CPU Frequency Scaling Optimization in Sustainable Edge Computing

Yu Luo, *Member, IEEE,* Lina Pu, *Member, IEEE,* and Chun-Hung Liu, *Senior Member, IEEE*

***Abstract*— Sustainable edge computing (SEC) is a promising technology that can reduce energy consumption and computing latency for the mobile Internet of things (IoT). By collecting renewable energy such as solar or wind energy from the environment, a sustainable cloudlet outside the electric grid can provide powerful computing capabilities for resource-constrained mobile IoT devices. In the real world, the density of sustainable energy can vary significantly over time. Therefore, the SEC cloudlet needs to dynamically adjust the clock frequency to balance energy consumption and computing latency. In this paper, we consider the limited energy storage of the cloudlet and the dynamic intensity of renewable energy, and then develop offline optimal CPU frequency scaling policies that (a) maximize the computing power of the cloudlet within a certain period of time, and (b) minimize the execution time given tasks offloaded to the cloudlet. An optimal tightest string policy is proposed to solve the optimization problem. In addition, a dynamic programming (DP) based suboptimal solution is introduced to simplify the practical implementation. How to design an online CPU frequency management strategy is also briefly discussed.**

***Index Terms*—Sustainable edge computing (SEC), cloudlet CPU frequency scaling, energy harvesting, mobile Internet of things (IoT)**

I. Introduction

In recent years, mobile edge computing (MEC) is emerging as a new computing paradigm for the mobile Internet of things (IoT) [1]. By deploying small-scale servers, called cloudlets, at the edge of the Internet, IoT devices can offload computing tasks to the cloudlet for preprocessing, thereby significantly reducing response time and saving network bandwidth between the IoT network and cloud servers.

Current research on MEC assumes that the cloudlet is always connected to the electric grid. With unlimited energy supply, the current research on task offloading has been focused on making offloading decisions based on the time-varying wireless channel and stochastic computing capability of mobile edge and cloudlets. The optimal offloading strategies were proposed to dynamically manage the task allocation and computing resources aiming at minimizing the overall latency [2]–[5], minimizing the local energy consumption [6]–[8], or seeking a tradeoff between the latency and energy consumption [9]–[11].

One essential difference between our work and conventional MEC is that we consider the constraints of limited energy in sustainable edge computing, where renewable energy is the power supply to cloudlets. As will be introduced in the paper, it is realistic to use solar or wind energy to power a high-performance cloudlet in the wild. With the energy harvesting capability, cloudlets can be deployed outside the coverage of the electric grid, which will greatly improve the scalability and sustainability of existing mobile edge computing enabling compute-intensive tasks to run on resource-constrained IoT in the wilderness, forest, and ocean environments [12].

Aside from the improved sustainability and scalability, the renewable energy supply brings new challenges to the computing power management for cloudlets. Compared to conventional HPCs, sustainable cloudlets tend to have very limited computing capacity. In addition, different from the constant power supply from power-grid, the power density of wind and solar energy is temporally dynamic and spatially heterogeneous [13]. Therefore, the energy harvested from the environment may not always allow cloudlets to run at full speed. Given the dynamics in the energy environment, how to manage the power consumption of cloudlets in order to maximize computing capacity needs to be carefully studied.

In order to effectively use the harvested energy, the clock frequency of the central processing unit (CPU) needs to be adjusted carefully since it determines the computing power of the cloudlet. Generally, the energy consumed by the processor in a clock cycle is approximately proportional to the square of CPU clock frequency [14]. As a result, operating at high clock frequency with high computing power is not energy efficient. If the cloudlet has unlimited energy storage and the task has no deadline, the CPU should always run at the lowest clock frequency for higher efficiency of energy utilization.

After considering the limitations of task deadlines and energy storage capacity, optimal computing power management becomes a complex problem. Specifically, if the cloudlet keeps running at minimum power, the energy storage may overflow, causing waste of renewable energy or missing the mission deadlines. Neither is acceptable, as the former reduces the efficiency of energy harvesting, while the latter introduces unexpected latency in computing. Therefore, the cloudlet needs to manage its computing power to adapt to the fluctuations in energy strength, which is a unique problem in sustainable edge computing.

In this paper, we developed two optimal offline CPU frequency scaling policies for the SEC cloudlet. In the first policy, we aim at optimizing the overall computing power so that the total number of computing tasks that can be completed by cloudlet in a specific time period is maximized. In the second policy, we optimize the time required for the cloudlet to complete a certain number of tasks so that the computing latency of the IoT network is minimized.

To achieve the above objectives, we build a model that can convert the CPU frequency management into an optimization problem with a concave objective function and several convex constraints. The constraints can prevent the cloudlet from violating the energy storage limitation and the energy

Y. Luo and C.-H Liu are with the Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS, 39759. e-mail: yu.luo@ece.msstate.edu; chliu@ece.msstate.edu

L. Pu is with Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487. e-mail: lina.pu@ua.edu

causality constraint (sustainable energy cannot be used before it arrives). We formulate the optimization problem in a concise and tractable way such that the Karush-Kuhn-Tucker (KKT) conditions can be employed to find its solution.

To give insight into the efficient frequency management, the tightest string and the directional water-filling strategy are adopted to solve the optimization problem from a graphical perspective and an algorithmic viewpoint, respectively. Moreover, a dynamic programming (DP) based computing method is introduced to convert the optimal CPU frequency management into a shortest path problem. This conversion simplifies the frequency scaling optimization problem and make it solvable using classic DP algorithms (e.g., Dijkstra).

Finally, we briefly introduce an online CPU frequency management strategy based on the optimal offline policy and the prediction value of the energy that can be harvested by the cloudlet in the near future. As will be shown in the paper, the performance of the online strategy depends on the prediction accuracy of the energy intensity and the variation of the sustainable energy in the surrounding environment.

To summarize, the major contributions of our work are summarized as follows:

a) We consider sustainable edge computing, where the cloudlets are powered by renewable energy. A system model is developed to optimize the CPU frequency scaling for sustainable cloudlet considering the dynamic power intensity of renewable energy. To the best of our knowledge, this is the first work for CPU frequency scaling optimization for sustainable cloudlets.

b) Based on the developed system model, we formulate the CPU frequency scaling into two optimization problems aiming to 1) maximize the computing capacity of cloudlets given constrained renewable energy and 2) minimize the execution time for a given computing task.

c) We provide an offline optimal solution, namely the tightest string policy, and a DP-based suboptimal solution for the CPU frequency scaling optimization problem in SEC. The DP method converts the complex optimization problem into a classic shortest path problem and significantly simplifies the implementation compared to the tightest string policy. Simulation results are provided to verify the theoretical analysis as well as the performance improvement compared to the benchmark policies.

d) The prediction-based online CPU frequency scaling strategy is briefly introduced. How the accuracy of energy prediction and changes in energy intensity affect the performance of the online strategy is evaluated carefully.

