



(a) Data Serving - wkl_1 (b) Data Serving - wkl_3 (c) Data Serving - wkl_6

(d) Redis - wkl_1 (e) Redis - wkl_3 (f) Redis - wkl_5

(g) Web Search - wkl_1 (h) Web Search - wkl_3 (i) Web Search - wkl_6

Fig. 8: The comparison between the proposed workload-driven DVFS (in blue line with circle markers) and the default Linux *intel* scaling governor (in orange line with triangle markers) reveals advantages of the proposed method across various aspects.

plication performance, but also significantly improves energy efficiency and lowers operating temperatures compared to existing Linux scaling governors. This demonstrates its potential to operate servers in a more sustainable and environmentally friendly manner.

D. Evaluation of GreenDVFS on Unknown Workloads

The analysis presented so far has been conducted using known workloads of the application, which has been seen by the LSTM and the proposed method. However, in real-world scenarios, the application may encounter workloads that the LSTM has not been trained to handle. In this context, we investigate the LSTM's performance in scenarios involving unknown workloads in this section. The configurations for these unknown workloads are outlined in Table VI. Our approach involved selecting application workloads that are not included in the LSTM's training set. As a result, the unknown workload levels vary considerably from those listed in Table II.

Thanks to the proposed LSTM-based workload detection's ability to output the probability distributions of possible workload levels. It classifies unknown workloads as similar to known and trained workload levels. Then, the proposed workload-driven DVFS technique calculates the optimal frequency, taking into account these probability distributions for unknown workloads. As a result, GreenDVFS demonstrates remarkable robustness when faced with untrained workload.

The final experimental results demonstrate the advantages of the proposed method compared to the *intel* governor are illustrated in Fig. 9. In summary, it takes optimal DVFS actions for unknown workloads to improve energy efficiency by up to 19% and lower operating temperature by up to 35% without

TABLE V: Improvements (%) of different scaling governors against *intel*

Method	*powersave*			*ondemand*			*conservative*			*schedutil*			*performance*			*proposed*		
App.	WS	RS	DS	WS	RS	DS	WS	RS	DS	WS	RS	DS	WS	RS	DS	WS	RS	DS
Perf	-156.4	-70.5	-38.5	0.3	-3.6	0.0	-1.1	-6.9	0.0	-0.5	-3.5	0.1	0.9	4.9	-0.1	0.9	1.3	1.3
Power	7.9	15.6	22.8	-0.4	4.0	0.4	0.0	3.2	0.0	-3.0	1.0	0.1	1.6	-1.6	-0.1	3.6	5.2	16.7
Energy	-42.1	-65.9	-24.5	0.0	0.1	0.4	-1.0	-4.0	0.0	-3.3	-2.6	0.2	2.5	3.2	-0.3	4.6	6.5	18.3
Temp	21.6	18.2	32.4	0.7	4.5	0.0	5.4	3.3	-0.1	1.8	1.1	0.0	0.9	-3.8	-1.1	12.9	12.0	30.1

TABLE VI: Unknown workloads setting for different applications

	wkl_{u1}	wkl_{u2}	wkl_{u3}	wkl_{u4}	wkl_{u5}	wkl_{u6}
DS (Throughput)	4.5K	9.5K	10K	14.5K	19.5K	24.5K
RS (Connections)	3	10	20	25	42	-
WS (Clients)	35	75	125	150	250	325

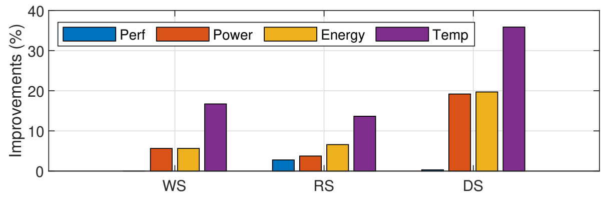


Fig. 9: The improvements of proposed workload-driven DVFS over the *intel* governor when facing with unknown and untrained workloads.

compromising application's performance.

VI. Conclusions

With the absence of process and application's real demands information, existing Linux scaling governors need to use inaccurate CPU load information for frequency scaling, leading to non-optimal DVFS actions. To address this challenge, in this work, we have proposed an LSTM-based, time series, workload-driven DVFS methodology, called GreenDVFS. Our approach has been designed to detect the runtime workload characteristics with high accuracy, thus facilitating the selection of an optimal runtime frequency. This enables us to boost energy efficiency and reduce the operational temperature for cloud servers and applications.

Our experimental results indicate that by incorporating detailed application profiling and real-time workload detection, we have achieved energy savings of up to 18% and reduction in operational temperature of 30% without compromising the performance of the applications. We believe that this proposed methodology demonstrates a way for more eco-friendly and sustainable cloud server and application management practices, contributing to the broader goal of fostering a more sustainable society.

Acknowledgments

This work has been partially supported by the HeatingBits S4S project of EPFL, an Industrial Research Grant from the Huawei Cloud Intelligent Cloud Technologies Initiative to ESL-EPFL, and Grant PID2021-126576NB-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe".

