A Framework for Carbon-aware Real-Time Workload Management in Clouds using Renewables-driven Cores

Tharindu B. Hewage, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya

***Abstract*—Cloud platforms commonly exploit workload temporal flexibility to reduce their carbon emissions. They suspend/resume workload execution for when and where the energy is greenest. However, increasingly prevalent delay-intolerant real-time workloads challenge this approach. To this end, we present a framework to harvest green renewable energy for real-time workloads in cloud systems. We use *Renewables-driven cores* in servers to dynamically switch CPU cores between real-time and low-power profiles, matching renewable energy availability. We then develop a VM Execution Model to guarantee running VMs are allocated with cores in the *real-time power profile*. If such cores are insufficient, we conduct criticality-aware VM evictions as needed. Furthermore, we develop a VM Packing Algorithm to utilize available cores across the servers. We introduce the *Green Cores* concept in our algorithm to convert renewable energy usage into a server inventory attribute. Based on this, we jointly optimize for renewable energy utilization and reduction of VM eviction incidents. We implement a prototype of our framework in OpenStack as *openstack-gc*. Using an experimental *openstack-gc* cloud and a large-scale simulation testbed, we expose our framework to VMs running RTEval, a real-time evaluation program, and a 14-day Azure VM arrival trace. Our results show: i) a $6.52\times$ reduction in coefficient of variation of real-time latency over an existing workload temporal flexibility-based solution, and ii) a joint 79.64% reduction in eviction incidents with a 34.83% increase in energy harvest over the state-of-the-art packing algorithms. We open source *openstack-gc* at https://github.com/tharindu-b-hewage/openstack-gc.**

***Index Terms*—Carbon-aware computing, real-time, cloud computing, sustainability, renewable energy.**

I. Introduction

Data centers consumed approximately 1-1.3% of global electricity demand in 2022 [1]. Between 2015 and 2022, data center energy usage increased by 20-70% [1]. The recent unprecedented compute demand due to Artificial Intelligence (AI) and Machine Learning (ML) workloads indicates that this trend will continue to grow [2]. Data centers often connect to electricity grids with shares of energy generation based on fossil fuels. As a result, in 2020, data centers were responsible for 0.9% of energy-related greenhouse gas (GHG) emissions [1]. Climate crisis-driven road maps necessitate that data center emissions drop by half by 2030 to meet global Net Zero Emissions goals [1], [2].

In response to GHG emissions, electrical grids continue to integrate low-emission renewable energy sources. In 2022, the share of renewables in total electricity generation was 39% and is projected to be 91% by 2035 [3]. However, growing variable-availability (intermittent) renewable energy sources, such as solar and wind, challenge electrical grids [2]. Between 2022 and 2035, energy reports project the share of solar and wind renewables in total generation to rise from 12% to 58% [3].

Data centers develop various *load matching* strategies to match workload execution over intermittent renewable energy. Amongst them, *load shifting* is commonly practised [4]–[7]. *Load shifting* uses workloads with temporal flexibility to suspend/resume their execution. For example, Google's delay-tolerant workloads, such as machine learning, data compaction, and data processing, tolerate delays as long as their work gets completed within 24 hours [6]. Workloads execute in periods when renewable energy capacity is higher, resulting in reduced GHG emissions. However, *load shifting* falls short when applied to real-time workloads with strict response time boundaries [8]. Real-time workloads cannot tolerate the delays inherent in *load shifting*.

Nevertheless, the growing prevalence of real-time cloud applications, such as autonomous vehicles, industrial automation [9], and railway control systems [10] expects to account for nearly 30% of the world data by 2025 [11]. As a result, cloud operators will eventually have to incorporate growing real-time workloads in intermittent renewable energy integration. In this context, one must find an alternative *load matching* strategy to *load shifting* for delay-intolerant real-time workloads. Existing solutions, such as applying CPU-wide *low power profile* to match renewable energy supply [12], often result in increased latency, making them unsuitable for real-time applications. Moreover, techniques like Harvest Virtual Machines (HVMs), which allow uninterrupted execution of workloads with reduced resources [13], can still degrade performance and fail to meet real-time constraints.

Given these challenges, there is a need for an efficient strategy to integrate renewable energy into real-time cloud systems (*Real-Time Clouds*). To this end, we propose a framework to harvest renewable energy in *Real-Time Clouds*. We use *Renewables-driven cores* to integrate renewable energy for servers. It dynamically switches the power profiles of each CPU core between a *real-time power profile* and a *low power profile* to match renewable energy intermittency. Then, our framework applies a twofold solution to utilize this dynamic core availability. First, we develop a **VM Execution Model** to guarantee that real-time virtual machines (VMs) occupy cores at the *real-time power profile*. Our model adopts renewable energy fluctuations by conducting criticality-aware VM evictions as needed. Secondly, we develop a **VM Packing Algorithm** to optimize the use of available cores across servers. It reduces the likelihood of VM evictions while maximizing renewable energy utilization (renewable energy harvest). Our algorithm

T. B. Hewage, M. A. Rodriguez, R. Buyya are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Parkville, VIC 3010, Australia.

S. Ilager is with the Informatics Institute, University of Amsterdam.

frames renewable energy management as a VM placement optimization problem by introducing the concept of ***Green Cores***. *Green Cores* presents each server as an inventory of two virtual CPU core types: Green and Regular. Green cores quantify renewable energy usage, whereas Regular cores quantify core usage that does not increase risks of VM eviction incidents. Using *Green Cores*, we achieve a computationally inexpensive VM packing algorithm, which is required to handle VM throughput at scale [6].

We implement our framework in OpenStack [14] as *openstack-gc*. We combine OpenStack's control plane with an on-node daemon service. The daemon service implements *Renewables-driven cores* in the server using per-core sleep states. *openstack-gc*'s control plane then communicates with the daemon service to orchestrate our VM Execution Model and VM Packing Algorithm. We evaluate our framework at the core-level using VMs running RTEval, a program from the Real-Time Linux project to measure real-time performance [15]. We evaluate our framework at the server-level using a 14-day VM arrival trace from Azure [16]. We use two testbeds: an experimental *openstack-gc* cloud deployed on an HPE ProLiant server with a 12-core Intel Xeon CPU, and a large-scale simulation testbed. We make the following contributions in designing, implementing, and evaluating our framework.

- We propose a core-level **VM Execution Model** to utilize renewable energy without degrading real-time latency performance in VMs. We leverage criticality-aware VM evictions for that.
- We propose a server-level **VM Packing Algorithm** to reduce VM eviction incidents over renewable energy utilization.
- We implement a prototype of our framework in OpenStack, detailing its design and demonstrating its practicality.
- We evaluate our approach against multiple baselines. Our results show: i) $6.52\times$ reduction in coefficient of variation of real-time latency in VMs over the existing workload temporal flexibility-based VM execution model, and ii) a joint optimization of 79.64% reduction in VM eviction incidents and 34.83% increase in utilized renewable energy over state-of-the-art packing algorithms [17].

The rest of the paper is organized as follows: Section II provides the background and motivation for our problem with a use case study. Section III details our system model and problem formulation. Section V outlines the design of our proposed framework. Section VI describes the implementation of *openstack-gc*. Section VII presents the performance evaluation of our framework. Section VIII discusses related work, and Section IX concludes the paper and outlines future work.

II. Background, Motivation, and Use case

This section provides background on real-time workloads in clouds (*Real-Time Clouds*) and the application of *Renewables-driven cores*. It then motivates our contributions with a use case study. Finally, it outlines the key takeaways.

A. Real-Time Clouds

Real-Time Clouds deploy cloud-based real-time applications, such as industry 4.0 use cases [18], transport use cases [10], and software-defined networks [19]. A key requirement in real-time computing is to produce computation results in a bounded time [18]. Clouds achieve that by tuning the entire virtualization stack to reduce latency in executing application instructions [10], [20]–[22], such as setting each CPU core to a consistent high-performance power profile and pinning each virtual machine (VM) core to a dedicated physical core, resulting rigid VM placement constraints while delivering deterministic performance. Further, real-time cloud systems employ an application-specific middleware layer over VMs to provide fault-tolerance in VM failures [10], [19].

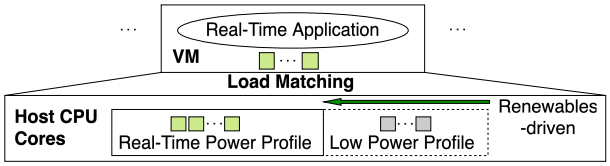

Fig. 1. *Renewables-driven cores* in *Real-Time Clouds*

B. Renewables-driven Cores

Renewables-driven cores is a load-matching technique that dynamically adjusts per-core power draw in CPU to match server load for renewable energy dynamics [23], [24]. Unlike the CPU-wide throttling techniques [17], it narrows power optimization to the core level. However, such per-core power dynamics must be efficiently utilized, adhering to application performance requirements using a suitable workload execution model. Existing works that utilize *Renewables-driven cores* in clouds use Harvest Virtual Machine (HVM) as the workload execution model [25], which dynamically shares available CPU cores among the VM cores.

C. Motivation

We outline the motivation behind our proposed framework, specifically focusing on the rationale for selecting *Renewables-driven cores* as the load-matching technique for *Real-Time Clouds*. Then, we discuss the lack of static compute allocation in the existing VM execution solution for *Renewables-driven cores* and how it can impact real-time VMs. Following this, we present our approach to addressing this limitation by exploiting the presence of mixed-criticality in *Real-Time Clouds*, composed of VMs hosting *critical* components and *best-effort* components, to implement criticality-aware VM evictions.

How does Renewables-driven cores align with the needs of Real-Time Clouds? Renewable energy integration using *Renewables-driven cores* as the load-matching technique enables avoiding both suspend/resume and CPU-wide throttling of workloads. Figure 1 illustrates a scenario where a set of cores reside in the *low power profile*. However, the remaining cores reside in the *real-time power profile*. They provide the opportunity to serve real-time VMs amidst renewable energy fluctuations.