The rest of the paper is organized as follows: Section II introduces the related work. The system model of the CPU frequency management for the SEC cloudlet is introduced in Section III. In Section IV, we formulate the computing power maximization problem. The theoretical solution and suboptimal DP solution are provided in Sections V and VI. How to minimize the execution time is studied in Section VII. We evaluate the performance of the proposed CPU frequency scaling policies in Section VIII and conclude our work in Section IX.

II. Related Work

MEC can significantly enhance the computing capability of resource-constrained IoT devices by offloading computing tasks to cloudlets. Extensive research has been conducted to develop practical energy consumption models [15]–[17], develop optimal CPU frequency scaling and task offloading policies [8]–[10], and investigate MEC for renewable energy powered edge devices [18]–[20].

In MEC, by offloading all or partial computation tasks to the cloudlet [21], IoT devices can save a significant amount of energy on computation but at the cost of higher execution latency. The total execution delay is composed of the local computing latency, transmission delay, and computing latency on the cloudlet. The quality of the wireless channel and the CPU frequency of mobile devices and cloudlets as well as the task allocation strategy determine the execution latency and the energy consumption. The task offloading and CPU scaling strategy needs to comprehensively consider the quality of the wireless channel, the power consumption of communications and computations, the deadline of each task, and the topology of the tasks (sequential, parallel, or general dependency [22], [23]).

When a task is executed locally, the execution latency and energy consumption can be adjusted by controlling the CPU frequencies of mobile devices with dynamic voltage and frequency scaling (DVFS) techniques [8], [10], [18], [19]. In the scenario when the tasks are offloaded to the cloudlets, the transmission delay becomes the dominant factor in the total execution latency. In the literature, various wireless channels (e.g., Block fading [18], Raleigh fading [8]) and communication systems (e.g., TDMA [19], FDMA [7], NOMA [19]) are considered for optimal task allocation. Besides offloading tasks to a single cloudlet [18], a more complicated scenario where one edge device offloads tasks to M MEC servers is investigated in [10] and the scenario with N edge devices offloading tasks to one MEC server is studied in [7].

In [18]–[20], the mobile edge with energy harvesting capabilities is considered. [18] jointly optimizes the computing power and communication rate of edge devices with respect to battery level. Besides the execution latency, the authors also considered the execution cost caused by task failure when there is no sufficient energy on the edge devices. In [19], the authors consider an application scenario where the edge devices harvest radio energy from a dedicated power station. Both the binary offloading and the partial offloading schemes are studied to maximize the computation efficiency of mobile edge devices. [20] investigates when to switch ON and OFF of fog devices considering the integration of renewable energy to reduce energy consumption and execution latency.

In the existing research, there are several underlying assumptions: 1) The cloudlets or edge/cloud servers have significantly higher computing resources than mobile edges; 2) Unlike mobile edges that need to adjust the computing power and transmission rate for energy efficiency considerations, cloudlets have no energy constraints (e.g., connected to the power grid) and thus can always operate at the maximum power in order to achieve low task execution latency.

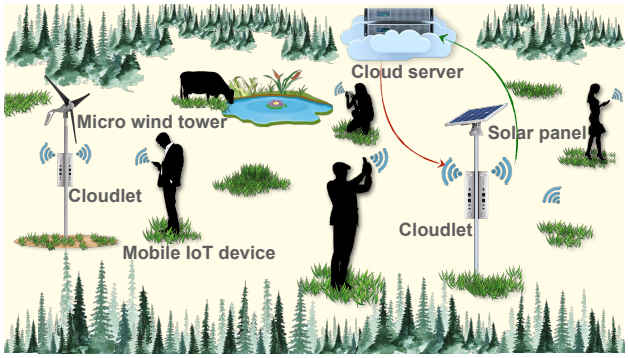


Figure 1: Network architecture of SEC with mobile IoT devices.

However, in sustainable edge computing, where the cloudlets are powered by renewable energy, the aforementioned assumptions do not hold due to the dynamic energy intensity of renewable energy. How to optimize the CPU frequency of the cloudlet in a dynamic energy environment has not yet been studied, which is the focus of this paper. Note that the proposed optimal CPU frequency scaling strategy for sustainable cloudlets can be integrated with existing task offloading [7], [8], [10], [18], [19] solutions making them more practical and efficient in sustainable edge computing.

III. System Model

In this section, we first briefly introduce the SEC architecture with mobile IoT devices, and then propose two optimization problems that optimize the performance of computation-intensive IoT networks and the latency-critical IoT networks.

A. Network Architecture

We consider a mobile IoT network, where many mobile devices are running computation-intensive applications in the wild, such as AR and target recognition. Due to size, energy, and heat dissipation limitations, performing all tasks locally is inefficient. In order to reduce energy consumption and computing latency, mobile devices offload their tasks to nearby sustainable cloudlets, as shown in Fig. 1.

Different from mobile IoT devices, the cloudlet can have a relatively large size so that it can scavenge solar or wind energy from the surrounding environment to power high-performance CPUs. Taking wind energy as an example, even with a micro wind turbine (weight: 12 kg, rotor-swept area: 1.2 m^2), it can harvest energy at 177 W and 524 W power when the wind speeds are 11 m/s (24.6 mph) and 20 m/s (44.7 mph), respectively [24]. These energy harvesting rates are sufficient to drive high-performance server processors, such as Intel Xeon Gold 6328HL [25] or AMD EPYC 7501 [26], the thermal design point (TDP) of which are 165 W and 175 W, respectively.

Taking solar energy as another example, the peak intensity of solar energy in non-shaded areas can reach 600 W/m^2 [27]. The power conversion efficiency of commercial solar panels is between 15% and 20% [28]. Therefore, a two square meters (1.4 m×1.4 m) solar panel can generate 180 W to 240 W of power, which can easily drive high-performance CPUs.

