ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

Energy-Efficient Per-Core DVFS for Virtual Machine Management in Cloud Data Centers

Kritwara Rattanaopas[1] and Pichaya Tandayya[2]

ABSTRACT

Energy efficiency and thermal management are critical challenges in virtualized cloud data centers, particularly for optimizing parallel workloads. Dynamic Voltage and Frequency Scaling (DVFS) is widely used to balance power consumption and computational performance. This study proposes an Adaptive Threshold Per-Core DVFS Governor that dynamically adjusts CPU core frequencies based on per-core utilization, improving energy efficiency and workload performance. The proposed algorithm is evaluated using Charm++ parallel workloads and is benchmarked against existing Linux governors, including the Conservative, OnDemand, and Performance governors. Experimental results demonstrate that the proposed approach achieves superior energy efficiency per Giga-instructions compared to the Conservative and OnDemand governors while maintaining performance levels comparable to the Performance governor. Furthermore, the proposed method reduces average CPU temperature by approximately 5% (2.5°C lower) compared to the Performance governor, contributing to enhanced thermal management in cloud computing environments. These findings highlight the potential of the adaptive per-core DVFS mechanism for improving energy efficiency and performance in virtualized data centers.

Article information:

Keywords: Real-time Operating System, Energy-efficient Computing, Per-core DVFS, Linux Kernel Power Management

Article history:
Received: October 14, 2024
Revised: March 22, 2025
Accepted: April 8, 2025
Published: April 26, 2025 (Online)

DOI: 10.37936/ecti-cit.2025192.258820

1. INTRODUCTION

Currently, most data center infrastructures are based on cloud technology. Cloud data centers generally feature a huge number of resources on either heterogeneous or homogeneous physical clusters. Virtualization technology is the key to helping break the limitations of both physical cluster types. Virtual machines can be allocated to all physical machines in clusters through the virtualization layer, which is called a hypervisor in Infrastructure as a Service (IaaS) cloud standard model. The IaaS model is a dynamic consolidation of virtual machines. There have been many research issues concerning improving energy efficiency and resource utilization in data centers [1] The energy efficiency techniques in IaaS cloud platforms along with dynamic management can be grouped into two levels, including the hardware level (physical machines) and the software level (virtualization). The hardware level focuses on the Central Processor Unit (CPU) and the link state adaptation of network communication. The software level focuses on virtual machines for hardware consolidation and resource allocation. Research topics for virtualization and virtual machines include scheduling techniques like Bin-Packing, Evolutionary Genetic Algorithm (GA), and Natural Inspired Algorithms (NIA). These algorithms applied the most popular methods to allocate and provision for virtual machines to achieve the required optimization and energy efficiency.

Dynamic Voltage Frequency Scaling (DVFS) is supported by modern processors, as this technique is now common to adjust the frequency and the voltage on each CPU core. For example, the DVFS scheme [2] helped limit the processor's temperature and the DVFS schedulers [3],[4] benefited in virtualized cloud data centers. Previous DVFS models [5],[6],[7] studied various types of parallel workloads that challenge reducing the average energy consumption.

However, previous works did not address dynamic workloads nor provided a dynamic DVFS threshold for controlling CPU frequency at the Linux kernel

[1,2]The authors are with the Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University, Thailand, Email: kritwara.r@psu.ac.th and pichaya.t@psu.ac.th
[2]Corresponding author: pichaya.t@psu.ac.th

level. This paper investigates dynamic frequency scaling on each active CPU core, aiming to optimize both the CPU frequency and utilization thresholds to reduce energy consumption while maintaining performance comparable to that of the Performance governor, and better than the default OnDemand governor and other alternative governors in terms of energy efficiency. This approach helps prevent overheating and balances performance with utilization. The paper proposes an Adaptive Threshold Per-core DVFS governor based on the Linux governor, which will be tested using parallel workloads on physical machines.

The paper is organized as follows: Section 2 reviews related work. Section 3 presents the static per-core DVFS design, which leads to a more optimized adaptive threshold per-core DVFS. Section 4 provides background on DVFS and virtualization. Section 5 describes the experimental setup and parallel workloads. Section 6 discusses the experimental results, including comparisons of average temperatures and energy consumption. Finally, Section 7 concludes with a summary of the proposed algorithm's performance and potential future work.

2. BACKGROUND

This section provides some background related to power consumption models and Linux's DVFS governors. Firstly, it introduces the DVFS and illustrates the OnDemand and Conservative governors, which are deployed in the Linux kernel version 2.6.9 and later. Finally, it describes the virtualization technology in our testing system architecture.

2.1 Dynamic voltage frequency scaling

Dynamic voltage frequency scaling (DVFS) [8][9] can help create flexibility in the CPU power capability by changing its frequency. This algorithm has default policies that can be implemented with the Linux kernel called "Governor." The DVFS governors include OnDemand, Conservative, and UserSpace on Intel Xeon [10][11].

The CPU Frequency Scaling (CPUFreq) is implemented along with the Linux kernel to control the CPU's frequency, which works like a device driver. The cpufreq file in the Linux kernel facilitates the viewing and editing of the frequency for each CPU core located at the path "/sys/devices/system/cpu/cpux." There are various "governor" policies for CPUFreq that cater to different performance requirements. The governor in our testing system architecture works for Intel Xeon Quad-cores E5630 and features five types, including Performance, PowerSave, Conservative, OnDemand, and User-level. The power consumption model of CMOS logic gates can be formulated as shown in Equation (1) [8].

$$P_{work} = CfV^2 + P_{idle} \quad (1)$$

where $\boldsymbol{P_{work}}$ represents active power consumption. The term $\boldsymbol{C}$ is the capacitance of the gate, which affects the energy required for switching, while $\boldsymbol{V}$ is the applied voltage, with power consumption increasing quadratically with $\boldsymbol{V}$. The clock frequency $\boldsymbol{f}$ controls the rate of switching, and higher frequencies lead to greater power usage. $\boldsymbol{P_{idle}}$ accounts for the static power consumed when the system is idle, primarily from leakage currents. Additionally, the relationship between frequency and voltage is linear, $\boldsymbol{f \propto V}$ and $\boldsymbol{P_{idle}}$ also include energy consumption from other server components. Optimizing both dynamic and idle power is essential for enhancing energy efficiency, especially in low-power, high-performance systems.