Why is a static compute allocation important to real-time VMs?: To apply *Renewables-driven cores* in *Real-Time Clouds*, we need a workload execution model to match dynamic core availability for VMs. In this regard, the existing solution is Harvest VMs (HVMs) [13]. HVM's approach is to change the number of physical cores (pCPUs) in the VM but preserve the number of virtual cores (vCPUs). It allows con-

TABLE I
MIXED-CRITICALITY USE CASES IN 5G NETWORK SLICING [9]

Scenario	Reliability	Criticality
Autonomous Driving	99.999%	Critical
Industrial Machinery	99.999%	Critical
4K/8K HD Video	-	Best-effort
Mass Gathering	-	Best-effort

tinued execution of the VM amidst dynamic core availability. However, for real-time VMs, this dynamic compute allocation can introduce performance degradation. For instance, if the number of pCPUs is less than that of vCPUs, vCPUs oversubscribe physical cores, leading to scheduling delays, which must be avoided with real-time compute [21]. Therefore, insufficient pCPU allocations can lead to undesirable real-time latency spikes in the VMs. Maintaining a static compute allocation is important to avoid such scenarios. In Section VII-B, we practically show the VM's real-time performance degradation when the number of pCPUs falls below vCPUs.

Opportunities in mixed-criticality within Real-Time Clouds to provide static compute allocation for VMs over Renewables-driven cores: An alternative to the HVM approach of continuing the execution of VMs with insufficient pCPUs is to evict the VMs. Existing works show opportunities for this in *Real-Time Clouds* via the mixed-criticality of real-time systems [10], [19], [26]. Firstly, they model real-time system components as either *critical* or *best-effort*. Then, an application-specific middleware layer exploits this mixed-criticality to provide fault tolerance for component failures. In real-time cloud systems, application-specific middleware layers use reconfiguration policies to recover the system upon VM failures [19]. In this context, there is an opportunity to conduct VM evictions in *Real-Time Clouds* safely. Since a VM eviction is a well-defined failure event, the application-specific middleware layer can tolerate it through reconfiguration. More importantly, we can guarantee a static compute allocation for VMs with a fixed allocation of CPU cores and conduct criticality-aware VM eviction if cores are insufficient instead of continued VM execution with degraded performance.

D. Usecase

To further motivate our approach, we experiment with a real-time cloud use case of 5G Network Slicing via Virtual Network Functions (VNFs) [9]. As the application-specific middleware layer, we employ the production-grade VNF management and orchestration middleware, OSM MANO [19]. We map VNFs to *critical* and *best-effort* components based on the service quality level of their network slice. Table I denotes an example where the criticality of four different 5G scenarios is interpreted based on service reliability. Figure 2 illustrates our study. We connect the MANO deployment with a real-time tuned two-node OpenStack deployment as the real-time cloud. We use the auto-heal feature of MANO as the reconfiguration policy [19]. Each VNF is deployed as a VM in OpenStack with two virtual CPU cores. We use 50% *Renewables-driven cores* in servers at 100% initial renewable energy capacity. Once the deployment stabilizes, we drop that to 0%, reducing server core count by half and evicting VMs to load match. We repeat the experiment for two VM scheduling approaches in OpenStack.

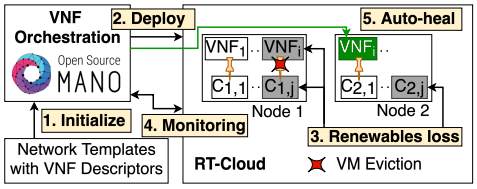

Fig. 2. Use case: understanding the effect of load matching with evictions via application-level reconfiguration. 5G network slicing prototype of open source NFV Management and Orchestration (MANO) with OpenStack as the virtualized infrastructure provider (i.e. real-time cloud provider). MANO's auto-healing feature facilitates application-level reconfiguration over VM failures [19].

TABLE II
COMPARISON OF THE VM PACKING INVENTORY OVER DIFFERENT PACKING STRATEGIES AS 5G NETWORK SLICING PROTOTYPE CONDUCT LOAD MATCHING FOR RENEWABLE ENERGY LOSS VIA VM EVICTIONS.

Packing	Before		After		Evictions
	Node 1	Node 2	Node 1	Node 2	
Tightly	0/8	8/8	4/4	4/4	2
Spread	4/8	4/8	4/4	4/4	0

Table II denotes our observations. Firstly, upon the loss of renewable energy capacity, MANO reconfigured the available cores through auto-healing. Secondly, OpenStack initiates VM evictions. Thus, knowledge of VM criticality is beneficial in reducing the impact of eviction. For example, it permits evicting *best-effort* components prior to *critical* components. Thirdly, the VM packing strategy can change the number of VM evictions. Of the two packing approaches used in our use case, the Tightly approach triggered two eviction incidents, whereas the spreading approach yielded none.

E. Key Takeaways

From our motivations and the use case experiment, we identify the following key takeaways:

1) *Renewables-driven cores* enable integrating renewable energy in *Real-Time Clouds*. In that,
   a) A static compute allocation for VMs ensures their real-time performance.
   b) Criticality-aware VM evictions enable maintaining static compute allocations over *Renewables-driven cores*.
2) The server-level VM packing strategy can influence the likelihood of VM eviction incidents.

Motivated by the above, we design our framework to apply *Renewables-driven cores* in *Real-Time Clouds*. It addresses the point 1 using a VM Execution Model and the point 2 using a VM Packing Algorithm. It advances cloud renewable energy integration by providing deterministic computing amidst the supply intermittency of renewable energy. It enables carbon-efficient computing with time-critical real-time cloud workloads, which is otherwise considered inflexible for carbon optimization [6].

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

Our system model is shown in Figure 3. In that, each server receives dedicated allocations of grid and renewable energy capacities through a mixed power delivery system. Allocations are even across all servers. We use homogeneous servers for

simplicity, but the model can be adapted for heterogeneous servers by allocating power capacities proportionately. Each server monitors the dynamic availability of its allocated renewable capacity for *load matching*. We model renewable energy as intermittent and carbon-free and grid energy as static and carbon-intensive. An application-specific middleware layer manages each VM. It can tolerate VM eviction incidents. At arrival, VMs provide their criticality to the cloud control plane as either *critical* or *best-effort*. A VM packing algorithm then places VMs in the server inventory.

B. Problem Formulation

Server power modeling: Data center power modeling outlines key load elements, such as information and technology (IT), cooling, and internal power conditioning system [27]. IT load corresponds to server power consumption of active VM execution and idle power draw, which can be modeled encompassing the power consumption of server components such as CPU, Memory, GPU and HDD [28], [29]. Our focus in this paper is integrating intermittent renewable energy with the IT load for CPU-dominant real-time workloads [10], [21]. In that regard, server power can be estimated using a linear function (f) of CPU power ($P_{CPU}(t)$) [29] with over 90% accuracy. Based on that, we derive a CPU utilization-aware linear multi-piece server power model for the server power usage of the power distribution unit (PDU). First, we state server power at time t ($P_S(t)$) as,

$$P_S(t) = f(P_{CPU}(t))$$

In multi-core CPUs, the cumulative sum of core power becomes a close upper bound of $P_{CPU}(t)$ [30]. Based on that, we derive an upper-bound to server power and use that as an estimate to $P_{CPU}(t)$. Therefore, for a server with N number of cores,

$$P_S(t) \simeq f(\sum_{i=1}^{N} P_{CORE_i}(t)) \tag{1}$$

where $P_{CORE_i}(t)$ is the power consumption of i^{th} core at time t. Commodity servers often consist of homogeneous cores. Thus, we apply the same model here. Next, we model power states for the three distinct states of $P_{CORE_i}(t)$. When a core is unused, its power state is either,

- *Active* $\equiv P_{CORE_i}(t) = P_{ACT}$ (an idle core)
- *Sleep* $\equiv P_{CORE_i}(t) = P_{SLP}$ (a core in the *low power profile*)

In contrast, a core pinned to a VM exhibits a power state of $P_{CORE_i}(t) = F(U_{CORE_i}(t))$. Where $U_{CORE_i}(t)$ is the utilization of the core at time t, and F is a linear function [30]. Dynamics of $U_{CORE_i}(t)$ depends on the VM workload, which is a black box to the cloud operator [17]. Therefore, in packing problems, a representative utilization statistic is commonly estimated based on historical data [17], [29]. Based on this, we use U_{RT} to estimate $U_{CORE_i}(t)$. Cloud operator sets the exact U_{RT} value using deployment-specific data. As a result, the pinned power state of a core becomes,
Pinned $\equiv P_{CORE_i}(t) = P_{PIN}$, where $P_{PIN} = F(U_{RT})$

We verify our core power model with an Intel Xeon Silver CPU with 12 cores. For that, we use $U_{RT} = 100\%$. We wake up cores from 1 to 12 and plot CPU package power

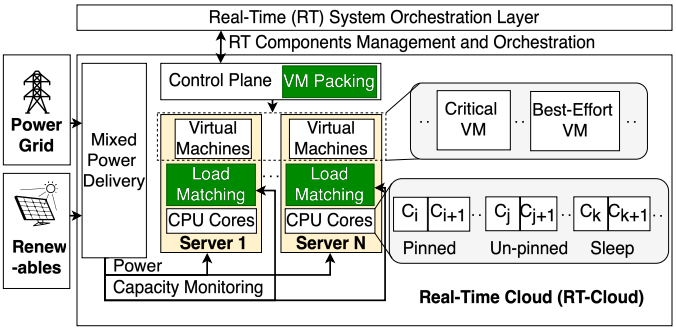

Fig. 3. A high-level system model of the proposed carbon-aware real-time cloud. We highlight components with our contributions in green.

obtained through Intel's RAPL interface. We conduct the same experiment for both *Active* and *Pinned* scenarios. Figure 4 illustrates our results. Graphs that sustain linear trends with constant slopes, corresponding to P_{PIN} and P_{ACT}. The same trends imply the P_{SLP}.