After receiving computing tasks from mobile devices, the cloudlet will preprocess the raw data (e.g., feature extraction or data compression), and then upload useful information to the cloud server via satellite internet constellation (e.g., Starlink by SpaceX [29]) for further data processing, or directly download computing results to IoT devices. With the assistance of the cloudlet, internet traffic can be greatly alleviated and the workload of mobile devices can be significantly reduced.

B. Energy Model

Denote the clock frequency of the cloudlet CPU at time t by $f_c(t)$. According to processor design [30], the CPU's power consumption can be divided into three parts: the short-circuit power, the transistor leakage power, and the dynamic power, where the last part dominates the others when the CPU is running. Therefore, we use the dynamic power to approximate the total power consumption of the cloudlet.

According to the CMOS circuit theory [14], the dynamic power at time t, denoted by $P_d(t)$, can be calculated as

$$P_d(t) = \alpha f_c(t) V_c^2(t), \tag{1}$$

where α is a constant related to the processor architecture and $V_c(t)$ is the CPU power supply voltage at time t. Furthermore, f_c is proportional to V_c [30]. By adopting DVFS technology, modern processors can dynamically scales down the voltage based on the frequency requirement [31]. Consequently, the CPU clock frequency can be written as a function of the dynamic power:

$$f_c(t) = \beta P_d^{\frac{1}{3}}(t), \quad P_d \geq 0, \tag{2}$$

where β is the frequency scaling coefficient and it is a positive constant.

In nature, the power density of sustainable energy changes over time. Let $p_h(t)$ be the incident power at time t. According to the energy causality constraint, the cloudlet cannot use the energy that has not yet arrived. As a result, the total energy consumed by cloudlet[1] cannot exceed the cumulative harvested energy, i.e.,

$$\int_0^t P_d(u)\,\mathrm{d}u \leq \int_0^t p_h(u)\,\mathrm{d}u, \qquad \forall t \geq 0. \tag{3}$$

Another constraint is resulted from the limited capacity of the energy storage, E_{max}. To avoid harvested energy overflow from the energy storage, the difference between the cumulative energy harvested from the beginning to any time point and the total energy consumed by the cloudlet during that time period cannot be larger than E_{max}, i.e.,

$$\int_0^t p_h(u)\,\mathrm{d}u - \int_0^t P_d(u)\,\mathrm{d}u \leq E_{max}, \quad \forall t \geq 0. \tag{4}$$

C. Optimization Problem Forumation

The computing power of the cloudlet linearly increases with the CPU clock frequency since the clock cycle needed to

[1] A more general power model that considers the static system power and CPU leakage power could be considered [16], [17]. As such, $P_d(u)$ in the energy causality constraint and battery capacity constraint can be replaced by the total power consumption of CPU.

execute a machine code is fixed. Without loss of generality, assume that it takes CPU an average of one clock cycle to execute a machine code, and then according to the relationship between f_c and P_d given in (2), we can write the following two optimization problems under the constraints of energy causality and energy storage capacity:

Computing power maximization: The computing power of cloudlets determines the execution latency in SEC. Unlike conventional MEC, where cloudlets are assumed to have fixed computing capacity, renewable energy-powered cloudlets have time-varying computing power depending on the intensity of harvested energy. In the computing power maximization problem, we manage the dynamic power of the CPU in order to maximize the overall computing capacity of the cloudlet in a certain time period, $[0,\, T_p]$:

$$\begin{aligned}
&\textbf{\textit{P1:}}\ \max_{P_d(t)\geq 0} \int_0^{T_p} \beta P_d^{\frac{1}{3}}(t)\,\mathrm{d}t, && T_p>0,\\
\textbf{s.t.}\ &\textbf{\textit{C1:}}\ \int_0^t P_d(u)\,\mathrm{d}u \leq \int_0^t p_h(u)\,\mathrm{d}u, && \forall t\in[0,T_p],\\
&\textbf{\textit{C2:}}\ \int_0^t p_h(u)\,\mathrm{d}u - \int_0^t P_d(u)\,\mathrm{d}u \leq E_{max}, && \forall t\in[0,T_p].
\end{aligned} \tag{5}$$

Execution time minimization: It minimizes the time for the cloudlet to execute a total number of M_p machine codes:

$$\begin{aligned}
&\textbf{\textit{P2:}}\ \min_{P_d\geq 0}\ T_p, && T_p>0,\\
\textbf{s.t.}\ &\textbf{\textit{C1:}}\ \int_0^{T_p} \beta P_d^{\frac{1}{3}}(u)\,\mathrm{d}u = M_p,\\
&\textbf{\textit{C2:}}\ \int_0^t P_d(u)\,\mathrm{d}u \leq \int_0^t p_h(u)\,\mathrm{d}u, && \forall t\in[0,T_p],\\
&\textbf{\textit{C3:}}\ \int_0^t p_h(u)\,\mathrm{d}u - \int_0^t P_d(u)\,\mathrm{d}u \leq E_{max}, && \forall t\in[0,T_p].
\end{aligned} \tag{6}$$

The objective function of **P1** enables the cloudlet to achieve the best computing performance in a certain period of time. Therefore, the CPU frequency scaling policy obtained from **P1** is suitable for IoT networks that run computation-intensive applications. In contrast, the objective function of **P2** allows the cloudlet to complete a certain number of tasks in the shortest time. As will be introduced in Section VII, the optimization problem **P2** can be converted into **P1**. The solution of **P2** can be used to estimate the minimum computing latency to finish specific tasks offloaded from mobile IoT devices.

IV. Processing Power Optimization

Based on the optimization problem **P1** given in Section III-B, this section studies how to manage the CPU clock frequency in order to maximize the computing power of the cloudlet within a certain period of time.