The authors would like to thank Dr. Xavier Ouvrard and Dr. Miguel Peon Quiros, who are part of the EcoCloud Center of EPFL, for their support with the experimental evaluation of the server infrastructure.

References

[1] Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023
[2] A. S. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.
[3] R. Ge *et al.*, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2009.
[4] D. C. Snowdon *et al.*, "Power management and dynamic voltage scaling: Myths and facts," in *Proceedings of the 2005 workshop on power aware real-time Computing*, vol. 31, 2005, p. 34.
[5] D. Brodowski *et al.*, "CPU frequency and voltage scaling code in the Linux kernel," Linux kernel documentation, p. 66, 2013.
[6] Load tracking in the scheduler. [Online]. Available: https://lwn.net/Articles/639543/
[7] T. Palit *et al.*, "Demystifying cloud benchmarking," in *IEEE Int. Sym. on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 122–132.
[8] C. Lin *et al.*, "A workload-aware DVFS robust to concurrent tasks for mobile devices," in *Proceedings of the 29th Annual Int. Conf. on Mobile Computing and Networking*, 2023, pp. 1–16.
[9] G. Ali *et al.*, "Automating CPU dynamic thermal control for high performance computing," in *2022 22nd IEEE Int. Sym. on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 514–523.
[10] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the linux Sym.*, vol. 2, no. 00216, 2006, pp. 215–230.
[11] C. Scordino *et al.*, "Real-time and energy efficiency in Linux: theory and practice," *ACM SIGAPP Applied Computing Review*, vol. 18, no. 4, pp. 18–30, 2019.
[12] R. Hebbar and A. Milenković, "PMU-events-driven DVFS techniques for improving energy efficiency of modern processors," *ACM Trans. on Modeling and Performance Evaluation of Computing Systems*, vol. 7, no. 1, pp. 1–31, 2022.
[13] D. P. Johnson *et al.*, "Frequency scaling of processing unit based on aggregate thread CPI metric," Jul. 10 2012, US Patent 8,219,993.
[14] L. Costero *et al.*, "Dynamic power budget redistribution under a power cap on multi-application environments," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100865, 2023.
[15] S. Akram *et al.*, "DVFS performance prediction for managed multithreaded applications," in *IEEE Int. Sym. on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 12–23.
[16] Y. Liu *et al.*, "Fastcap: An efficient and fair algorithm for power capping in many-core systems," in *IEEE Int. Sym. on performance analysis of systems and software (ISPASS)*, 2016, pp. 57–68.
[17] M. Korkmaz *et al.*, "Workload-aware CPU performance scaling for transactional database systems," in *Int. Conf. on Management of Data*, 2018, pp. 291–306.
[18] K. Liu *et al.*, "Vincent: Green hot methods in the JVM," *Science of Computer Programming*, vol. 230, p. 102962, 2023.

[19] J.-G. Park *et al.*, "Ml-gov: A machine learning enhanced integrated cpu-gpu dvfs governor for mobile gaming," in *Proceedings of the 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia*, 2017, pp. 12–21.

[20] F. M. M. ul Islam *et al.*, "Task aware hybrid dvfs for multi-core real-time systems using machine learning," *Information Sciences*, vol. 433, pp. 315–332, 2018.

[21] D. Huang *et al.*, "Reinforcement learning-based joint reliability and performance optimization for hybrid-cache computing servers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5596–5609, 2022.

[22] J. Yang *et al.*, "Workload predicting-based automatic scaling in service clouds," in *IEEE Sixth Int. Conf. on Cloud Computing*, 2013, pp. 810–815.

[23] K. Cetinski and M. B. Juric, "AME-WPC: Advanced model for efficient workload prediction in the cloud," *Journal of Network and Computer Applications*, vol. 55, pp. 191–201, 2015.

[24] R. Cao *et al.*, "Load prediction for data centers based on database service," in *IEEE 42nd annual computer software and applications Conf. (COMPSAC)*, vol. 1, 2018, pp. 728–737.

[25] W. Zhong *et al.*, "A load prediction model for cloud computing using pso-based weighted wavelet support vector machine," *Applied Intelligence*, vol. 48, no. 11, pp. 4072–4083, 2018.

[26] J. Gao *et al.*, "Machine learning based workload prediction in cloud computing," in *29th Int. Conf. on computer communications and networks (ICCCN)*, 2020, pp. 1–9.

[27] V. Jayakumar *et al.*, "A self-optimized generic workload prediction framework for cloud computing," *IEEE Int. Parallel and Distributed Processing Sym. (IPDPS)*, pp. 779–788, 2020.

[28] I. Goodfellow *et al.*, *Deep learning*. MIT press, 2016.

[29] Redis benchmark. [Online]. Available: https://redis.io/docs/management/optimization/benchmarks/

[30] D. Hackenberg *et al.*, "An energy efficiency feature survey of the intel haswell processor," in *IEEE Int. parallel and distributed processing Sym. workshop*, 2015, pp. 896–904.