We then apply the core power model in Equation 1 and derive the following model to estimate server power using core counts as variables.

$$P_S(t) \simeq f(m(t) \times P_{PIN} + l(t) \times P_{SLP} + (N - m(t) - l(t)) \times P_{ACT}) \tag{2}$$

where at time t, $m(t)$ is the pinned core count and $l(t)$ is the sleeping core count.

Renewable energy harvest: Electricity generated from replenishing renewable energy sources (i.e., renewables) emits significantly lower amounts of carbon when compared to grid energy, which often relies on fossil fuel-based energy generation. However, most renewable energy sources rely on intermittent natural resources that vary depending on the time of the day and geographical location, such as solar and wind [2]. In return, renewables yield intermittent power capacities compared to stable grid power.

In our model, renewable energy harvesting denotes maximizing the utilization of the intermittent power capacity of renewables. We use a heterogeneous server power allocation of dedicated renewable and grid energy capacities. In doing so, each server is guaranteed a specific amount of stable power capacity, allowing the resource management layer to utilize that in maintaining the stringiest service level objectives (SLOs) of real-time VMs. The portion of the renewable energy capacity is set by the data center operator, depending on the fault tolerance levels of the data center power delivery. In Section VII-A, we provide an example of deciding that with our large-scale testbed design. The dynamics of the renewable capacity availability depends on the volume pattern of the renewable energy source. Figure 13a illustrates an

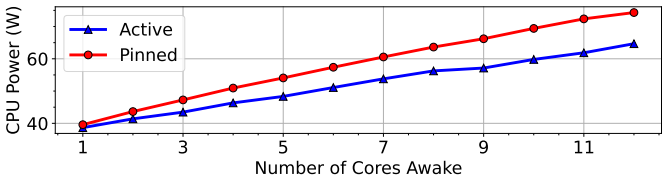

Fig. 4. CPU power as cores awake in *Renewables-driven cores*: measured with an Intel Xeon Silver CPU where core power states are: i) *Sleep* $\equiv$ sleep state of C6, ii) *Active* $\equiv$ sleep state of $POLL$, and iii) *Pinned* $\equiv$ pinned with 100% utilization .

example of that with our 14-day VM packing experiments. We assume renewable energy does not incur additional costs besides supply dynamics.

As a result of the heterogeneous server power allocation, harvesting renewables in our system model must be done at the server level. When server power meets the grid capacity (P_{GRID}), we denote $P_S(t) = P_{GRID}$. Renewable energy harvesting begins when $P_S(t) > P_{GRID}$. Therefore, for an arbitrary time period ΔT, we denote harvested renewable energy ($E_{RW}(\Delta T)$) of the server as,

$$E_{RW}(\Delta T) = \int_{\Delta T} \{u(P_S(t) - P_{GRID}) \times (P_S(t) - P_{GRID})\}\, dt \quad (3)$$

where u is the unit step function.

Service quality: We model service quality with real-time latency performance of VMs and the number of VM eviction incidents. In our system model, an application-specific middleware layer provides fault tolerance over VM evictions. Therefore, we prefer minimizing VM evictions as an objective at the resource management layer. Meanwhile, real-time latency performance impacts application business logic. We prefer a bounded latency performance for that.

Problem formulation: We formulate our problem as follows: Given an arbitrary ΔT period, maximize renewable energy harvesting while preserving service quality. Thus our **objective** is:

$$\text{Maximize} \quad E_{RW}(\Delta T) \quad \text{and} \quad \text{Minimize} \quad n$$

where $E_{RW}(\Delta T)$ is the harvested renewable energy derived in Equation 3, and n is the number of VM eviction incidents. The objective function should satisfy the following **constraints**:

$$\bar{l}_i \leq \bar{l}_{\max_i} \quad \text{and} \quad \sigma_i \leq \sigma_{\max_i} \quad \text{for} \quad vm_i \in S_{vm}$$

S_{vm} is the virtual machines executed during ΔT and $\bar{l}_i$ and σ_i are the mean and variance of real-time latency, respectively. $\bar{l}_{\max_i}$ and $\sigma_{\max_i}$ are deployment-specific upper bounds.

IV. Green Cores: Convert Renewables Utilization into a Packing Attribute

In this section, we introduce the concept of *Green Cores*, which converts the utilization of renewables as a server packing attribute, and outline its core idea, formulation, and boundaries. *Green Cores* enables us to design a framework in Section V to efficiently harvest renewables in *Real-Time Clouds*. In Section VII, we show the superiority of that over existing VM management approaches.

A. High-level idea of Green Cores

The core idea behind *Green Cores* is to identify the actual harvest of renewables, which is not reflected in *Renewables-driven cores*. Although *Renewables-driven cores* increases the available CPU cores matching that of renewables capacity, its increased core count is not an accurate signal for renewables harvest. Instead, a combination of core availability and their utilization with VMs encompass the server utilization of renewable energy capacity. Existing works addressing similar problems integrate feedback signals from power systems to identify server power draw and conduct VM packing accordingly [17] with the added complexity of integrating power domain and VM packing. Instead, we calculate a server inventory of *Green Cores* using the characteristics of *Renewables-driven cores* and our power allocation, which derives a server inventory of green and regular virtual core types. Unlike *Renewables-driven cores*, the number of green cores utilized maps to renewables harvest.

B. Comparison between Renewables-driven cores and Green Cores

Although both *Renewables-driven cores* and *Green Cores* may sound similar, they are distinct concepts. *Renewables-driven cores* is a physical notation that refers to the availability of additional cores corresponding to renewable energy dynamics. However, utilization of those additional cores does not necessarily utilize available renewables capacity. In contrast, the *Green Cores* is a virtual notation that converts the utilization of renewables into a packing attribute, such that utilizing a green core map to renewables harvest.

C. Formulation of the Green Cores server inventory

In a server, we denote the number of cores that remain in a constant *real-time power profile* as R where $0 \leq R \leq N$, such that the size of *Renewables-driven cores* at time t ($l(t)$) is $0 \leq l(t) \leq N - R$. We choose a value for R such that when R cores are at a *Pinned* power state and $l(t)$ is $N - R$, the server power draw meets the grid capacity. Using our server power model in Equation 1 we derive,

$$P_S(t) \simeq f(R \times P_{PIN} + (N - R) \times P_{SLP}) = P_{GRID} \quad (4)$$

where $m(t) = R$, $l(t) = N - R$, and P_{GRID} is the grid capacity.

We derive an equation for the amount of renewable energy harvested by subtracting Equation 4 from Equation 1.

$$\begin{aligned} P_S(t) - P_{GRID} = f(&m(t) \times P_{PIN} + l(t) \times P_{SLP} \\ &+ (N - m(t) - l(t)) \times P_{ACT} \\ &- R \times P_{PIN} - (N - R) \times P_{SLP}) \end{aligned} \quad (5)$$

We then model $m(t)$ with R as $m(t) = R + g(t)$ where $g(t)$ is an arbitrary function. Substituting this model in Equation 5 yields:

$$\begin{aligned} P_S(t) - P_{GRID} = f(&g(t) \times (P_{PIN} - P_{ACT}) \\ &+ (P_{ACT} - P_{SLP}) \times ((N - R) - l(t))) \end{aligned}$$

Here, we denote the leakage power ($L(t)$): power drawn by *Renewables-driven cores* at the *Active* state as $L(t) = (P_{ACT} - P_{SLP}) \times ((N - R) - l(t))$.

$$P_S(t) - P_{GRID} = f(g(t) \times (P_{PIN} - P_{ACT}) + L(t)) \quad (6)$$

Then, substituting Equation 6 in Equation 3, we estimate renewable energy harvest for an arbitrary time period δt where $g(t) \geq 0$,

$$E_{RW}(\delta t) = \int_{\delta t} f(g(t) \times (P_{PIN} - P_{ACT}) + L(t))\, dt$$

where f is the linear function to map CPU power into server power, in which a positive input yields a positive power value. Here, when δt is small enough to match the measurement interval of the system, the $E_{RW}(\delta t)$ can be stated as,

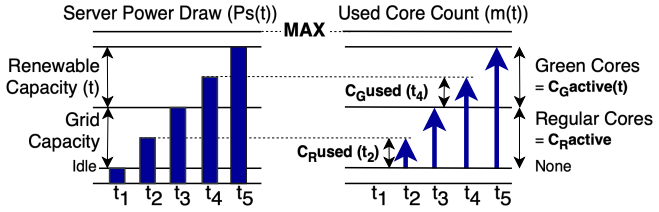

Fig. 5. Comparison of server power draw vs proposed server inventory of *Green Cores*.

$$E_{RW}(\delta t) \simeq f(g(\delta t) \times (P_{PIN} - P_{ACT}) + L(\delta t)) \times \delta t$$

where both $g(\delta t)$ and $L(\delta t)$ are values measured for the δt interval. The design of our *Renewables-driven cores* ensures that $L(\delta t)$ is independent from workload execution. In contrast, the value of $g(\delta t)$ depends on the packing decisions of the cloud's control plane. Therefore, if $g(\delta t)$ is changed with different packing algorithms,

$$E_{RW}(\delta t) \propto g(\delta t) \tag{7}$$

Equation 7 states that if the packing algorithm positively increases $g(\delta t)$, the renewable energy harvest increases. However, a positive $g(\delta t)$ also means that $m(t) > R$, implying VMs are pinned to *Renewables-driven cores*, thus increasing the eviction possibilities.

Based on the calculation, We derive the server inventory of *Green Cores*. *Green Cores* presents a server with CPU cores of two types: Green and Regular. For Green cores, active cores ($C_{G_{active}}$) are calculated with $(N - R) - l(t)$ and used cores ($C_{G_{used}}$) are calculated with $g(t)$ if $g(t) \geq 0$ (otherwise is set to 0). For Regular cores, active ($C_{R_{active}}$), and used ($C_{R_{used}}$) cores are calculated with R, and $m(t)$ if $m(t) < R$ (otherwise is set to R), respectively. Calculations of Green cores quantify the usage of renewable energy capacity, and calculations of Regular cores quantify the usage of cores that do not increase the risks of VM eviction incidents.