To solve **P1**, we first discretize p_h in (5). Let Δt represent a short time period, which is an aliquot part of T_p; i.e., T_p can be divided by Δt. The time between $[n\Delta t, (n+1)\Delta t]$ is referred to as the time slot n, which is represented by t_n. When Δt is small, it is reasonable to assume that the incident power of sustainable energy remains constant within a time slot, then we have that

$$p_h(t) = p_h[n], \quad t\in[n\Delta t,(n+1)\Delta t)\,,\ n=0,1,2\ldots. \tag{7}$$

Let $E_h[n]\triangleq \Delta t\, p_h[n]$ be the energy received by the cloudlet in the n^{th} time slot, then the accumulative energy harvested by the cloudlet in $[0,t]$ is

$$\int_0^t p_h(u)\,\mathrm{d}u = \sum_{i=0}^{n} E_h[i], \quad t\geq 0,\ n=0,1,2\ldots. \tag{8}$$

By substituting (8) into the constraints of (5), we can obtain the following Lemma and Corollary:

Lemma 1. *Under the optimal policy, the CPU clock frequency remains unchanged within a time slot.*

Corollary 1. *For a given total amount of energy consumed in a certain time period, the computing power can be maximized if the CPU clock frequency remains unchanged.*

Proof. As shown in (2), the CPU clock frequency is a concave function of the dynamic power. Therefore, the proof of Lemma 1 and Corollary 1 can refer to the proof of inequality (2.8) in the BT-problem of [32]. □

According to Lemma 1, the optimal clock frequency of the CPU in time slot n can be expressed in a discrete form given by:

$$f_c^*(t) = f_c^*[n],\ t\in[n\Delta t,(n+1)\Delta t)\,,\ n=0,1,2\ldots. \tag{9}$$

In addition, from Lemma 1 and (2), it can be realized that $P_d(t)$ in (5) becomes a piece-wise linear function of t. Let $P_d[n]$ represent $P_d(t)$ at time $n\Delta t$, then we have that

$$f_c[n] = \beta P_d^{\frac{1}{3}}[n], \qquad n=0,\ldots,N_p, \tag{10}$$

where $N_p = T_p/\Delta t - 1$.

Through the above discretization process, the continuous optimization problem **P1** can be converted into a piece-wise optimization problem:

$$\begin{aligned}
&\textbf{\textit{P3:}}\ \max_{P_d[i]\geq 0} \sum_{i=0}^{N_p} \beta P_d^{\frac{1}{3}}[i]\Delta t, && N_p=0,1,2\ldots,\\
\textbf{s.t.}\ &\textbf{\textit{C1:}}\ \sum_{i=0}^{n} P_d[i]\,\Delta t \leq \sum_{i=0}^{n} E_h[i], && n=0,\ldots,N_p,\\
&\textbf{\textit{C2:}}\ \sum_{i=0}^{n} E_h[i] - \sum_{i=0}^{n} P_d[i]\,\Delta t \leq E_{max}, && n=1,\ldots,N_p,
\end{aligned} \tag{11}$$

In the optimization problem **P3**, the objective function is concave because it is a linear combination of concave functions. In addition, **C1** and **C2** in (11) are composed of linear constraints, so they are convex. Consequently, there exist KKT multiplier sets $\boldsymbol{\mu} = \{\mu_0,\ldots,\mu_{N_p}\}$ and $\boldsymbol{\lambda} = \{\lambda_1,\ldots,\lambda_{N_p+1}\}$ to make the following conditions hold:

Stationarity:

$$\nabla_{P_d^*[j]}\mathcal{L} = \nabla_{P_d^*[j]}\left(\sum_{i=0}^{N_p}\beta P_d^{\frac{1}{3}}[i]\Delta t\right) - \sum_{n=0}^{N_p}\mu_n\nabla_{P_d^*[j]}\left(\sum_{i=0}^{n}P_d[i]\,\Delta t - \sum_{i=0}^{n}E_h[i]\right) - \sum_{n=1}^{N_p}\lambda_n\nabla_{P_d^*[j]}\left(\sum_{i=0}^{n}E_h[i] - \sum_{i=0}^{n}P_d[i]\,\Delta t - E_{max}\right) = 0, \tag{12}$$

where $\mathcal{L}$ is the Lagrangian function and $\nabla_x(\cdot)$ represents the partial derivative with respect to x.

Complementary slackness:

$$\mu_n\left(\sum_{i=0}^{n}P_d[i]\,\Delta t - \sum_{i=0}^{n}E_h[i]\right) = 0, \qquad n = 0,\ldots,N_p \tag{13}$$

$$\lambda_n\left(\sum_{i=0}^{n}E_h[i] - \sum_{i=0}^{n}P_d[i]\Delta t - E_{max}\right) = 0,\ n = 1,\ldots,N_p \tag{14}$$

Dual feasibility:

$$\begin{cases}\mu_n \geq 0, & n = 0,\ldots,N_p,\\ \lambda_n \geq 0, & n = 1,\ldots,N_p.\end{cases} \tag{15}$$

In the following two sections, we will introduce how to use the KKT conditions to find the optimal solution to **P3** from the graphical perspective and the algorithmic viewpoint.

V. Graphical Perspective of Problem P3

This section studies how to adjust the CPU clock frequency from the graphical point of view to maximize the computing power of the cloudlet in a certain period of time. We first construct an feasible energy tunnel based on the constrains of energy causality and energy storage capacity. Afterward, the solution to **P3** is given. Then, the optimal policy, called the tightest string policy, is introduced to efficiently manage the CPU frequency. Finally, we introduce a suboptimal DP-based method for practical implementation.

A. Feasible energy Tunnel

Fig. 2 illustrates a feasible energy tunnel, where the upper bound represents the accumulative harvested energy (i.e., $\sum_{i=0}^{n}E_h[i]$) and the tunnel width is the battery capacity, E_{max}. X-axis and Y-axis of Fig. 2 are the time and the accumulative energy. We call the curve, $\sum_{i=0}^{n}P_d[i]\,\Delta t$, the dynamic energy curve which represents the accumulative energy consumed by the cloudlet. According to the constraints of **P3**, the dynamic energy cannot fall outside the tunnel: If it excesses the upper bound of the tunnel, the energy causality constrain will be violated; if it is below the lower bound of the tunnel, the constraint of the energy storage capacity will not hold.

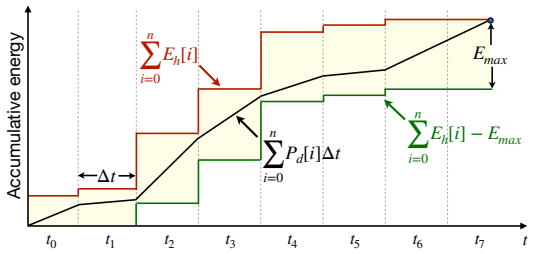


Figure 2: Feasible energy tunnel

The dynamic energy curve is admissible if it is in the feasible energy tunnel. In the optimization problem **P3**, we aim at finding an admissible curve to maximize $\mathscr{F}(P_d)$, where

$$\mathscr{F}(P_d) = \sum_{i=0}^{N_p}\beta P_d^{\frac{1}{3}}[i], \qquad N_p = 0,1,2\ldots. \tag{16}$$

Since the dynamic energy consumed by the CPU increases monotonically with time, we can obtain the following Corollary according to Lemma 1:

Corollary 2. *Under the optimal policy, the dynamic energy curve touches neither the upper bound nor the lower bound of the feasible energy tunnel in a time slot.*

Proof. The dynamic energy increases monotonically with time. Therefore, if the dynamic energy curve touches the lower bound of the feasible energy tunnel at time T_l, where $T_l \in (m\Delta t,\,(m+1)\Delta t)$, the curve must be below the lower bound of the feasible energy tunnel between $(m\Delta t,\,T_l)$, which violates the constraint of energy storage capacity, as shown in curve (e) of Fig. 3.