2.2 Governors in the Linux kernel

The Linux kernel offers various CPU frequency scaling governors [10] to balance power efficiency and performance. The Performance governor maintains the CPU at its highest frequency within the defined minimum and maximum frequency limits to ensure optimal performance. In contrast, the Powersave governor operates the CPU at the lowest frequency within the same range to prioritize energy savings. The Userspace governor enables users or programs with root privileges to manually set the CPU frequency via the *sysfs* interface. Meanwhile, the OnDemand governor adjusts the CPU frequency dynamically based on system load, with frequency changes triggered by the scheduler's load estimation mechanism. These governors allow the system to manage CPU performance and power consumption according to varying workload demands.

2.3 OnDemand governor

The OnDemand governor [9][10] of the Linux kernel version 2.6.9 or later can be described as shown in Algorithm 1. This governor can change the frequency inside the kernel with little overhead and determine the optimized frequency corresponding to the CPU utilization.

Algorithm 1. Original OnDemand algorithm [9]
Input: utilization
Output: the frequency of each core processor
1. **Define** cpu-frequency list
2. **for** every CPU in the system
3. every *X* millisecond
4. Get the utilization since the last check
5. **if** (utilization > *UP_THRESHOLD*) **then**
6. Increase the frequency to MAX
7. every *Y* millisecond
8. Get the utilization since the last check
9. **if** (utilization < *DOWN_THRESHOLD*) **then**
10. Decrease the frequency by 20%

Algorithm 1 is the original OnDemand algorithm. It includes the policy on how to increase and decrease the frequency. If the current CPU utilization is more

than the $UP\_THRESHOLD$ setup point, the new frequency of the current core will be increased to the maximum available frequency, and this process is repeated every X millisecond. If the current CPU utilization is lower than the $DOWN\_THRESHOLD$ setup point, the new frequency of the current core will be decreased by 20% and this process is repeated every Y millisecond.

In addition, the Linux kernels older than version 2.6.9 have only classical governors, including Performance and PowerSave. The Performance model maintains the highest possible frequency of CPU and the PowerSave model maintains the lowest possible frequency of CPU.

2.4 Conservative governor

The Conservative governor [10] in Linux adjusts the CPU frequency based on system load, similar to the OnDemand governor, but does so more gradually. It increases or decreases the CPU speed in smaller steps, making it more energy-efficient and suitable for battery-powered devices. While OnDemand responds quickly to load changes, Conservative provides smoother scaling, balancing power consumption and performance. The OnDemand governor is the default in many Linux distributions, but users can switch to Conservative for better power management on portable devices.

2.5 Virtualization

Virtualization technology [8][13][14] is widely used in the cloud data centers. It deploys software for various OSes and builds virtual hardware for some appliance servers. Hypervisor is a software layer in the Linux kernel that can provide several execution environments. Virtual Machines (VMs) with multiple guest OSes are built over this layer. This solution is not only provided by commercial software but also by open-source software. For example, the popular open-source software includes KVM, Xen, and VMware free ESXi. These are bare-metal hypervisors. Thus, their instance machine is called a virtual machine which runs a guest OS. Meanwhile, OpenVZ and Docker are the popular alternative hypervisors that use a container-based virtualization platform. Therefore, their instance machine is called a container.

This paper investigates the KVM hypervisor. KVM is a classic and native hypervisor working on the Redhat and Debian OSes running on physical servers. Also, it is the most popular virtualization technology [8], like Xen and VMware, exploited in the virtualized cloud data center. Moreover, it supports the $perf-kvm$ which is a VM self-performance monitor tool similar to Linux's $perf$ for a physical machine. The KVM kernel supports both 32-bit and 64-bit QEMU native performance. In addition, it does not only run multiple VMs on various guest OSes, but it also supports Virtual Shell ($virsh$) commands.

3. RELATED WORKS

Nowadays, most data centers are concerned about and committed to the energy efficiency scheme. Cloud data centers usually apply several approaches together to optimize the power consumption for physical machines. The taxonomy of energy-efficient cloud data centers [8] divides such power management techniques into static and dynamic policies. Dynamic power management is one of the most popular research topics. In 2005, the power reduction in microprocessor techniques only focused on the hardware level and the first generation of Dynamic Voltage Scaling. Moreover, AMD virtualization (AMD-V) and Intel virtualization (VT-x) are the first generation of x86 hardware virtualization. Most virtualization data centers voluntarily employ their own alternative data center architectures. Consequentially, this raised more energy efficiency research topics. Also, the hypervisor-based system is a new layer situated between the kernel and hardware levels in virtualization technology. The energy management prototypes over the hypervisor level [15][16] were investigated and compared between a host-level subsystem and a guest-level energy management without DVFS support in 2007. In 2005, the hardware level [15] with Dynamic Voltage Scaling (DVS) was the starting point for the microprocessor-supported type. However, the energy-aware scheduling framework [4] focused on controlling the physical machines with the bin-packing technique. Moreover, Mobile Cloud Computing (MCC) and Vehicular Networking (VN) [3] focused on the edge of network devices with the new cloud-fog architecture. The cool virtualization data center with DVFS [2] used a limiting processor temperatures scheme on a case study of 32 nodes (128 cores) on Intel Xeon processors. However, the longer execution time was the trade-off for saving energy and cooling.

Table 1 compares related works and summarizes their methodologies and contributions, emphasizing the applications of the DVFS technique for power reduction across both physical machines and simulation tools.

Table 1: *Comparison of Related Works on DVFS and Linux Governor.*

Comparison with Current Work	Related Works
DVFS only, without Linux governor.	[5], [7], [18]
DVFS only in simulation	[6], [8]
DVFS and Linux governor across all CPUs.	[15], [18], [20], [21]
Per-core DVFS for CPU: DVFS programming with Linux governor.	[17], [10], [12], [22], [23], [24]

Acun *et al.* [17] highlighted the importance of optimizing interval time to reduce energy consumption