This is illustrated in Figure 5 with a side-by-side comparison between power domain and the server inventory of *Green Cores*. In this scenario, the number of pinned cores ($m(t)$) increases from t_1 to t_5. As a result, server power draw ($P_S(t)$) increases. Until t_3, server power draw is less than the grid capacity, where $C_{G_{used}} = 0$ and $C_{R_{used}}$ is proportionate to the utilized energy capacity. During this period, there are no risks of VM eviction incidents. Beyond t_3, the risk of VM eviction incidents increases as the server power draw utilizes renewable energy capacity, where $C_{G_{used}}$ is proportionate to the utilized energy capacity and $C_{R_{used}}$ is capped at R.

D. Boundaries of the Green Cores server inventory

The derivation of *Green Cores* server inventory is tightly coupled with *Renewables-driven cores* and the power model in our system model. As a result, it relies on the accuracy of estimation techniques we used in Section III-B. Nevertheless, the core idea of *Green Cores* can be applied to similar contexts by adjusting the inventory calculation for their specific system models.

V. Design

In this section, we outline the design of our framework. It combines a core-level VM Execution Model with a server-level VM Packing Algorithm. The VM Execution Model guarantees a static compute allocation for VMs at the core-level amidst the intermittency of *Renewables-driven cores*. To do so, it exploits the mixed-criticality of *Real-Time Clouds* and conducts criticality-aware VM evictions. In order to reduce the severity of that, we pack VMs at the server-level to optimize the number of best-effort and critical VM types provided to each server while maximizing the per-server utilization of renewable energy capacity. We do that with our server-level VM Packing Algorithm.

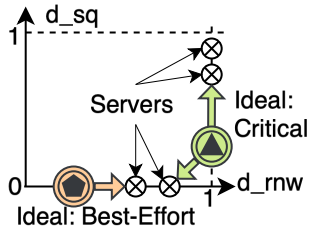

Fig. 6. Joint optimization of renewable energy harvest and VM eviction incidents via the distance to ideal points in the Euclidean space of the proposed server inventory of *Green Cores*.

A. Design of the Core-level VM Execution Model

We select a subset of cores and apply *Renewables-driven cores* to them. At 100% renewable energy capacity, we set all server cores to the *real-time power profile*. At 0% renewable energy capacity, we put all cores in the subset to a *low power profile*. For in-between, we set the *real-time power profile* to a partial amount of cores in the subset and set the *low power profile* for the rest. In this case, the number of cores in the *real-time power profile* is proportionate to available renewable energy capacity. For example, at 50% renewable energy capacity, half of the cores in the subset are set to the *real-time power profile*. In our approach, *Renewables-driven cores* dynamics depend solely on the renewable energy intermittency and are independent of the workload execution dynamics.

We pin VM cores to server cores set to the *real-time power profile* at the VM deployment and do not change it for the duration of the VM lifetime. If the number of such server cores is insufficient to serve running VMs, we perform a minimum amount of criticality-aware VM evictions. We evict *best-effort* VMs first and *critical* VMs as a last resort. Our model guarantees a static compute allocation for a VM's lifetime. The VM eviction events trigger well-defined VM failure events at the application-specific middleware layer, allowing it to recover through reconfiguration. In the next section, we design a server-level VM Packing Algorithm to reduce the possibility of such VM eviction events.

B. Design of the Server-level VM Packing Algorithm

Possibilities of VM eviction incidents with our VM Execution Model increases when the number of VMs packed in a server begin renewable energy harvesting (see Equation 3). Therefore, optimizing VM eviction incidents must be conducted jointly with optimizing the renewable energy harvest. We convert that joint optimization task into a VM packing optimization problem using the *Green Cores* server inventory we introduced in Section IV. We then design a server-level VM packing algorithm to address that.

Algorithm 1 Proposed VM Packing Algorithm

1: **function** GETPLACEMENTPREFERENCES(V: VM, S: Candidate servers, τ_1: Ideal point for *critical* VMs, τ_2: Ideal point for *best-effort* VMs)
2: $\epsilon \leftarrow GetCriticality(V)$
3: $\tau \leftarrow GetIdealPoint(\epsilon, \tau_1, \tau_2)$
4: **for all** $s_i \in S$ **do**
5: $d_{sq_i} \leftarrow GetRNW(s_i)$
6: $d_{rnw_i} \leftarrow GetSQ(s_i)$
7: $d_i \leftarrow GetDistance(d_{sq_i}, d_{rnw_i}, \tau)$
8: $s_i.score \leftarrow 1 - d_i$
9: **end for**
10: **return** $getSorted(S)$
11: **end function**
12: **function** GETIDEALPOINT(ϵ, τ_1, τ_2)
13: **return** τ_1 if ϵ is *critical* else τ_2
14: **end function**
15: **function** GETSQ(s_i)
16: $C_{R_{active}}(t), C_{R_{used}}(t) \leftarrow s_i$
17: **return** $\frac{|C_{R_{active}}(t) - C_{R_{used}}(t)|}{C_{R_{active}}(t)}$
18: **end function**
19: **function** GETRNW(s_i)
20: $C_{G_{active}}(t), C_{G_{used}}(t) \leftarrow s_i$
21: **return** $\frac{|C_{G_{active}(t)} - C_{G_{used}}(t)|}{C_{G_{active}}(t)}$
22: **end function**
23: **function** GETDISTANCE($d_{sq_i}, d_{rnw_i}, \tau$)
24: $d_{sq_\tau}, d_{rnw_\tau} \leftarrow \tau$
25: $distance \leftarrow \frac{\sqrt{(d_{sq_\tau} - d_{sq_i})^2 + (d_{rnw_\tau} - d_{rnw_i})^2}}{\sqrt{2}}$
26: **return** $distance$
27: **end function**

Algorithm 1 outlines the proposed VM packing algorithm. It takes the VM creation request (V) and the set of candidate servers (S) as inputs. It then provides a sorted list of candidate servers in the order of placement preference as the output. Additionally, it takes two other input parameters, called ideal points, each for critical VMs (τ_1) and best-effort VMs (τ_2). Our intuition behind the packing algorithm stems from the representation of a server in *Green Cores*. In that, a server is presented with two attributes: green cores and regular cores. Our algorithm uses those attributes to represent a server in a two-dimensional Euclidean feature space. Figure 6 illustrates that. For an arbitrary time t, we calculate a two-dimensional feature vector $\equiv (d_{rnw}, d_{sq})$ to represent a server in this space. We denote the axis d_{rnw} to quantify the opportunity to harvest renewables. We denote the axis d_{sq} to quantify the opportunity to deploy VMs with a minimum probability for an eviction incident. Firstly, we get the criticality of the V (ϵ), which is either critical or best-effort (line 2). Then, we filter the corresponding ideal point (τ) for ϵ (line 3). Afterward, we iterate through each server in S and calculate its feature vector. For the i^{th} server, we calculate the value for d_{rnw} as d_{sq_i} with $GetRNW$ subroutine (line 5), and we calculate the value for d_{sq} as d_{rnw_i} with $GetSQ$ subroutine (line 6). Using both, we calculate the Euclidean distance between the feature vector and the τ using the $GetDistance$ subroutine (line 7) and derive the preference score from that as closest being the higher (line 8). Using the calculated preference score, we sort the S and provide it as the output (line 10).

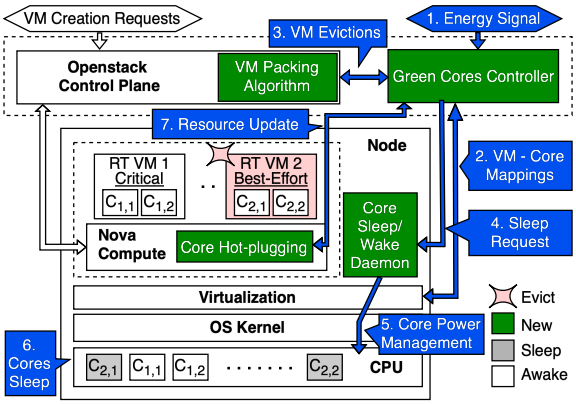

Fig. 7. System architecture of OpenStack-GC. It outlines the server load matching workflow for intermittent availability of renewable energy.

Process of VM Packing: A server inventory is empty at first, where its feature vector maps to $\equiv (1, 1)$. As VMs get packed, their regular core usage increases. Thus, the feature vector moves vertically towards $\equiv (1, 0)$. Once all regular cores are used, its green core usage increases; thus, the feature vector moves horizontally towards $\equiv (0, 0)$. This behavior allows the packing algorithm to decide its server preference depending on VM criticality. Figure 6 illustrates a scenario of ideal point placement. In that, the ideal point for critical VMs is placed between $(1, 1)$ and $(1, 0)$, such that critical VMs prefer servers with available cores supported by stable grid energy (refer Section IV-C) to reduce the eviction risks. In contrast, the ideal point for best-effort VMs is between $(0, 0)$ and $(1, 0)$, such that best-effort VMs prefer servers that draw power beyond the grid allocation to maximize renewables harvest. Tunable ideal points in our algorithm enable the cloud operator to adjust for deployment-specific performances [18]. In Section VII, we show its superiority in the sensitivity analysis of our large-scale VM packing testbed.

VI. IMPLEMENTATION

In this section, we outline *openstack-gc*: implementation of our framework in OpenStack.

Implementation of Openstack-GC: Figure 7 illustrates the system architecture of *openstack-gc*. We highlight newly added OpenStack extensions in green. We deploy an on-node daemon service to realize *Renewables-driven cores*. We introduce a Green Cores Controller at the control plane to orchestrate the proposed VM Execution Model. We implement the proposed VM Packing Algorithm as a VM scheduling algorithm in OpenStack.