According to Lemma 1, if the optimal dynamic energy curve reaches the upper bound of the feasible energy tunnel at time T_u, then the curve would exceed the upper bound of the tunnel during $(T_u,\,(m+1)\Delta t)$, which violates the energy causality constraint, as shown in curve (d) of Fig. 3. □

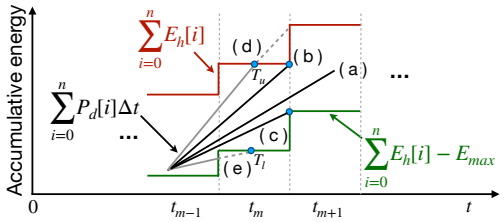


Figure 3: Three possible states of the dynamic energy curve.

According to Corollary 2 and the energy causality and storage capacity constrains, the optimal dynamic energy curve has only three potential states at the end of t_m: (a) passing through the energy feasible tunnel, (b) reaching the upper bound of the tunnel, and (c) touching the lower bound of the tunnel, as shown in Fig. 3. States (d) and (e) will not occur.

B. Solution of Precessing Power Optimization

By solving the KKT stationarity condition in (12), we have:

$$P_d^*[i] = \left[\frac{3}{\beta}\left(\sum_{n=i}^{N_p}\mu_n - \sum_{n=i}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}}, N_p = 0, 1, 2 \ldots. \quad (17)$$

Combining (17) with the complementary slackness and dual feasibility of KKT conditions, we obtain following Lemmas:

Lemma 2. *Under the optimal policy, the CPU clock frequency remains unchanged when the energy storage is neither full nor empty (i.e.,* $0 < \sum_{i=0}^{m}(E_h[m] - P_d[i]\Delta t) < E_{max}$, $\forall\, m \in [0, N_p - 1]$: $f_c^*[m] = f_c^*[m+1]$*).*

Proof. The dynamic energy curve in Lemma 2 corresponds to the state (a) in Fig. 3. In this state, we have that

$$\begin{cases} \sum_{i=0}^{m} P_d[i]\,\Delta t - \sum_{i=0}^{m} E_h[i] \neq 0, & (18) \\ \sum_{i=0}^{m} E_h[i] - \sum_{i=0}^{m} P_d[i]\Delta t - E_{max} \neq 0. & (19) \end{cases}$$

Substituting (18) and (19) into the complementary slackness of the KKT conditions, (13) and (14), it can be obtained that $\mu_m = 0$ and $\lambda_m = 0$. Then, according to (17), it can be obtained that

$$\begin{aligned} P_d^*[m] &= \left[\frac{3}{\beta}\left(\mu_m + \sum_{n=m+1}^{N_p}\mu_n - \lambda_m - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ &= \left[\frac{3}{\beta}\left(\sum_{n=m+1}^{N_p}\mu_n - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ . &= P_d^*[m+1]. \end{aligned} \quad (20)$$

From (20) and the relation between the CPU clock frequency and the dynamic power given in (10), we have that $f_c^*[m+1] = f_c^*[m]$. □

Lemma 3. *Under the optimal policy, the CPU clock frequency increases monotonically when the energy storage becomes empty (i.e.,* $\sum_{i=0}^{m} P_d[i]\Delta t = \sum_{i=0}^{m} E_h[m]$, $\forall\, m \in [0, N_p - 1]$: $f_c^*[m+1] \geq f_c^*[m]$*).*

Proof. The dynamic energy curve in Lemma 3 corresponds to the state (b) in Fig. 3. In this state, we have that

$$\begin{cases} \sum_{i=0}^{m} P_d[i]\,\Delta t - \sum_{i=0}^{m} E_h[i] = 0, & (21) \\ \sum_{i=0}^{m} E_h[i] - \sum_{i=0}^{m} P_d[i]\Delta t - E_{max} \neq 0. & (22) \end{cases}$$

Substituting (22) into the complementary slackness of the KKT conditions given in (14), it can be obtained that $\lambda_m = 0$. Substituting (21) into (13), and then according the dual feasibility of the KKT conditions given in (15), we have that $\mu_m \geq 0$. Fianlly, according to (17), it can be obtained that

$$\begin{aligned} P_d^*[m] &= \left[\frac{3}{\beta}\left(\mu_m + \sum_{n=m+1}^{N_p}\mu_n - \lambda_m - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ &\leq \left[\frac{3}{\beta}\left(\sum_{n=m+1}^{N_p}\mu_n - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ &= P_d^*[m+1]. \end{aligned} \quad (23)$$

In (23), there is an inequality because $\mu_m \geq 0$ and $x^{-\frac{3}{2}}$ is a monotonically decreasing function of x.

From (23) and the relation between the CPU clock frequency and the dynamic power described in (10), we have $f_c^*[m+1] \geq f_c^*[m]$. □

Lemma 4. *Under the optimal policy, the CPU clock frequency decreases monotonically when the energy storage becomes full (i.e.,* $\sum_{i=0}^{m} P_d[i]\Delta t = \sum_{i=0}^{m} E_h[m] + E_{max}$, $\forall\, m \in [0, N_p - 1]$: $f_c^*[m+1] \leq f_c^*[m]$*).*

Proof. The dynamic energy curve in Lemma 4 corresponds to the state (c) in Fig. 3. In this state, we have that

$$\begin{cases} \sum_{i=0}^{m} P_d[i]\,\Delta t - \sum_{i=0}^{m} E_h[i] \neq 0, & (24) \\ \sum_{i=0}^{m} E_h[i] - \sum_{i=0}^{m} P_d[i]\Delta t - E_{max} = 0. & (25) \end{cases}$$

Substituting (24) into the complementary slackness of the KKT conditions given in (13), it can be obtained that $\mu_m = 0$. Substituting (25) into (14), and then according to the dual feasibility of the KKT conditions given in (15), we have that $\lambda_m \geq 0$. Finally, according to (17), it can be obtained that

$$\begin{aligned} P_d^*[m] &= \left[\frac{3}{\beta}\left(\mu_m + \sum_{n=m+1}^{N_p}\mu_n - \lambda_m - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ &\geq \left[\frac{3}{\beta}\left(\sum_{n=m+1}^{N_p}\mu_n - \sum_{n=m+1}^{N_p}\lambda_n\right)\right]^{-\frac{3}{2}} \\ &= P_d^*[m+1]. \end{aligned} \quad (26)$$

In (26), there is an inequality because $\lambda_m \geq 0$ and $x^{-\frac{3}{2}}$ is a monotonically decreasing function of x.