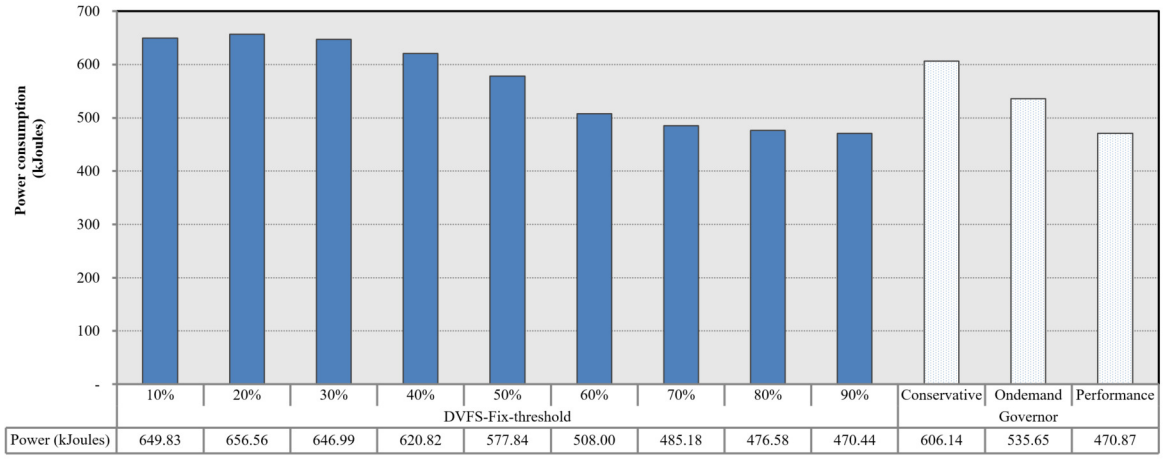


Fig.1: *Energy consumption for each Per-core DVFS threshold compared with the standard governors, including Conservative, OnDemand, and Performance.*

and CPU utilization. Building on this, we propose the Adaptive Threshold Per-Core (ATP) governor, derived from the OnDemand governor's frequency-switching mechanism. Integrated with virtualization technology, ATP enhances workload management, optimizes energy consumption, and mitigates power-performance trade-offs observed in Krzywda *et al.* [18]. Based on the Dynamic Voltage and Frequency Scaling (DVFS) approach in Edge Computing- Based Microservices (ECM) proposed by Zouhir *et al.* [24], ATP is designed for latency-sensitive applications like Virtual Reality/Augmented Reality (VR/AR) and Internet of Things (IoT), dynamically adjusting CPU frequency based on real-time core utilization and eliminating the need for predefined control modes.

4. PROPOSED METHODOLOGY

Our proposed algorithms are different from several fundamental techniques in that they aim to handle naturally dynamic workloads with a dynamic threshold of DVFS for controlling the CPU frequency in the Linux kernel layer. Currently, most processors support the Advanced Configuration and Power Interface (ACPI) of which the P-state, the coprocessor Power Management (PM), concerns the CPU frequency scaling, which is called "*acpi-cpufreq*" We propose two algorithms to modify the OnDemand governor concept. The first algorithm is called "*Per-core DVFS"* which offers a static CPU utilization threshold. The second algorithm or "*Adaptive threshold per-core DVFS"* offers a dynamic CPU utilization threshold that addresses the CPU resources allocated to the workload.

4.1 Per-core DVFS

Algorithm 2 uses a static threshold to increase and decrease the CPUFreq on each CPU core. We propose a static DVFS approach. We provide only one threshold, which is different from the OnDemand governor. Our DVFS algorithm uses the original idea of the OnDemand governor for increasing and decreasing the frequency of each CPU core. The goal of increasing the frequency is to provide the maximum available frequency per core to support a new increasing workload.

Workload overload can affect the CPU temperature. If we elevate only one higher step from the current frequency, then the new workload may still be overloaded, which may still increase the CPU temperature. According to the Intel Xeon feature, the decreasing frequency method performs just one lower step from the current frequency, which does not exactly equal 20% as appeared in the original algorithm. We cannot immediately step the frequency down to the minimum because the next workload may spike up again, causing the CPU to overload as well as increasing the temperature.

Algorithm 2. Per-core DVFS
Input: the processor idle times and frequencies
Output: the frequency of each core processor

```
1:  Define the cpu-frequency list
2:  Define the policy for the performance governor
3:  Define the threshold setup point
4:  While (True)
5:    Require: all cores' processor utilization
6:    Require: all cores' processor frequency
7:    for core ∈ Cores_total do
8:        if utilization(core) < threshold then
9:        Update to the maximum Frequency
10:                  else
1.        Lower the cpu-frequency for one step
11:       end
12:   end for
13: end while
```

In Algorithm 2, the per-core DVFS focuses on a single static threshold value. The condition in Line 8 is to compare the CPU utilization with a static threshold. If the CPU utilization is more than the threshold, then it will increase the CPU frequency to the maximum available frequency. The upper step is to increase the CPU frequency to the maximum avail-

able frequency because the design aims to prevent overheating (temperature rising) from the overload state. Moreover, in case of lowering the CPU utilization, it is to decrease the CPU frequency for one step. In this case, Intel Xeon Quad-cores E5630 has eight available frequencies, including 0.6 GHz, 1.73 GHz, 1.87 GHz, 2.00 GHz, 2.13 GHz, 2.27 GHz, 2.40 GHz, and 2.54 GHz.

4.2 Adaptive threshold per-core DVFS

The concept of per-core Dynamic Voltage and Frequency Scaling (DVFS) serves as the foundation for our proposed adaptive threshold per-core DVFS technique. Fig. 1 depicts the energy consumption outcomes corresponding to each threshold step in the per-core DVFS scheme. These measurements were obtained using the Power Indicator, which interfaces with the *apcupsd* service for APC UPS units. Data for energy consumption at each threshold step was retrieved through the *apcaccess* command. The thresholds range from 10% to 90% of the CPU utilization and the gap between each step is 10%. We compare the per-core DVFS on each threshold with the standard governor policies, including Conservative, OnDemand, and Performance.

Figure 1 presents the energy consumption for the per-core DVFS threshold using a bar graph. The first three bars on the left represent the standard Linux governors: the Conservative governor, which has the highest energy consumption at approximately 606.14 kJoules; the OnDemand governor, consuming around 535.65 kJoules; and the Performance governor, which consumes the least energy at about 470.87 kJoules. The remaining bars represent different per-core DVFS configurations, allowing direct comparison with the standard governors.

Our adaptive threshold per-core DVFS consumes energy higher than the other standard governors if the DVFS threshold is lower than 45% of CPU utilization. It consumes energy lower than the OnDemand governor when the DVFS threshold is more than 55% of CPU utilization. It approaches the amount of the Performance governor when the DVFS threshold is more than 85% of CPU utilization but with the trade-off for its performance and efficiency.

Consecutively, we select the CPU utilization range for the per-core DVFS thresholds in between 45% to 85% as the base for our new algorithm. For usage efficiency, the minimum threshold should be higher than 45% and the maximum threshold should be lower than 85%.