Renewables-driven cores: We implement an on-node daemon service in Golang to control per-core power profiles. By making an API call to the daemon service, the *openstack-gc* control plane can specify the number of cores to put into a specific power profile. If the *real-time power profile* is requested, the daemon service sets the requested number of cores into a high-performance state. If the *low power profile* is requested, the daemon service sets the requested cores into a deep sleep state. The daemon service wraps the Intel Power

TABLE III
OPENSTACK-GC PROTOTYPE: NODE SPECIFICATIONS

Attribute	Description
Server Model	ProLiant DL380 Gen10
CPU	Intel(R) Xeon(R) Silver 4214
Physical Cores	12
Hyper Threading	Disabled
Renewables-driven Cores	6
Real-time power profile	C-state = $POLL$ at 2699 MHz
Low power profile	C-state = C6

TABLE IV
OPENSTACK-GC PROTOTYPE: VM SPECIFICATIONS

Attribute	Description
Resources	CPU: 6 Cores, RAM: 6GB
OS	CentOS 7
Kernel	Linux 3.10.0 + CERN's Real-Time patches
System Load	Load test of RTEval [15]
Latency Monitoring	Cyclictest [31]

Optimization Library[1] and overrides the kernel management of the sleep state and operating frequency of each core to achieve this.

VM Execution Model: We orchestrate our VM Execution Model using the load matching workflows of *openstack-gc*. First, we enable the dedicated cores feature in OpenStack to pin each VM core to a dedicated server core, resulting in a static core allocation for each deployed VM. Then, our load-matching workflows of *openstack-gc* take place. Suppose an increased energy capacity signal arrives to *openstack-gc*. In that case, the Green Cores Controller calculates and notifies on-node daemon services to set the required number of cores from the *low power profile* to the *real-time power profile*. If a decreased energy capacity signal is provided to *openstack-gc*, the Green Cores Controller pings APIs of the virtualization layer (*openstack-gc* uses Libvirt[2]) in each node to obtain mappings of VM cores to server cores. Then, the Green Cores Controller calculates and triggers criticality-aware VM evictions by blocking API calls to the OpenStack. Upon completion, the required cores are put to the *low power profile* using on-node daemon services. Figure 7 illustrates the workflow for the decreased energy capacity. In both cases, our modified Nova Compute, OpenStack's on-node compute service, periodically polls the Green Cores Controller to obtain cores at the *low power profile*. Afterwards, Nova Compute signals the control place to omit cores from VM scheduling in the *low power profile*.

VM Packing Algorithm: We modify the OpenStack scheduler service to poll the Green Cores Controller and obtain server inventory attributes of *Green Cores* for all server nodes. To provide that, the Green Cores Controller pings virtualization layers of servers to obtain core usage information and calculates server inventory attributes of *Green Cores*. Our implementation of the proposed VM Packing Algorithm as a VM scheduling algorithm in OpenStack consumes obtained *Green Cores* server attributes to make VM placement decisions.

VII. PERFORMANCE EVALUATION

We evaluate our core-level VM Execution Model and the server-level VM Packing Algorithm using a multi-node *openstack-gc* prototype deployment. Further, we evaluate its efficacy at scale using long-running production VM traces over a large-scale simulation testbed.

A. Experimental Setup

Openstack-GC prototype experiments: We deploy a prototype two-node *openstack-gc* cloud on HPE ProLiant servers with 12-core Intel Xeon Silver CPUs. Table III outlines its node specifications.

Workload: Table IV outlines the VM specifications for our real-time workloads. We use CentOS 7 VMs with CERN's real-time kernel patches [32] applied. We run the RTEval tool from the Linux foundation project, Real-Time Linux [15], to emulate a system load. Alongside the load, RTEval continuously measures the VM kernel's real-time performance via the Cyclictest tool [31]. Further, we synthesize 30-minute traces for VM arrivals and renewables dynamics from Microsoft Azure's VM packing trace [16] and ELIA solar data [33].

Baseline: We use Harvest Virtual Machines (HVM): the existing VM execution model over *Renewables-driven cores* [13] to evaluate advancements of our VM Execution Model. We use Openstack's default VM packing implementation in its nova scheduling service [14] to evaluate advancements of our VM Packing Algorithm.

Metrics: We use Intel's Running Average Power Limit (RAPL) [34] interface to capture CPU metrics in the server every 0.5 seconds. We collect i) core residencies at the C6 sleep state and ii) core operating frequency in MHz. Further, we use server power estimation using the linear power model of CPU power that is shown over 90% accuracy [29]. For that, we collect PkgWatt metric in RAPL (power consumption of the CPU socket [35]) as the server power metric (Figure 4 illustrate the verification of CPU power estimation with RAPL for our system model). Inside VMs, we measure real-time performance with the latency to wake up a real-time thread using the Cyclictest tool [31]. For VM packing performance, we use the *Eviction Incidents* to count the number of eviction incidents of *best-effort* and *critical* VMs. We use the *Normalized Lifetime (nLT)* to measure the severity of an eviction incident. For each evicted VM, we normalize its lifetime from the original lifetime in the trace. A lower nLT value implies increased severity. We use scheduling overhead time to measure the same with packing algorithms. For that, we analyze logs from OpenStack to identify the scheduling duration for VM creation requests.

Trace-driven simulations at scale: We use 8K+ servers, each with 40 CPU cores, to match the realistic similar values in Microsoft's Azure's cloud zones [36]. Existing fallback mechanisms of Azure's data center power delivery suggest that a 12% power overdraw is manageable [17]. To operate within that, we add four cores in each server and use them as *Renewables-driven cores*.

Workload: We use the full 14-day Azure VM packing trace [16], which contains request arrivals, resource requirements, lifetime on Azure, and criticality. We use renewable dynamics from ELIA solar data [33]. We normalize and scale the renewable dynamics trace, so that the maximum renewable energy capacity can wake all *Renewables-driven cores* in a

[1]https://github.com/intel/power-optimization-library.git
[2]https://www.libvirt.org

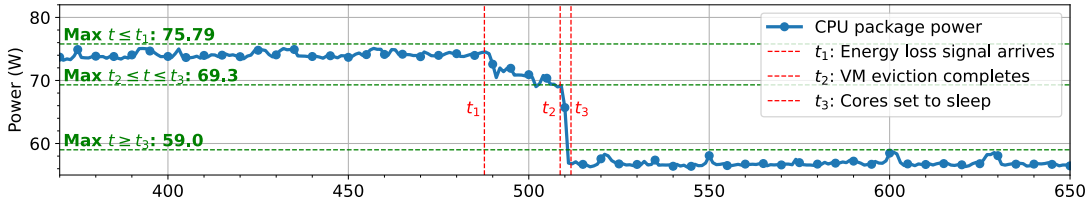

Fig. 8. CPU package power measurements obtained through Intel's RAPL interface [34] during the load matching experiment of the OpenStack-GC prototype. The proposed VM Execution Model manages the server load over the renewable energy availability. Before t_1, the server executes two 6-core VMs. An energy loss signal at t_1 triggers the eviction of one VM to unpin six cores, completing at t_2. The unpinned cores enter into deep sleep at t_3.

server.

Baselines: To evaluate the proposed VM packing algorithm, we use two comparison baselines. The *Best-Fit packing (best-fit)* is a commonly used packing approach in production clouds [36], [37] that packs VMs tightly in servers. We use it to evaluate our advancements over a commonly used VM packing approach. The *Criticality-Aware packing (crt-aware)* is a packing approach that reduces VM throttling incidents in power over-subscribed data centers [17]. Similar to our problem context, it leverages VM criticality to reduce VM performance impact incidents incurred from server load exceeding available power capacity. We use it to evaluate our advancements over the state-of-the-art.

Metrics: In addition to *Eviction Incidents* and *Normalized Lifetime (nLT)*, we use the *Harvested Renewables* to measure utilization of renewable energy capacity. Using derivations of *Green Cores*, we calculate it as $\equiv \int_0^T C_{G_{used}}(t)dt$ for a period of T.

B. Evaluation of Core-level VM Execution Model

We evaluate the proposed VM Execution Model's ability to maintain the server load to match available renewable energy capacity and its impact on the real-time performance of VMs. Firstly, we signal *openstack-gc* prototype deployment with a 100% energy capacity to wake all *Renewables-driven cores* in the server. Then, we pin all cores by deploying two 6-core VMs. In both VMs, we run the RTEval program for the duration of the experiment to emulate a peak load. Then, we signal a 0% energy capacity.

Figure 8 shows the CPU package power observed throughout. We collect it via Intel's Running Average Power Limit (RAPL) [34] interface. The CPU package draws up to 75.79W at peak load with a relatively constant trend. t_1 denotes the arrival of the energy loss signal for 0% renewable energy capacity. *openstack-gc*'s response shows a two-stage power reduction; t_1 - t_2 and t_2 - t_3. The former shows the power reduction from evicting one of the VMs to unpin six cores. Latter shows the power reduction from putting unpinned cores to deep sleep. After t_3, CPU package power does not exceed 59W. It translates to a 22% reduction of the peak power draw. With the linear model of CPU power to server power [6], our *openstack-gc* deployment shows a reduction of 22% of the server peak power in matching 100% to 0% loss of renewable energy capacity.

Figure 9 shows CPU core power characteristics. We capture operating frequency and C6 deep sleep state residency (i.e. in a given measurement period, the percentage that the core resided in the sleep state) of cores. We average it for the six cores that

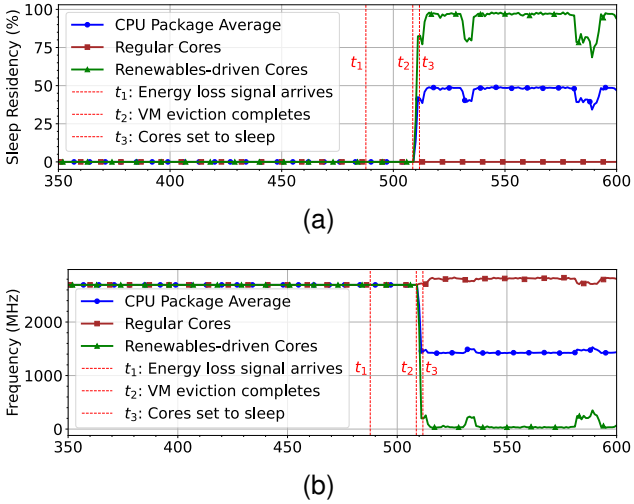

Fig. 9. CPU core power characteristics in Openstack-GC prototype during server load matching for renewable energy loss. We obtain power metrics through Intel's RAPL interface [34]. (a) CPU core C6 deep sleep state residency. (b) CPU core operating frequency.

enter deep sleep state after t_3 (Renewables-driven cores), for the six cores that continue to operate, and for overall. Until t_2, *openstack-gc* maintains a constant 2700 MHz operating frequency of cores with 0% residency in deep sleep, showing cores operating at the *real-time power profile*. Afterwards, Renewable-driven cores mostly reside in a deep sleep with 0 MHz operating frequency, showing their *low power profile*. It shows that *openstack-gc* only changes power profiles after the VM eviction completion at t_2. Throughout the lifetimes of VMs, VM cores are allocated with physical cores in the *real-time power profile*.