From (26) and the relation between the CPU clock frequency and the dynamic power given in (10), we have $f_c^*[m+1] \leq f_c^*[m]$. □

Lemma 5. *Under the optimal policy, the cloudlet consumes all the harvested energy by the end of the last slot (i.e.,* $\sum_{i=0}^{N_p} P_d^*[i]\,\Delta t = \sum_{i=0}^{N_p} E_h[i]$*).*

Proof. If energy is not exhausted in the last time slot with the optimal $P_d^*[i]$, $i = 1, \ldots, N_p$, we can always find $P_d'[N_p] >$

$P_d^*[N_p]$ that consumes all the collected energy. Because $\mathscr{F}$ in (16) is a monotonically increasing function of P_d, we thus have $\mathscr{F}(P_d'[N_p]) > \mathscr{F}(P_d^*[N_p])$, which means that $P_d^*[N_p]$ is not optimal. Therefore, the optimal policy must consume all harvested energy in the last time slot. □

C. Tightest String Policy

In this subsection, we present a tightest string policy utilizing the proved Lemmas and Corollaries to optimize the computing power for the cloudlet.

As shown in Fig. 4, we first mark several turning points in the feasible energy tunnel as p_i, $i = 0, \ldots, 8$, where p_0 is the starting point. The optimal dynamic energy curve can be obtained through the following steps.

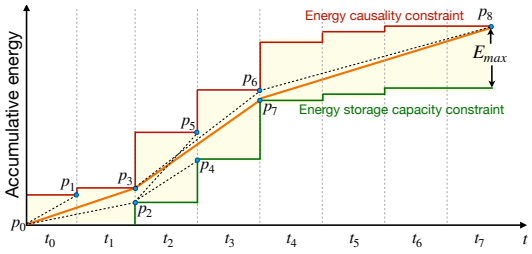


Figure 4: The tightest string policy for computing capacity maximization.

Step 1: Connect p_0 with all turning points in the feasible energy tunnel. Remove the strings that have any portion fall outside the tunnel. The rest of the strings, which are $\overline{p_0p_1}$, $\overline{p_0p_2}$, and $\overline{p_0p_3}$, are considered as admissible starting strings.

Step 2: According to Corollary 2, amongst all admissible starting strings, we keep the one with the longest duration and remove others. If two starting strings have the longest duration, such as $\overline{p_0p_2}$ and $\overline{p_0p_3}$ in Fig. 4, and then go to the next step to examine each retained string.

Step 3: We check $\overline{p_0p_2}$ first. Let p_2 be the new starting point, and then repeat Step 1 and Step 2 to obtain all admissible strings, $\overline{p_2p_4}$ and $\overline{p_2p_5}$. $\overline{p_0p_2}$ hits the lower bound of the feasible energy tunnel, which means the energy storage is full. In this case, according to Lemma 4, the CPU will reduce the frequency and the dynamic power in the next slot. As a result, the slope of $\overline{p_2p_i}$ must be smaller than that of $\overline{p_0p_2}$. However, it can be observed from Fig. 4 that the slopes of $\overline{p_2p_4}$ and $\overline{p_2p_5}$ are both greater than the slop of $\overline{p_0p_2}$. Therefore, $\overline{p_0p_2}$ needs to be removed from the admissible strings, and only $\overline{p_0p_3}$ is retained.

Step 4: Let p_3 be the new starting point, and then we repeat Step 1 and Step 2 to obtain all admissible strings, which are $\overline{p_3p_6}$ and $\overline{p_3p_7}$. We first check $\overline{p_3p_6}$. According to Lemma 5, the cloudlet must spend all received energy at the end of the time slot. Therefore, the end point of the dynamic energy curve is p_8. As shown in the figure, $\overline{p_3p_6}$ reaches the upper bound of the feasible energy tunnel. In this case, the CPU will increase the frequency and the dynamic power in the next slot based on Lemma 3. Therefore, the slope of $\overline{p_6p_i}$ should be greater than that the slope of $\overline{p_3p_6}$, which is unsatisfactory. Therefore, $\overline{p_3p_6}$ is removed from the admissible strings. Finally, $\overline{p_3p_7}$ and $\overline{p_7p_8}$ are retained as the optimal solution.

Step 5: After obtaining the optimal strings through Step 1 to Step 5, the optimal dynamic power of the CPU in each time slot is available, which is the slope of the strings. Finally, the optimal CPU frequency can be calculated by (10).

The optimal strings obtained through the above steps are the tightest that follows the Lemmas. The corresponding CPU frequency scaling policy is called the tightest string policy. To be specific, assume that a thread ball is placed in the feasible energy tunnel. We tie one end of the thread ball to the starting point p_0, and then withdraw the thread at the endpoint p_8. The process will not stop until the thread is fully tightened. Finally, the thread left in the tunnel has the shortest length and the tightest shape. A similar observation can also be found in the transmission scheduling of wireless communications with deadline constraints [32].

D. Dynamic Programming Solution

Although the tightest string method leads to an optimal solution, the implementation of Step 1 to Step 5 is not straightforward as it is not easy to find the optimal string under the constraints of Lemma 3 and Lemma 4. In this subsection, we introduce a dynamic programming method for practical implementation. Specifically, the complex optimization problem is converted into a classic shortest path problem with negligible performance degradation and thus can be solved by DP algorithms [33]. Not that the DP method is a general and fast solution that can be easily applied to different power models in the literature [15]–[17].