Equation 2 shows our calculation for the adaptive CPU utilization threshold and Algorithm 3 presents the related policy derived from Algorithm 2 to control the DVFS and specify the suitable frequency according to Figure 1. Lines 8-9 are added in Algorithm 3 to do so.

$$T_{CPU} = T_{Min} + U_{CPU} \times (T_{Max} - T_{Min}) \quad (2)$$

where T_{CPU} is the new CPU per-core threshold, T_{Min} is the minimum threshold of the user space (45%), T_{Max} is the maximum threshold of the user space (85%), and U_{CPU} is the CPU utilization average determined by Equation 3.

$$U_{CPU} = 100\% - \frac{\sum_1^n CPU_{idle}}{n} \quad (3)$$

where n is the core number and CPU_{idle} is the CPU idle time for each core.

Algorithm 3. Adaptive threshold per-core DVFS
Input: the processor idle times and frequencies
Output: the frequency of processor cores
1. **Define** the cpu-frequency list
2. **Define** the policy for the performance governor
3. **While (True)**
4. **Require:** all cores' processor *utilization*
5. **Require:** all cores' processor *frequency*
6. **for** *core* ∈ $Cores_{total}$ **do**
7. **Determine** *average utilization(core)*
8. **Determine** $T_{CPU}(core) = T_{Min} + U_{CPU}(core) \times (T_{Max} - T_{Min})$
9. **If** $utilization(core) < T_{CPU}(core)$ **then**
10. Update to the *maximum Frequency*
11. **Else**
12. Delay down state
13. Lower the *cpu-frequency* for one step
14. **end**
15. **end for**
16. **end while**

Algorithm 3 is our proposed DVFS algorithm. It uses the per-core average utilization from the previous utilization. Also, the new CPU per-core threshold can be derived from Equation 2 as the per-core average utilization in Equation 3 in Lines 8-9. Consequently, Algorithm 3 is called "Adaptive Threshold Per-core DVFS or ATP DVFS" and is to be compared with the default governors including Performance, OnDemand and Conservative on Intel Xeon Quad-cores E5630 as illustrated in the Experimental Methodology section.

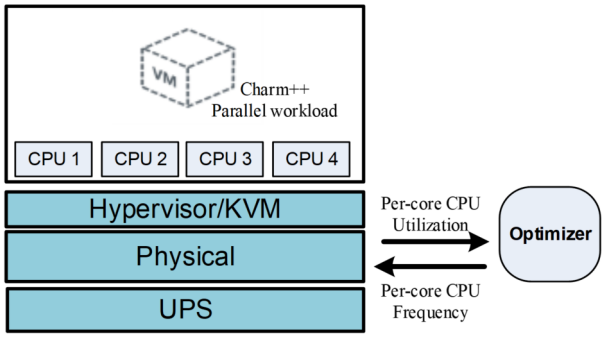
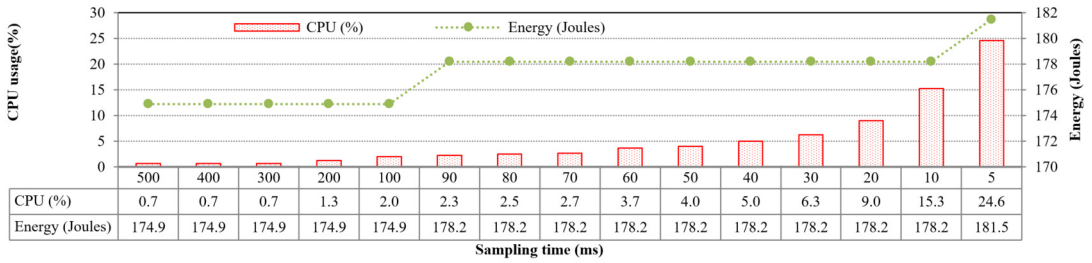
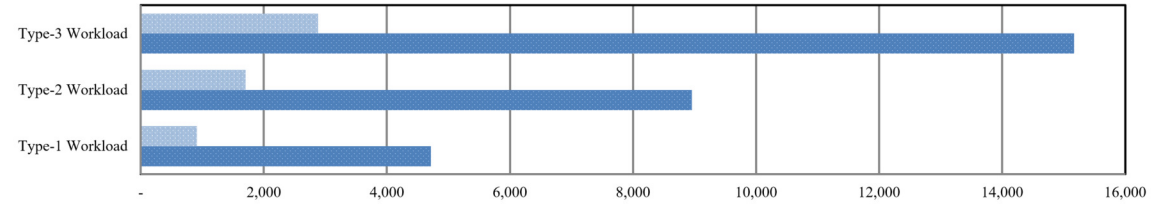


Fig.2: *The research system architecture.*



Sampling time (ms)	500	400	300	200	100	90	80	70	60	50	40	30	20	10	5
CPU (%)	0.7	0.7	0.7	1.3	2.0	2.3	2.5	2.7	3.7	4.0	5.0	6.3	9.0	15.3	24.6
Energy (Joules)	174.9	174.9	174.9	174.9	174.9	178.2	178.2	178.2	178.2	178.2	178.2	178.2	178.2	178.2	181.5

Fig.3: *Energy consumption and CPU utilization with different sampling times.*



	Type-1 Workload	Type-2 Workload	Type-3 Workload
branches M/sec	908.41	1,704.61	2,881.02
instructions (Giga)	4,716.95	8,959.61	15,172.72

Fig.4: *Parallel workload behaviors on different types of workloads.*

5. EXPERIMENTAL SETUP

In order to evaluate the energy efficiency of our proposed algorithms, the per-core DVFS, and the adaptive threshold per-core DVFS, we tested them on the physical rack server with Intel Xeon Quad-cores E5630 2.53GHz, RAM 8 GB, SATA 500 GB, and Gigabit Ethernet. The Power Indicator was the *apcupsd* service, which is a UPS control system for APC UPS with the RJ45 to USB console cable, applying the *apcaccess* command, described in the system architecture subsection. In our experiment, the workload characteristics were parallel workloads of Fine-grained Communication with Topological Routing and Aggregation of Messages called Stream-All-to-All benchmark from the Charm++ Parallel Programming Laboratory [25]. This workload is to be described in the parallel workload subsection.

5.1 System architecture

The target server features Intel Xeon Quad-cores with enabled virtual technology. The quad cores used the same CPU governor, and in the background processes, our DVFS algorithms were running along with the Charm++ parallel workload. In Figure 2, our DVFS algorithm is called an "Optimizer" as it receives the current frequency and utilization of each core and controls the new frequency of each core.

Figure 2 describes the software in our testing system environment running the CentOS 7 64 bits, KVM hypervisor, and *apcupsd* daemon. The Optimizer agent controls the physical layer directly according to each CPU core.

To understand how the proposed DVFS algorithms work, the CPU utilization and energy consumption at each interval are collected and shown in Figure 2. We are interested in the intervals of more than 100 milliseconds which have the CPU utilization lower than 2% and the idle power consumption lower than 174 Joules. These are acceptable costs for the algorithms' overhead.