The spikes in maintaining the *low power profile* show the characteristics of controlling core power through Intel's Power Optimization library that we use in *openstack-gc*. Our prototype also has an overhead of running external services, including the OpenStack control plane services in the same server, which could be attributed to the spikes shown. Despite that, the server power draw shows a 59W upper bound in figure 8, showing *openstack-gc* can maintain a constant power reduction over the intermittent spikes in the *low power profile*.

We then evaluate the superiority of our VM Execution Model's static core allocation shown in the load matching experiments by comparing that with baseline HVM's approach of dynamic physical core allocation. HVM's approach is to change the number of physical cores (pCPUs) allocated to the VM while preserving the number of virtual cores (vCPUs). To evaluate its workload impact on real-time computing, we monitor the real-time performance of an HVM over the dynamic allocation of pCPUs. Our experimental HVM consists

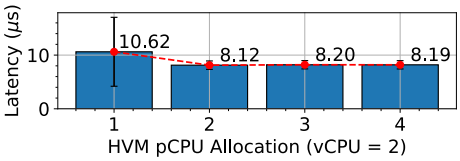

Fig. 10. Comparison of real-time latency performance of a two-core Harvest VM (HVM) [13] over different physical CPU core allocations.

of 2 vCPUs. We set pCPUs to the *real-time power profile*. We then execute the RTEval [15] program inside the HVM to measure the real-time performance. We dynamically adjust the mapping of pCPUs through our virtualization management, libvirt's APIs[3].

The results are illustrated in Figure 10. When the number of pCPUs $\geq$ to the number of vCPUs, the HVM sustains a consistent real-time latency performance, having both mean and mean absolute deviation statistics consistent for all three cases of pCPUs ≥ 2. The opposite shows increased latency variance inside the HVM. Compared to pCPUs $= 2-$which maps to the performance of our VM Execution Model due to its static core allocation$-$the case of pCPUs $= 1$ increases the mean latency by 30% alongside a $7.32\times$ increase of the mean absolute deviation of latency. In contrast, our VM Execution Model can incur VM evictions. In the next experiment, we evaluate its impact on real-time application performance.

Next, we evaluate the impact of the proposed VM Execution Model at the application layer. We use the Harvest VM (HVM) as a comparison baseline, the existing VM execution model over *Renewables-driven cores* [13]. We use an experimental deployment of OSM MANO [19], a Virtual Network Functions (VNF) orchestration and management application layer, to match a real-time application layer having both *critical* and *best-effort* components (see Table I). In public clouds, server utilization is around 60%, and the packing density (i.e. utilization of servers running at least one VM) is around 85% [36]. To match that, we use two 12-core servers and tightly pack one server with two 6-core VMs while the other is left unused. MANO is a generic orchestration layer where the exact time-bound requirements depend on the use case. Therefore, for VMs, we measure the real-time latency of the VM kernel for a consistent real-time latency performance independent of the use case. To match application-level reconfiguration over component failures, we enable MANO's auto-heal feature, which reconfigures itself via VM redeployment. HVM executes VMs under resource variations. To match it's worst-case, we set the dynamics of Renewables-driven cores to sleep five cores in both servers, such that HVM executes a VM with 6 VM cores allocated to one physical core. In contrast, the proposed approach evicts one of the VMs, triggering MANO to redeploy it in the unused server. We obtain the time taken for reconfiguration via MANO's event logs.

Figure 11 shows the real-time latency performance of the affected VM in both the proposed and HVM approaches. In both methods, the remaining VM continues executing under the same core allocations without any performance impact since the server's physical core count is sufficient. Until the server core sleep event at time axis $=$ 300, the affected

[3]https://www.libvirt.org

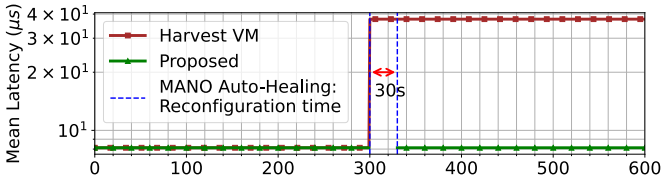

Fig. 11. Application layer real-time performance comparison of VM execution models. Experimental setup deploys VNF orchestration application, OSM MANO [19], over two 12-core server nodes of *openstack-gc* prototype. We plot the mean real-time latency obtained through RTEval [15] running in the affected VMs. MANO's auto-heal feature is enabled to reconfigure itself upon VM eviction by redeploying. Initially, two 6-core VMs tightly pack a server where the other is left unused to match resource usages of public clouds [36]. At $t = 300$, worst-case resource reduction with an HVM is emulated by sleeping five cores in each server. The proposed model evicts a VM, where HVM reduces the physical core count to one in a VM.

VM in both approaches shows the same mean real-time latency. Afterwards, the core count reduces. With HVM, the affected VM's mean real-time latency increases from $8.13\mu s$ to $37.65\mu s$. When comparing the coefficient of variation of the VM's real-time latency, it increases to $6.52\times$. With the proposed approach, the affected VM undergoes a 30-second service unavailability. However, when it resumes afterwards, the VM retains the same real-time latency performance. The results show that, unlike the existing temporal flexibility-based approach, the proposed VM Execution Model maintains intact real-time latency performance. In doing so, it incurs brief service unavailability from VM evictions as a trade-off. In the next section, we evaluate the role of our proposed VM packing algorithm in reducing the impact of that on the application layer.

C. Evaluation of Server-level VM Packing Algorithm

We first evaluate the practical aspects of our VM Packing Algorithm with the *openstack-gc* prototype over the default OpenStack. We focus on the scheduling overhead of our implementation and the impact of service quality on renewables dynamics. Then, we replay long-running VM arrivals and renewable dynamics to evaluate renewables harvest and long-term service quality impact with the large-scale simulation testbed.

We write a Python client to read VM arrivals in the Azure trace and make VM creation requests to *openstack-gc* deployment in real time. For each VM request, it spawns a lifecycle management thread, which then waits in real-time and makes the VM creation request. Afterward, it periodically polls the deployment to check the deployed VM status. Management thread completes if the VM has prematurely deleted, which is then marked as an eviction incident, or the VM has lived to the lifetime provided in the trace data, which then is deleted via a request made to the deployment. In parallel, a separate client emulates renewables dynamics by reading the trace data. It emulates a single peak renewables dynamics matching 24-hour solar availability by switching *Renewables-driven cores* through *openstack-gc* APIs.

Figure 12 illustrates our results. Our proposed algorithm outperforms OpenStack nova regarding the severity of eviction incidents. Eviction incident counts in Figure 12a show our proposed algorithm reduces critical VM percentage from 0.205 to 0.195 while leveraging that with best-effort VM evictions.

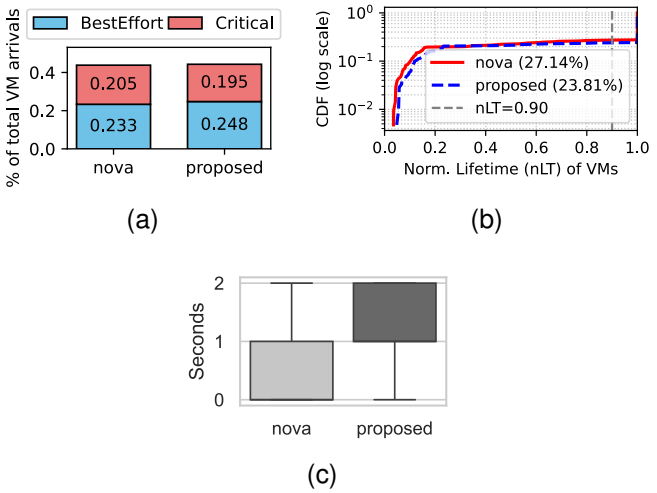

Fig. 12. Performance of the proposed VM Packing Algorithm in VM packing experiments of *openstack-gc* prototype. (a) Number of VM eviction incidents. (b) Distribution of normalized lifetimes (nLT) of VMs with nLT CDF value $\leq 90\%$. (c) Distribution of scheduling overhead in VM deployment.

In Figure 12b, our proposed algorithm reduces CDF value for the 90% of normalized lifetime of VMs from 27.14% to 23.81%. In return, Figure 12c illustrates the distribution of scheduling overhead in VM requests. Note that OpenStack logs that we leverage for that have a granularity of seconds. In measuring the overhead values, we disable the synchronization overhead of *openstack-gc* for OpenStack nova. In return, results demonstrate the impact of additional scheduling overhead in *openstack-gc*, where the overhead distribution concentrates to 2 seconds from 1 second. Increased scheduling overhead can delay the VM deployment, resulting in lesser optimized packing decisions. For example, servers may increase critical VM eviction incidents with renewable dynamics due to the lack of best-effort VMs. Most of the scheduling overhead is attributed to the synchronization implementation of *openstack-gc* prototype, where the controller polls each server to collect information in calculating the server inventory of *Green Cores*. With that, the overhead shown in the results can increase with the deployment size. However, apart from that, the remaining algorithm implementation does not significantly increase the scheduling overhead. We use OpenStack's default filter scheduler and integrate our algorithm with its existing iteration of servers, avoiding additional re-iterations. Nevertheless, our synchronization implementation in the *openstack-gc* prototype can be improved by applying scheduling optimization techniques. For instance, Azure's production VM scheduler, Protean [36], addresses a similar scaling problem in VM packing by implementing an optimistic concurrency model. Collectively, the results of VM packing with *openstack-gc* prototype highlight its potential in improving VM eviction severity and opportunities to improve its scheduling overhead for production deployments. Next, we evaluate the proposed VM packing algorithm at scale for renewables harvesting over long-running experiments.