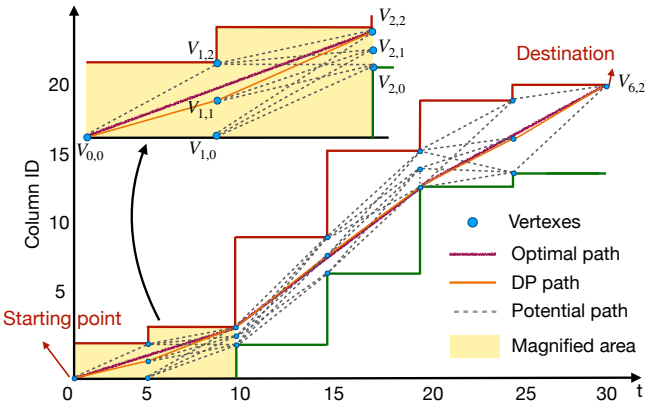


Figure 5: Dynamic programming based frequency management.

In the DP solution, finding the optimal frequency management strategy within the energy tunnel is converted into seeking the shortest path from a Starting point to a Destination. According to Lemma 2 and the inspiration of the tightest string policy, the dynamic energy curve needs to stay within the feasible energy tunnel; in addition, the dynamic power will not change within a time slot. Therefore, we evenly place k "vertexes" between the lower bound and the upper bound of the feasible energy tunnel at the end of each time slot (k=3 in Fig. 5). The vertexes are fully connected with the vertexes at adjacent slots, forming directed edges in the weighted graph. The slope of the edge represents the dynamic power, P_d, of the CPU. The corresponding CPU frequency calculated with (2) is considered as the weight of the edge.

The Starting point, $V_{0,0}$, of the dynamic energy curve indicates that the cloudlet consumes zero energy at time 0. The Destination, $V_{6,2}$, means the cloudlet will consume all harvested energy at the end of task execution according to Lemma 5. When it moves along the edges toward the Destination, the cloudlet consumes dynamic power and gets the CPU frequency as weights of edges.

Following the above description, maximizing the computing capacity of the cloudlet in a certain period of time is converted into a routing problem with weighted rewards on different paths. We aim at finding a route between the starting point and the end point for the virtual point to maximize the total costs. To achieve this goal, DP-based methods like the Dijkstra's algorithm can be directly applied to find the optimal solution. Compared to the tightest string policy, the DP solution significantly simplifies the computing capacity maximization problem. The computational complexity of the Dijkstra's algorithm is $\mathcal{O}(kn\log(kn))$, where n is the number of slots in the feasible energy tunnel [34].

However, this simplification converting the tightest string to a DP problem is at the cost of minor performance degradation. As shown in the magnified area of Fig. 5, the optimal DP path, $V_{0,0}\rightarrow V_{1,1}\rightarrow V_{2,2}$, is slightly deviated from the optimal tightest string path, $V_{0,0}\rightarrow V_{2,2}$. By increasing vertex density (i.e., k), the results obtained from the DP method can gradually approach the optimal solution. In the next section, we evaluated the performance of the DP algorithm with respect to the grid density.

E. Performance of DP Algorithm

In Fig. 6, we evaluate the impact of grid density on the performance of the DP-based CPU scaling policy. The total running time of the cloudlet is $T_p = 600$ min. The maximum capacity of the energy storage is $E_{max} = 100$ Wh. The grid density on the Y-axis is defined as the number of vertexes distributed between the upper bound and the lower bound of the feasible energy tunnel. The normalized performance is the ratio of the total CPU clock cycles obtained by the DP algorithm to that achieved by the tightest string policy. The results presented in the figure are the average of 10 independent tests.

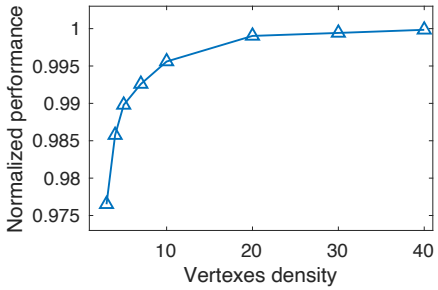


Figure 6: The normalized performance of the DP algorithm with respect to the grid density.

When $k = 3$, the performance degradation is about 2.3%. With the increment of grid density, the performance of the DP-based frequency scaling strategy gradually approaches the optimal tightest string policy. The performance gap is reduced to less than 1% when k is greater than 6. However, a larger k will result in higher computational complexity. Therefore, we recommend a moderate k between 3 and 6. The computational complexity of calculating the CPU frequency scaling policy is negligible considering the high computing capacity of cloudlets.

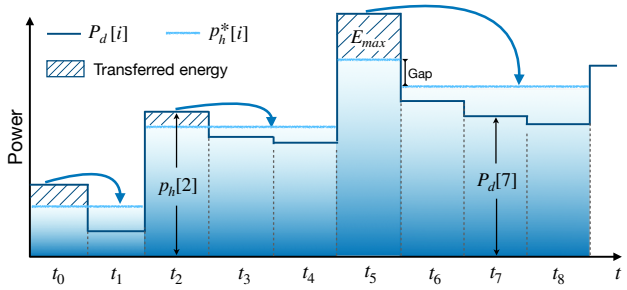


Figure 7: Directional water-filling algorithm for computing power optimization.

VI. Algorithm Perspective of Problem P3

In this section, we study how to manage the CPU frequency through the directional water-filling algorithm to maximize the computing power of the cloudlet in a certain period of time. It interprets the solution of the optimization problem **P3** from an algorithmic perspective.

The water-filling algorithm plays a crucial role in the optimization problem [35]. It has been widely used for resource (e.g., frequency and power) allocation, transceiver optimization, and training optimization in communication systems [36], [37]. The direction water-filling algorithm was first proposed in [38] to find the optimal data transmission strategy for energy harvesting devices in wireless fading channels. The algorithm can be extended to our work because it describes the flow of water in a tank under the constraints of energy causality and energy storage capacity.

As shown in Fig. 7, we first construct the original water surface, the depth of which represents the incident power of the received energy in each time slot. Accordingly, the area of water equals to the energy harvested by the cloudlet. Afterward, water is allowed to flow under the constraints of the following rules:

a) Water can only flow from the left to the right. This signifies that the past energy can be saved and used in the future, but due to the causality of energy, the future energy cannot be transferred to the past.

b) Due to the constraint of the energy storage capacity, the total amount of energy transferred from one slot to another cannot exceed E_{max}.

c) To maximize the objective function in **P3**, water flows continuously until the water level is equalized or the rule (b) is violated.