Figure 3 illustrates the highest idle power consumption, which occurred during a 5-millisecond interval, reaching 181 Joules and over 24% CPU utilization. The frequency and utilization of each CPU core were monitored at specific sampling intervals using Python's *psutil* library, a cross-platform and free software tool for retrieving system utilization data. It is important to note that shorter loops or check intervals contribute to higher energy consumption and CPU usage. More frequent checks increase the CPU's active processing time, preventing it from entering low-power states and thus leading to higher overall power consumption. For our scenario, we selected a 100-millisecond interval to balance performance and power efficiency.

5.2 Parallel workloads

The parallel workloads in our experiment were set up on Charm++, which is a C++-language-based parallel programming system. Charm++ has been designed for HPC programming and supporting processor virtualization. The proposed and designed program runs on the testing environment, including four virtual CPU cores, 8 GB RAM, 100 GB virtual SCSI, and 1 Gigabit Ethernet. There are three levels of testing parallel and continuous workloads according to the number of involved CPU cores: two executing cores (Type-1 workload), three executing cores

(Type-2 workload), and four executing cores (Type-3 workload). These workloads apply the topological virtual network routing called the Topological Routing and Aggregation of Messages (TRAM). Also, the parallel workloads can be divided into two patterns: branches and instructions [19].

In addition, the branch pattern, or the branch tracer, is an execution path recorder tool that is supported by modern processors and is available for Pentium 4, Xeon, and Intel processors [20]. It is called the "HW-based branch tracer" [20]. Figure 4 reveals the behavior of such parallel workloads in terms of instructions and branches. In the experiment, parallel workloads consist of the three behaviors (Type-1, Type-2, and Type-3).

Figure 4 illustrates the parallel workloads, including Type-1, Type-2, and Type-3 workloads. Among them, the Type-3 workload has the largest problem size, containing 15,172.72 Giga-instructions and 2,881.02 branches. The Type-2 workload is approximately half the size of the Type-3 workload, with 8,959.61 Giga-instructions and 1,704.61 branches. Meanwhile, the Type-1 workload has the smallest problem size, accounting for only one-third of the Type-3 workload, with 4,716.95 Giga-instructions and 908.41 branches. Figures 5 to 7 present the average CPU usage percentage for each parallel workload, highlighting their respective workload characteristics.

There are three levels of testing parallel and continuous workloads based on the number of CPU cores involved: two executing cores (Type-1 workload), three executing cores (Type-2 workload), and four executing cores (Type-3 workload). Figures 5-7 show the CPU usage for each workload type. Despite the increasing workload sizes, CPU usage remains relatively stable across all types. Specifically, Figure 5 illustrates the CPU usage for the Type-1 workload with two executing cores, Figure 6 shows the usage for the Type-2 workload with three executing cores, and Figure 7 presents the usage for the Type-3 workload with four executing cores. In all cases, Charm++ efficiently balances the load across the cores, preventing any single core from becoming overloaded and ensuring consistent CPU utilization, even when the number of cores and workload size increases.

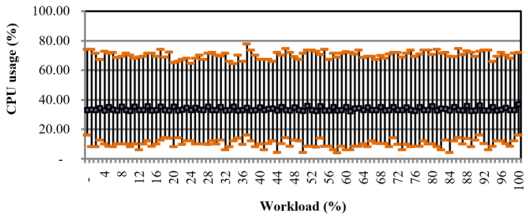


Fig.5: *The CPU usage of the Type-1 Workload.*

Figure 5 illustrates the Type-1 workload, where the average CPU usage is approximately 34%. This parallel workload runs on only two cores, alternating between them across the four virtual CPU cores of the testbed virtual machine, representing small and randomized workloads.

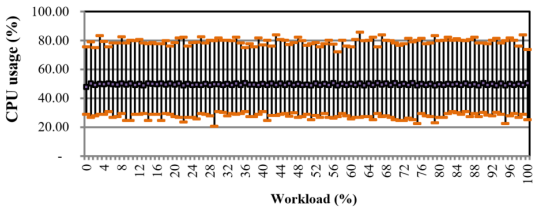


Fig.6: *The CPU usage of the Type-2 Workload.*

Figure 6 illustrates the Type-2 workload, where the average CPU core usage is approximately 49%. This parallel workload runs on three cores of the testbed virtual machine, which has four virtual CPU cores, representing an unbalanced, medium-sized workload.

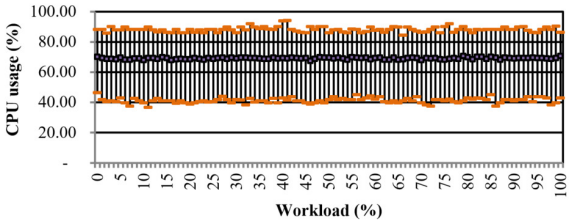


Fig.7: *The CPU usage of the Type-3 Workload.*

Figure 7 illustrates the Type-3 workload, where the average CPU usage is approximately 69%. This parallel workload runs on four cores of the testbed virtual machine, representing a balanced and large workload.

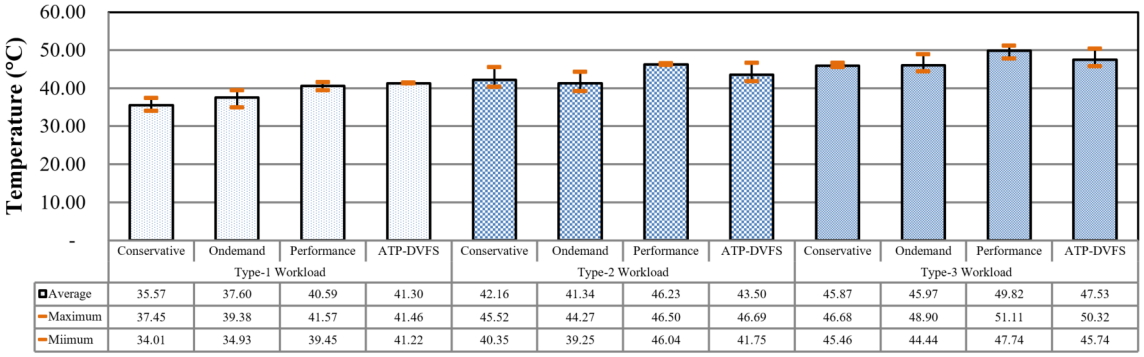
6. EXPERIMENTAL RESULTS AND DISCUSSION

In order to validate our proposed DVFS algorithm, ATP DVFS, we have set up three parallel workload scenarios from the mash streamer prototype which simulates fine-grained messages with virtual topologies, including four-core, three-core, and two-core parallel workloads.