For that, we use a large-scale simulation test bed to evaluate our framework at the data center scale. In the test bed, we first implement *Renewables-driven cores* with a trace of renewable energy dynamics and then implement the proposed VM Execution Model. We expose the test bed to a 14-day Azure VM workload arrival trace. We use our proposed VM packing algorithm and comparison baselines to determine VM allocations across servers.

We first tune our algorithm by observing its performance over short-running experiments. We aim to jointly optimize the reduction of VM eviction incidents and increase renewable energy harvest. Once tuned, we conduct 14-day packing experiments. Figure 13 shows (a) harvested renewable energy and (b) the number of VM evictions. Values for the former are normalized among the comparison baselines, and values for the latter are expressed as a percentage of the total number of VM requests.

Both baselines show their inability to conduct joint optimization. The best-fit algorithm is most effective in harnessing renewable energy yet evicts over 2% of VM requests. It incurs the highest amount of *critical* VM evictions among the three algorithms. The crit-aware, on the other hand, shows the most effectiveness in reducing VM eviction incidents with 0.25% of total VMs evicted with a $1.703\times10^{-4}\%$ of *critical* VM evictions, the lowest amongst three algorithms. However, it shows the least harvested renewable energy with an 80% reduction from the best-fit algorithm. Our proposed algorithm shows a joint optimization, a 34.83% increase over crit-aware in harvested renewable energy and a 79.64% reduction of VM eviction incidents compared to best-fit. Our algorithm reaches 50% of the renewable energy harvest performance of best-fit with a 26.09% VM eviction incidents of best-fit, showing its joint optimization characteristic to favour lesser eviction incidents. Figure 15 shows distributions of a normalized lifetime (nLT) of evicted VMs. The proposed VM packing algorithm surpasses best-fit and approaches crit-aware with the CDF value for nLT $\leq 90\%$.

Sensitivity analysis: We conduct a sensitivity analysis of our algorithm's hyper-parameters. In the proposed packing algorithm, we represent each server using a 2-dimensional feature vector $\equiv (d_{rnw}, d_{sq})$: d_{rnw} quantifies renewable energy usage and d_{sq} quantifies the possibility of VM eviction incidents. Parameters of our algorithm are two instances of this vector (called ideal points), one for each *critical* and *best-effort* VM type. For initial values, we set *critical* VM ideal point to $(1, 0.5)$ such that those VMs prefer servers with the potential to reduce VM eviction incidents, and *best-effort* ideal point to $(0.2, 0.0)$ such that those VMs prefer servers with the potential to harvest renewable energy.

In subsequent experiments, we move *critical* ideal point closer to the other and conduct 24-hour packing experiments in each step. Figure 14 illustrates our results. As ideal points move closer, our proposed algorithm favours increasing renewable energy harvest, surpassing the leading baseline best-fit at a distance of 0.05. Although this behaviour compromises eviction incidents, the number of incidents is still less than that of the best-fit. In contrast, as ideal points deviate, our proposed algorithm favours decreasing eviction incidents, surpassing the leading baseline crt-aware at distances of 0.9875 and 1.30.

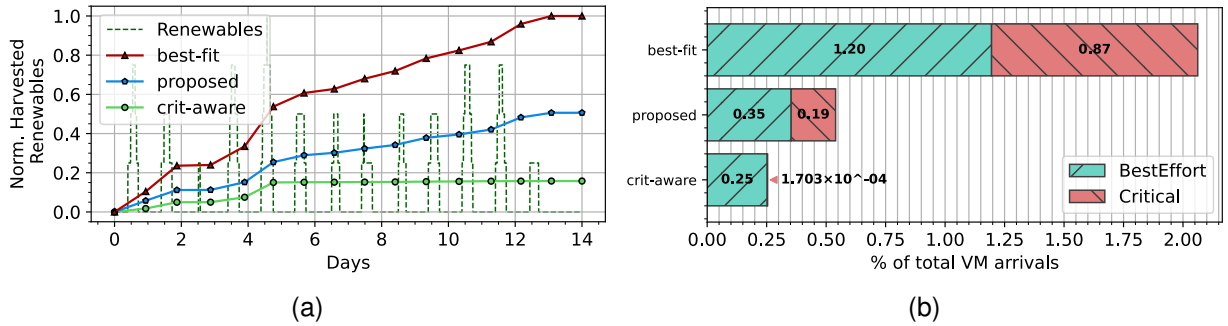

Fig. 13. Performance of the proposed VM Packing Algorithm in packing a 14-day Azure VM arrival trace [16] to jointly optimize renewable energy harvest and VM eviction incidents. (a) Accumulation of harvested renewable energy capacity. (b) The number of VM eviction incidents.

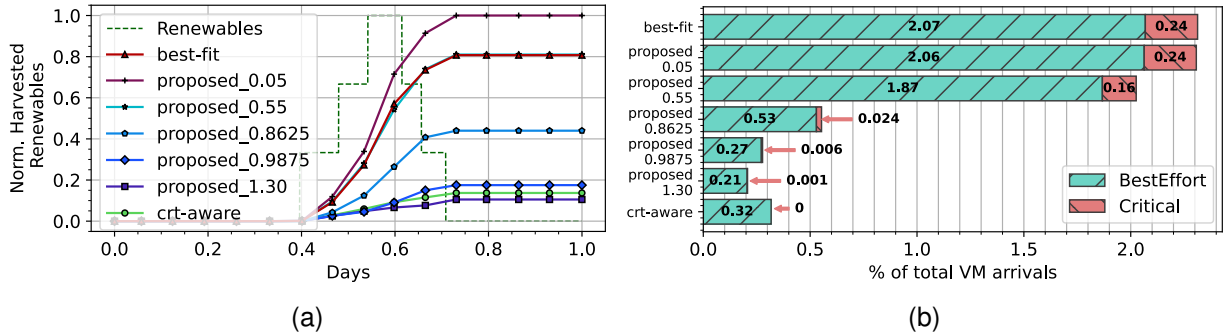

Fig. 14. Performance of the proposed packing algorithm during its sensitivity analysis. We change the distance between its ideal point parameters and conduct a packing experiment of 24 hours in each step. (a) Accumulation of harvested renewable energy capacity. (b) The number of VM eviction incidents.

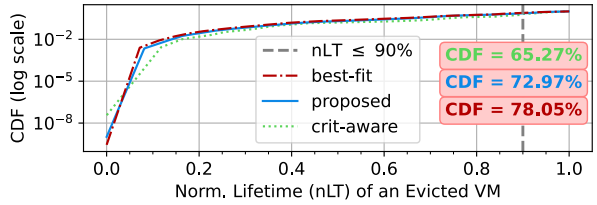

Fig. 15. Distribution of normalized lifetime (nLT) of evicted VMs during the 14-day VM packing experiment.

D. Discussion

Our evaluations show the potential of our framework to manage server power using *Renewables-driven cores*. Per-core application of *low power profile* demonstrates our framework can reduce the server power to match supply variations of renewable energy. CPU power metrics shown during that indicate that if a core pins to a VM, that core's power profile transition will not occur. Even if unused cores are insufficient, the framework evicts the VM first before changing the power profile. As a result, our framework guarantees a static compute allocation throughout a VM's lifetime. Evaluation of the real-time latency performance of VM kernels shows that the static compute allocation provided in our framework significantly outperforms existing workload temporal-flexibility-based VM execution solutions.

Our approach shows two trade-offs. Firstly, a sustained core power profile until the completion of VM evictions requires redundancies in the data center power delivery to support the short periods of server power overdraws. However, existing cloud data centers can support similar requirements [17]. Therefore, our framework fits into existing data center designs. Secondly, the static compute allocation requires VM evictions if enough unused cores are unavailable to match the energy supply. However, an application-specific middleware layer in clouds manages real-time VMs, which provides fault tolerance over VM evictions [10], [19]. Therefore, VM evictions in our framework do not incur application-level failures for real-time workloads. Moreover, large-scale packing experiments show that our framework can reduce the number of VM eviction incidents by utilizing core availability across the servers, jointly optimizing that with the utilization of renewable energy capacity. Our eviction-based approach exploits findings of a previous study showing that cloud applications prefer VM evictions over continued VM execution with performance degradation [17].

Performance of our framework improves with the presence of *best-effort* VMs. Therefore, cloud operators need to tune our algorithm according to the workload variations. Sensitivity analysis of our framework's packing algorithm parameters shows the ability to support that (see Section VII-C). The algorithm can be tuned to favour renewable energy harvesting for a deployment that expects an increased number of *best-effort* VMs. Otherwise, it can be tuned down to reduce the number of VM eviction incidents. Our framework design expects server utilization levels in typical data centers, where a slack of unused capacity is available [36]. It allows the real-time application layer to reconfigure in the events of VM evictions. If the data center utilization levels are much higher, the application layer may be unable to do so. In such cases, the algorithm tuning must be adjusted to reduce eviction incidents.

VIII. Related Work

Load matching with renewables-driven cores: Common load matching techniques for intermittent renewable energy,

TABLE V
COMPARISON OF RELEVANT WORK WITH OURS

Work	Renewables-driven Cores	Critical Workloads	Real-Time Readiness	VM Execution	Criticality-aware Packing	Renewables Harvest
SolarCore (2011) [23]	✓					✓
Chameleon (2013) [24]	✓					✓
Kumbhare et. al (2021) [17]		✓		✓	✓	
PowerMorph (2022) [38]	✓	✓				✓
Slackshed (2023) [13]	✓			✓		✓
Our Proposed	✓	✓	✓	✓	✓	✓

such as geographical load balancing, workload migration, admission control, and capacity planning [5]–[7], depend on either suspending/resuming or migrating flexible workloads. In contrast, *Renewables-driven cores* avoids both by performing load matching with dynamic core availability. SolarCore [23] and Chameleon [24] use per-core Dynamic Voltage and Frequency Scaling (DVFS) and Power Gating to implement *Renewables-driven cores*. In their work, workloads utilizing power-adjusted cores can undergo performance degradation, thus better suited for throughput workloads with flexible deadlines. PowerMorph [38] improves this via core grouping, hosting *critical* and *best-effort* workloads and power adjustments isolated to core groups. However, workload core affinity can dynamically change during load matching, unfavourable for time-critical workloads such as real-time compute [21]. Slackshed [13] implement *Renewables-driven cores* for virtual machine (VM) execution. They achieve uninterrupted VM execution at the expense of dynamic CPU allocation, thus better suited for throughput workloads with flexible time constraints. In contrast, our work preserves workload time boundaries over *Renewables-driven cores* and leverages criticality-aware VM evictions within safe limits of the application layer.