The experimental results show the differences in the CPU core temperatures on the three types of parallel workloads. Section 6.1 shows the policy effects on the environment and Section 6.2 illustrates the energy consumption versus the performance in the Joules energy efficiency per Giga- instructions (10^9) and the counter for accessed branches in Mega (10^6) times per second.

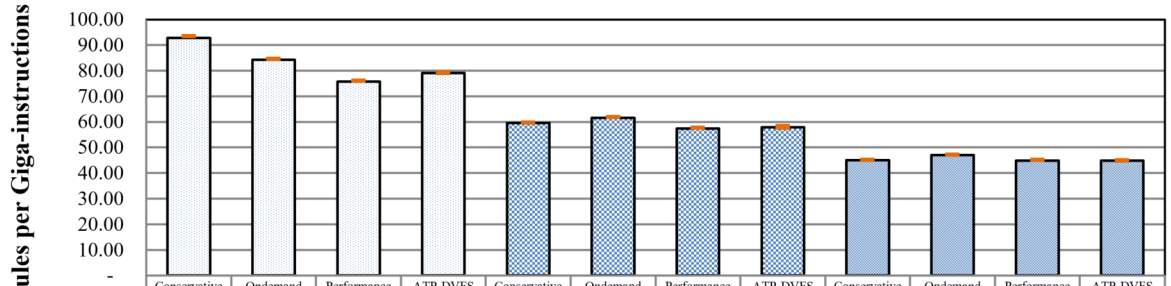
6.1 Thermal performance

This section presents the thermal performance in the parallel workload execution process with different governors. In addition, the CPU core frequency



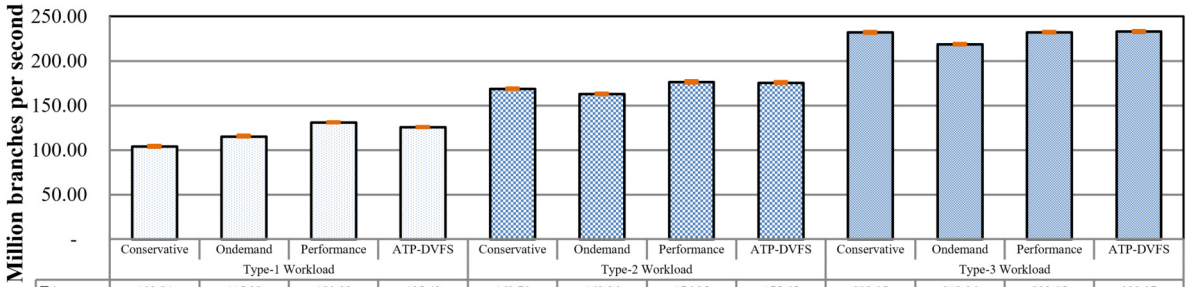
	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS
	Type-1 Workload				Type-2 Workload				Type-3 Workload			
Average	35.57	37.60	40.59	41.30	42.16	41.34	46.23	43.50	45.87	45.97	49.82	47.53
Maximum	37.45	39.38	41.57	41.46	45.52	44.27	46.50	46.69	46.68	48.90	51.11	50.32
Miimum	34.01	34.93	39.45	41.22	40.35	39.25	46.04	41.75	45.46	44.44	47.74	45.74

Fig.8: *The average CPU temperatures when applying different governor policies.*



	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS
	Type-1 Workload				Type-2 Workload				Type-3 Workload			
Average	92.85	84.29	75.74	79.06	59.56	61.64	57.45	57.88	44.95	46.96	44.92	44.81
Maximum	93.40	84.50	75.94	79.11	59.60	61.78	57.61	58.35	45.04	47.08	45.06	44.94
Miimum	93.40	84.50	75.94	78.98	59.60	61.78	57.61	57.59	45.04	47.08	45.06	44.67

Fig.9: *The average energy consumption in Joules per Giga-instructions (10^9) when applying different governor policies.*



	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS	Conservative	Ondemand	Performance	ATP-DVFS
	Type-1 Workload				Type-2 Workload				Type-3 Workload			
Average	103.84	115.28	130.98	125.43	168.70	162.86	176.35	175.65	232.05	218.86	232.05	232.97
Maximum	104.36	115.86	131.08	125.45	168.86	163.10	177.07	176.06	232.60	219.07	232.38	233.57
Miimum	103.42	114.90	130.85	125.41	168.55	162.63	175.64	175.14	231.58	218.63	231.81	232.67

Fig.10: *The average accessed branches per second and when applying different governor policies.*

control techniques have more different control parameters, including the temperature [2], Service Level Agreements (SLA) [5], [7] and the deadline of the parallel task [6]. In this experiment, we focus on the temperature of the CPU core and show the average CPU core temperature results of the power management policies in Figure 8. The Performance governor, using the highest CPU frequency on each core, gains the highest CPU temperature than the other governors, including the OnDemand, and Conservative governors. The Adaptive DVFS, the proposed algorithm, can keep a lower CPU temperature in some workloads.

Figure 8 does not only show the average CPU temperatures during the execution time but also presents the maximum and minimum CPU temperatures on each governor policy. The average CPU temperature during the execution time applying different governor policies is illustrated in Figure 8. It shows the comparison of thermal performance on each power management policy applying three parallel workload types. In the Type-1 workload, the Conservative governor's temperature was lower than the others at 35.57 °C. Also, ATP DVFS performed similarly to the Performance governor at 41.30 °C. In the Type-2 workload, the OnDemand governor shows the best thermal performance at 41.34 °C and the ATP DVFS temperature was lower than that of the Performance

governor at 43.50 °C with an unbalanced workload and medium workload.

The Performance governor's temperature was higher than that of the ATP DVFS which is close to 3°C. The Type-3 workload had a similar average temperature to those of Conservative and OnDemand. Also, the Performance governor still gained the highest temperature compared to the other standard governors and ATP DVFS. ATP DVFS's average temperature was lower than 2.29 °C. The thermal performance of the three load types is only presented in terms of temperature rather than energy consumption. Other performance factors are presented in the next section.

6.2 Performance and energy consumption

This part illustrates other performances in terms of energy consumption and execution time. The main goal of the proposed power management policy is to reduce the energy consumption demonstrated on the Charm++ parallel workload. The average energy consumption results are determined from the total energy consumption in Joules divided by the number of instructions. The performance results are kept in the output file of the *perf-kvm* tool [26]. Which runs on the host machine. Moreover, Figure 9 shows energy consumption by Joules per Giga-instructions on a virtual machine. Figure 10 shows the execution performance by the access branch rate per second. In addition, both figures represent the execution performance with the different parallel workload types.

Figure 9 illustrates the three groups of parallel workloads, including Type-1, Type-2, and Type-3. The y-axis shows the average total energy consumption in Joules. These energy consumptions are divided by the number of total instructions during their execution time. For the Type-1 workload, the Performance governor gained the best energy consumption performance at 75.74 Joules per Giga-instructions. ATP DVFS was second at 79.06 Joules per Giga-instructions. The third one was OnDemand at 84.29 Joules per Giga-instructions and Conservative consumed at 92.85 Joules per Giga-instructions. Moreover, the Type-2 workload results gained the best energy consumption performance for the Performance governor at 57.48 Joules per Giga-instructions and it had quite similar energy consumption performance to ATP DVFS at 57.88 Joules per Giga-instructions. On the other hand, the Conservative governor consumed 59.56 Joules per Giga-instructions which was lower than that of the OnDemand governor at 61.64 Joules per Giga-instructions.

The Type-3 workload consumed quite similar energy to those of the Performance governor at 44.92 Joules per Giga-instructions and ATP DVFS at 44.81 Joules per Giga-instructions.

Figure 10 shows the access branch performance for the same parallel workload types which represents the event control of CPU cores with different governor policies. The access event control performance of the CPU applying different governor policies is illustrated in Figure 10. The Type-1 workload gained the highest access branch rate for the Performance governor at 130.98 Million branches per second. The second was ATP DVFS at 125.43 Million branches per second. The third is OnDemand at 115.28 Million branches per second and the last one is Conservative at 103.84 Million branches per second.

The Type-2 workload is a medium parallel workload and it gained the highest branch rate with the Performance governor at 176.35 Million branches per second which is quite like that of ATP DVFS at 175.65 Million branches per second. The OnDemand governor was the last one at 162.86 Million branches per second and the Conservative governor was the third at 168.70 Million branches per second.

The Type-3 workload was the highest balanced parallel workload. It had both best access branch rates performance with the Performance governor at 232.05 Million branches per second and ATP DVFS at 232.97 Million branches per second. The third was Conservative at 232.05 Million branches per second and OnDemand was the last one at 218.86 Million branches per second.

Both parallel workloads, Type 2 and Type 3, had quite similar access branch rates in between those of Performance and ATP DVFS but only for the Type-1 workload, ATP DVFS' branch rate was lower than that of Performance governor. The Conservative governor gained better access branch rate performance than those of the OnDemand in the Type-2 and Type-3 workloads. However, the Type-1 workload was the only parallel workload type that had Conservative's access branch rate lower than the OnDemand's.

6.3 Discussion

The paper presents ATP Dynamic Voltage and Frequency Scaling (DVFS) as an energy-efficient policy that utilizes dynamic per-core CPU threshold management. Unlike traditional static threshold policies, such as those used by the OnDemand governor, ATP DVFS is based on a user-space threshold range of 45% to 85%, as outlined in Algorithm 3. This dynamic approach aims to optimize both energy consumption and thermal performance, positioning ATP DVFS as a promising candidate for green governors that balance energy efficiency, temperature control, and system performance.

The performance evaluation in this paper focuses on two main aspects: thermal performance and energy consumption. These key performance measures are applied to parallel workloads from the Charm++ parallel programming framework, using TRAM workloads. The workloads tested include the following scenarios: Type-1 (2 execution cores, random, small), Type-2 (3 execution cores, unbalanced, medium), and

Type-3 (4 execution cores, balanced, large).

The thermal results show that ATP DVFS behaves similarly to the Performance governor for the Type-1 workload. However, for the Type-2 and Type-3 workloads, ATP DVFS produced better thermal results than the Performance governor, though it was still higher than other default governors such as OnDemand and Conservative.

ATP DVFS demonstrates significant contributions to both energy efficiency and temperature reduction, making it a strong candidate for integration into green governors. For the Type-1 workload, ATP DVFS achieved an energy efficiency of 79.06 Joules per Giga-instructions, outperforming the OnDemand governor (84.29 Joules per Giga-instructions). Additionally, ATP DVFS attained a higher branch access rate compared to OnDemand (10.15 million branches per second) and the Conservative governor (21.59 million branches per second). For the Type-2 and Type-3 workloads, ATP DVFS showed performance and energy efficiency comparable to the Performance governor, while maintaining a higher branch access rate than both OnDemand and Conservative governors. Moreover, ATP DVFS led to an average CPU temperature reduction of approximately 5%, or 2.5°C.

These results highlight that ATP DVFS does not only offer energy savings compared to default power-saving governors but also performs similarly to the Performance governor. It provides a balanced approach to energy efficiency, temperature management, and performance. Unlike the green governor approach in [10], which focuses on CPUs with varying idle power consumption, ATP DVFS ensures consistent power consumption across all frequencies, particularly for Intel Xeon processors. This further supports its potential for efficient virtual machine workload management.

7. CONCLUSION

This work investigates DVFS (Dynamic Voltage and Frequency Scaling) policies and proposes a new algorithm based on the Linux governor. It does not only achieve superior performance and energy efficiency per Giga-instructions compared to the Conservative and OnDemand governors but also maintains performance levels close to those of the Performance governor. The algorithm dynamically adjusts the DVFS threshold at the Linux kernel layer and optimizes the frequency of each active CPU core, ensuring an efficient balance between performance, resource utilization, and energy consumption. This study presents a real-time control mechanism and evaluates the algorithm using virtualization technology on physical servers running Charm++ parallel workloads. The adaptive set point, or threshold, is determined based on CPU core utilization.

Experimental results show that the proposed algorithm outperforms both the OnDemand and Conservative governors across all parallel workload types. However, for Type-1 workloads, where average CPU usage is below 40% (small parallel workloads), its performance is slightly lower than that of the Performance governor. Nonetheless, for medium and large parallel workloads, the proposed DVFS algorithm maintains lower thermal levels than the Performance governor. This demonstrates its effectiveness in handling heavy parallel workloads, keeping performance close to the Performance governor while reducing CPU temperature.

The per-core adaptive threshold DVFS maintains lower CPU temperatures than the default Linux governors (OnDemand and Conservative) when parallel workloads exceed 50%. Moreover, its energy consumption per instruction is comparable to that of the Performance governor and more efficient than the OnDemand and Conservative governors in virtualized environments.