VM packing algorithms: VM packing is a widely studied research problem. Most existing works focus on variants of bin packing algorithms to improve resource utilization at scale [36], [39], yet consider servers as static inventories. Opposed to that, Kumbhare et al. [17] explore an inventory where servers oversubscribe power delivery, yielding a dynamically changing inventory capacity. They propose a criticality-aware packing algorithm to co-pack *critical* and *best-effort* components, reducing workload impact. However, in doing so, they do not consider renewable energy harvesting opportunities. In contrast, our work achieves joint optimization of workload impact and harvesting in dynamic inventories.

IX. CONCLUSIONS AND FUTURE WORK

In this work, we proposed a framework to harvest renewable energy with real-time workloads in clouds using *Renewables-driven cores*. Existing works address only flexible workloads and fail to preserve the time bounds of real-time computing. To address that, we used a two-fold design of: i) Core-level VM Execution Model for load matching via criticality-aware VM evictions, and ii) Server-level VM Packing Algorithm to reduce VM eviction incidents. We implemented our framework in OpenStack as *openstack-gc*. To evaluate, we used a prototype *openstack-gc* cloud and trace-driven simulations. As evidenced through empirical results, the static compute allocation provided by our framework demonstrated its superiority in real-time workloads by reducing the coefficient of variation of real-time latency in VMs by $6.52\times$ over the existing workload temporal-flexibility-based solution. Furthermore, our proposed framework showcased its safe energy harvesting capability with a joint 79.64% reduction of VM eviction incidents and 34.83% increase of harvested renewable energy over state-of-the-art baselines. Moreover, our sensitivity analysis of parameters demonstrated its capacity to cater to specific harvesting requirements.

In the future, we intend to advance the server power model in our work by leveraging prediction-based VM utilization models. Further, power profile transitions in our framework can impact CPU temperature. Thus, it is worth exploring the thermal impact of our framework on CPU longevity to help reduce data center embodied carbon footprint. Additionally, as AI and machine learning models become popular, adopting them in real-time workloads can require power-hungry GPU accelerators. Extending the concept of *Green Cores* virtual inventory to GPUs can help cloud operators reduce their operational carbon footprint. From the application perspective, advancing the *Green Cores* concept to handle VM evictions as a service quality constraint enables real-time systems with limited fault tolerance to integrate with our framework.

Software availability: *openstack-gc* has been open-sourced at https://github.com/tharindu-b-hewage/openstack-gc.

REFERENCES

[1] IEA. (2023) International energy agency's data centres and data transmission networks. [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks

[2] R. Bianchini, C. Belady, and A. Sivasubramaniam, "Datacenter power and energy management: past, present, and future," *IEEE Micro*, pp. 1–9, 2024, early access.

[3] IEA. (2023) International energy agency's report on low-emissions sources of electricity. [Online]. Available: https://www.iea.org/reports/low-emissions-sources-of-electricity

[4] T. Sukprasert, A. Souza, N. Bashir, D. Irwin, and P. Shenoy, "On the limitations of carbon-aware temporal and spatial workload shifting in the cloud," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 924–941.

[5] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[6] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne, "Carbon-aware computing for datacenters," *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270–1280, 2023.

[7] J. Zheng, A. A. Chien, and S. Suh, "Mitigating curtailment and carbon emissions through load migration between data centers," *Joule*, vol. 4, no. 10, pp. 2208–2222, 2020.

[8] E. Barbieri. (2023) What is real-time linux? part i. [Online]. Available: https://ubuntu.com/blog/what-is-real-time-linux-i

[9] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2449–2457.

[10] G. Gala, G. Fohler, P. Tummeltshammer, S. Resch, and R. Hametner, "Rt-cloud: Virtualization technologies and cloud computing for railway use-case," in *Proceedings of the 24th IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2021, pp. 105–113.

[11] J. R. David Reinsel, John Gantz. (2018) The digitization of the world from edge to core. [Online]. Available: https://www.readkong.com/page/the-digitization-of-the-world-from-edge-to-core-8666239

[12] L. Piga, I. Narayanan, A. Sundarrajan, M. Skach, Q. Deng, B. Maity, M. Chakkaravarthy, A. Huang, A. Dhanotia, and P. Malani, "Expanding datacenter capacity with dvfs boosting: A safe and scalable deployment experience," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024, pp. 150–165.

[13] A. Agarwal, S. Noghabi, I. Goiri, S. Seshan, and A. Badam, "Unlocking unallocated cloud capacity for long, uninterruptible workloads," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 457–478.

[14] Openstack. (2024) The most widely deployed open source cloud software in the world. [Online]. Available: https://www.openstack.org

[15] Linux. (2024) Linux foundation projects: Real-time linux tools: Rteval. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rteval

[16] Azure. (2020) Azure trace for packing 2020. [Online]. Available: https://github.com/Azure/AzurePublicDataset/blob/master/AzureTracesForPacking2020.md

[17] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, and R. Bianchini, "Prediction-Based power oversubscription in cloud platforms," in *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 473–487.

[18] Ubuntu. (2023) A cto's guide to real-time linux. [Online]. Available: https://ubuntu.com/engage/cto-guide-real-time-kernel

[19] ETSI. (2020) European telecommunications standards institute's osm mano autohealing. [Online]. Available: https://osm.etsi.org/docs/user-guide/v15/05-osm-usage.html#autohealing

[20] Intel. (2024) Overview of intel® time coordinated computing (tcc) tools – measurement library. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/real-time-systems-measurement-library.html

[21] Openstack. (2021) Real time. [Online]. Available: https://docs.openstack.org/nova/2023.2/admin/real-time.html

[22] Intel. (2016) Nfv performance optimization for virtualized customer premises equipment. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/nfv-performance-optimization-for-vcpe.html

[23] C. Li, W. Zhang, C.-B. Cho, and T. Li, "Solarcore: Solar energy driven multi-core architecture power management," in *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture*, 2011, pp. 205–216.

[24] C. Li, X. Li, R. Wang, T. Li, N. Goswami, and D. Qian, "Chameleon: Adapting throughput server to time-varying green power budget using online learning," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 100–105.

[25] P. Ambati, I. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing SLOs for Resource-Harvesting VMs in cloud platforms," in *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 735–751.

[26] G. Durrieu, G. Fohler, G. Gala, S. Girbal, D. Gracia Pérez, E. Noulard, C. Pagetti, and S. Pérez, "DREAMS about reconfiguration and adaptation in avionics," in *Proceedings of the ERTS 2016*, 2016, pp. 48–57.

[27] K. M. U. Ahmed, M. H. J. Bollen, and M. Alvarez, "A review of data centers energy consumption and reliability modeling," *IEEE Access*, vol. 9, pp. 152 536–152 563, 2021.

[28] S. Qi, D. Milojicic, C. Bash, and S. Pasricha, "Shield: Sustainable hybrid evolutionary learning framework for carbon, wastewater, and energy-aware data center management," in *Proceedings of the 14th International Green and Sustainable Computing Conference*, 2024, p. 56–62.

[29] A. Radovanovic, B. Chen, S. Talukdar, B. Roy, A. Duarte, and M. Shahbazi, "Power modeling for effective datacenter planning and compute management," *IEEE Transactions on Smart Grid*, vol. 13, no. 2, pp. 1611–1621, 2022.

[30] R. Basmadjian and H. de Meer, "Evaluating and modeling power consumption of multi-core processors," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, 2012.

[31] Linux. (2023) Linux foundation projects: Real-time linux tools: Cyclictest. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

[32] CERN. Rt (realtime) cern. [Online]. Available: https://linux.web.cern.ch/rt/

[33] ELIA. (2024) Elia group open data platform. [Online]. Available: https://www.elia.be/en/grid-data/open-data

[34] Intel, *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Intel, 2016.

[35] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, 2018.

[36] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM allocation service at scale," in *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 845–861.

[37] OpenStack. (2023) Compute schedulers. [Online]. Available: https://docs.openstack.org/nova/2023.2/admin/scheduling.html

[38] A. Jahanshahi, N. Yu, and D. Wong, "Powermorph: Qos-aware server power reshaping for data center regulation service," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, 2022.

[39] T. Baker, B. Aldawsari, M. Asim, H. Tawfik, Z. Maamar, and R. Buyya, "Cloud-senergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 242–252, 2018.

Tharindu B. Hewage is working toward a PhD with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, University of Melbourne, Australia. His research interests include distributed systems and cloud computing. His current research focuses on energy and carbon-aware resource management in edge-cloud systems.

Shashikant Ilager is an assistant professor at the Informatics Institute, University of Amsterdam, Netherlands. He is a member Multiscale Networked Systems research group. He works at the intersection of distributed systems, energy efficiency, and machine learning. His recent research explores the energy efficiency and performance optimization of data-intensive and distributed AI applications.

Maria Rodriguez Read is a lecturer in the School of Computing and Information Systems, University of Melbourne, Australia. Her research interests lie in the field of distributed and parallel systems. Her recent research works involve investigating how containerized and cloud-native applications can be better supported by cloud providers to offer users advantages in terms of reduced cost and more scalable, robust, and flexible application deployment.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored over 800 publications and seven textbooks. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=171, g-index=374, 156,100+ citations).