The proposed DVFS algorithm is beneficial for virtual machines running on a hypervisor and utilizing virtualization technology. Future research should explore its deployment and effectiveness in cloud container architectures.

ACKNOWLEDGEMENT

The authors would like to thank the Prince of Songkla University for the Graduate Study Scholarship.

AUTHOR CONTRIBUTIONS

Conceptualization, K.R. and P.T.; methodology, K.R.; software, K.R.; validation, K.R.; investigation, K.R.; writing—original draft preparation, K.R. and P.T.; writing—review and editing, K.R. and P.T.; supervision, P.T.; funding acquisition, K.R. and P.T.; All authors have read and agreed to the published version of the manuscript.

References

[1] I. Hamzaoui, B. Duthil, V. Courboulay and H. Medromi, “A Survey on the Current Challenges of Energy-Efficient Cloud Resources Management,” *SN Computer Science*, vol. 1, no. 73, 2020.

[2] O. Sarood, P. Miller, E. Totoni and L. V. Kalé, “ “Cool“ Load Balancing for High Performance Computing Data Centers,” in *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1752-1764, Dec. 2012.

[3] M. Shojafar, N. Cordeschi and E. Baccarelli, “Energy-Efficient Adaptive Resource Management for Real-Time Vehicular Cloud Services,” in *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 196-209, 1 Jan.-March 2019.

[4] K. Gupta and V. Katiyar, “Energy-Aware Scheduling Framework for resource allocation in

a virtualized cloud data centre," in *International Journal of Engineering and Technology*, vol. 9, no. 2, pp. 558-563, Apr. 2017.

[5] C. -C. Lin, J. -J. Chen, P. Liu and J. -J. Wu, "Energy-Efficient Core Allocation and Deployment for Container-Based Virtualization," *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Singapore, pp. 93-101, 2018.

[6] Y. Hao, J. Cao, T. Ma and S. Ji, "Adaptive energy-aware scheduling method in a meteorological cloud," *Future Generation Computer Systems*, vol. 101, 1142-1157, 2019.

[7] P. J. Kuehn and M. Mashaly, "DVFS-Power Management and Performance Engineering of Data Center Server Clusters," *2019 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Wengen, Switzerland, pp. 91-98, 2019.

[8] A. Beloglazov, R. Buyya, Y. C. Lee and A. Zomaya, "Chapter 3 - A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," *Advances in Computers*, vol. 82, pp 47 – 111, 2011.

[9] V. Pallipadi and A. Starikovskiy, "The OnDemand governor: past, present and future," *Proceedings of Linux Symposium*, vol. 2, pp. 223 – 238, 2006.

[10] V. Spiliopoulos, S. Kaxiras and G. Keramidas, "Green governors: A framework for Continuously Adaptive DVFS," *2011 International Green Computing Conference and Workshops*, Orlando, FL, USA, pp. 1-8, 2011.

[11] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar, "CPU frequency and voltage scaling code in the Linux (TM) kernel," *Linux Kernel Documentation*, vol. 66, 2013.

[12] D. Huang, L. Costero and D. Atienza, "Is the powersave governor really saving power?," *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Philadelphia, PA, USA, pp. 273-283, 2024.

[13] J. Stoess, C. Lang and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *Proc. 2007 USENIX Annual Technical Conference*, pp. 1–14, 2007.

[14] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 2–13, 2006.

[15] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 195–237, 2005.

[16] J. Stoess, C. Lang, and F. Bellosa, "Energy Management for Hypervisor-Based Virtual Machines," *The 2007 USENIX Annual Technical Conference*, pp. 1 – 14, 2007.

[17] B. Acun, K. Chandrasekar and L. V. Kale, "Fine-grained energy efficiency using per-core DVFS with an adaptive runtime system," in *Proc. 2019 Tenth Int. Green and Sustainable Computing Conf. (IGSC)*, pp. 1–8, 2019.

[18] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P. Östberg and E. Elmroth, "Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling," *Future Generation Computer Systems*, vol. 81, pp. 14–128, 2018.

[19] M. Getka and M. Karpowicz, "Fixed-point self-tuning CPU performance controller for Linux kernel," in *Proc. 2019 Int. Conf. High Performance Computing & Simulation (HPCS)*, Dublin, Ireland, pp. 470–477, 2019.

[20] A. Tzenetopoulos, D. Masouros, S. Xydis and D. Soudris, "Leveraging core and uncore frequency scaling for power-efficient serverless workflows," *arXiv preprint arXiv:2407.18386*, 2024.

[21] H. Kumar, N. Chawla S. Mukhopadhyay, "A DVFS based exploit to undermine resource allocation fairness in linux platforms," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020.

[22] D. Huang, L. Costero and D. Atienza, "Is the powersave governor really saving power?," *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Philadelphia, PA, USA, pp. 273-283, 2024.

[23] M. Bambagini, M. Marinoni, H. Aydin and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embedded Computing Systems (TECS)*, vol. 15, no. 1, pp. 1–34, 2016.

[24] Z. Bellal, L. Lahlou, N. Kara and I. El Khayat, "GAS: DVFS-Driven Energy Efficiency Approach for Latency-Guaranteed Edge Computing Microservices," in *IEEE Transactions on Green Communications and Networking*, vol. 9, no. 1, pp. 108-124, Mar. 2025.

[25] P. Miller, "Productive Parallel Programming with CHARM++," *Proceedings of the Symposium on High Performance Computing*, pp. 241–242, 2015.

[26] A. R. Ghods, "A Study of Linux Perf and Slab Allocation Sub-Systems," UWSpace. 2016. `http://hdl.handle.net/10012/10184`, (accessed Feb. 2021).

Kritwara Rattanaopas received his B.Eng. in Electrical Engineering from Prince of Songkla University (PSU), Thailand in 1998. He later earned his M.Eng. in Electrical Engineering in 2002 and an M.Sci. in Management Information Technology in 2006, both from PSU. In 2021, he obtained his Ph.D. in Computer Engineering from PSU. Currently, he is a Lecturer at the Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University, Hatyai. His research interests and contributions focus on high-performance computing, cloud computing, artificial intelligence, and data science.

Pichaya Tandayya graduated in Electrical Engineering (Communications) from Prince of Songkla University (PSU), Thailand in 1990. In 2001, She obtained her Ph.D. in Computer Science from the University of Manchester in the area of Distributed Interactive Simulation. Currently, she is an Associate Professor working at the Department of Computer Engineering, PSU. Her research works involve Parallel and Distributed Systems, High-performance Computing, Cloud Computing, Artificial Intelligence, and Assistive Technology.