Now, we take Fig. 7 as an example to show how the directional water-filling algorithm works. As shown in the figure, according to rules (a) and (b), in order to equalize the water level, the energy received in t_0 is filled into t_1, and the

energy collected in t_2 flows to t_3 and t_4. In addition, as stated in rule (c), the water level from t_5 to t_8 cannot be completely balanced since the maximum energy allowed to flow from t_5 cannot exceed E_{max}. As a result, there is a gap between the water surface of t_5 and t_6.

Based on the three rules, a new water surface is formed. The depth of the new surface represents the optimal dynamic power in each time slot. Finally, the optimal clock frequency of the CPU can be calculated via (10). The results obtained from the directional water-filling algorithm are the same as the tightest string policy because they are different interpretations of the same optimization problem.

In Fig. 7, we can observe an interesting phenomenon: If the energy harvested by the cloudlet in a time slot, such as t_2 and t_5, is much higher than the energy collected in the past few slots, then an "energy dam" will be formed, which prevents the flow of energy from the past to the future. Once the energy dam is formed, it will serve as the new starting point of the frequency management. In other words, the CPU frequency scheduled before the dam will not affect the frequency scaling strategy after the dam.

The above feature is useful for developing an online CPU frequency scaling strategy. Specifically, the energy dam can separate long-term frequency management into multiple short-term ones. Therefore, if the cloudlet is able to predict energy variations in the near future, it can adjust the clock frequency based on the current and predicted energy harvesting rate to achieve sub-optimal computing performance.

VII. Execution Time Optimization

In this section, we study how to manage the CPU clock frequency to minimize the task execution time based on the optimization problem **P2** given in Section III-B. To solve **P2**, we first discretize p_h as introduced in Section IV. Then, according to the constraint **C1** in the optimization problem **P2**, we have that T_p is a convex function of $P_d[i]$, where $i = 0, \ldots, N_p$. Therefore, to minimize the execution time of tasks offloaded from mobile IoT devices, the CPU clock frequency remains unchanged in a time slot. This conclusion is consistent with Lemma 1.

Let Δt is an aliquot part of T_p and $N_p = T_p/\Delta t - 1$, then the continuous optimization problem **P2** can be converted into the following piece-wise optimization problem:

$$
\begin{aligned}
&\textbf{\textit{P4:}} \ \arg\min_{P_d[i]\geq 0} N_p, \\
&\textbf{s.t.} \quad \textbf{\textit{C1:}} \ \sum_{i=0}^{N_p} \beta P_d^{\frac{1}{3}}[i] = M_p, \\
&\qquad \textbf{\textit{C2:}} \ \sum_{i=0}^{n} P_d[i]\,\Delta t \leq \sum_{i=0}^{n} E_h[i], \qquad n=0,\ldots,N_p, \\
&\qquad \textbf{\textit{C3:}} \ \sum_{i=0}^{n} E_h[i] - \sum_{i=0}^{n} P_d[i]\,\Delta t \leq E_{max}, \quad n=1,\ldots,N_p.
\end{aligned}
\tag{27}
$$

We can solve **P4** based on the following Lemma:

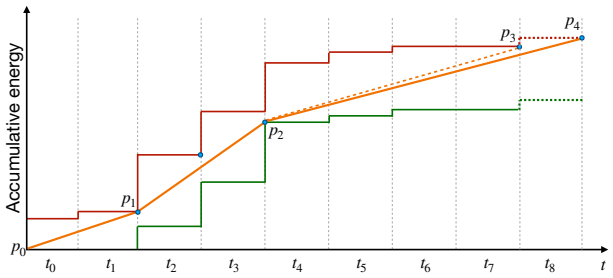


Figure 8: The impact of newly received energy on CPU frequency management.

Lemma 6. *$P_d^*[i]$ form the optimal policy of **P4** if they maximize $\sum_{i=0}^{N_p} \beta (P_d^*[i])^{\frac{1}{3}}$, where $i=1,\ldots,N_p$.*

Proof. Denote $\mathscr{Z}(P_d, N) = \sum_{i=0}^{N} \beta P_d^{\frac{1}{3}}[i]$. Assuming that $P_d^*[i]$ do not maximize $\mathscr{Z}(P_d^*, N_p)$, then we can always find $P_d'[i]$ making $\mathscr{Z}(P_d', N_p) - \mathscr{Z}(P_d^*, N_p) = \Delta M_p$, where $i=1,\ldots,N_p$ and $\Delta M_p > 0$. Because the dynamic energy increases monotonically with time, there is a N_x, $N_x < N_p$ that makes $\mathscr{Z}(P_d^*, N_p) = \mathscr{Z}(P_d^*, N_x) + \sum_{j=N_x}^{N_p} \beta P_d^{\frac{1}{3}}[j]$, where $\mathscr{Z}(P_d^*, N_x) = M_p$ and $\sum_{j=N_x}^{N_p} \beta P_d^{\frac{1}{3}}[j] = \Delta M_p$. As a result, $P_d^*[i]$ is not optimal because $N_x < N_p$. Therefore, the optimal solution of ***P4*** is $P_d^*[i]$ that maximizes $\sum_{i=0}^{N_p} \beta (P_d^*[i])^{\frac{1}{3}}$. □

Based on Lemma 6, the optimization problem **P4** can be converted into the optimization problem **P3**. The only difference between the the two problems is that the total number of time slots, N_p, in **P4** becomes a variable instead of a known value in **P3**. Although N_p cannot be written as a function of P_d that can be directly calculated, we can run the tightest string policy or the directional water-filling algorithm in each time slot to check if the current N_p and $P_d^*[i]$, $i = 0, \ldots, N_p$, make the constraint **C1** in (27) hold, as will be introduced next.

Here, we use the tightest string policy as an example to show how to manage the CPU clock frequency of the cloudlet so that the execution time of tasks offloaded from mobile IoT devices can be minimized:

Step 1: At the end of t_n, the cloudlet performs Step 1 to Step 5 introduced in Section V-C to find the optimal CPU frequency from t_0 to t_n.

Step 2: The cloudlet calculates the cumulative frequency. If $\sum_{i=0}^{n} f_c^*[i] < M_p$, and then more energy is needed to satisfy the constraint **C1** in (27). Therefore, the cloudlet waits for the new energy that will be arrived in the next time slot and let $n = n+1$, then go Step 3. If $\sum_{i=0}^{n} f_c^*[i] = M_p$, then go Step 4.

Step 3: Repeat Step 1. According to the new energy harvested in time slot $n+1$, the cloudlet calculates the new CPU frequencies between time slots t_0 and t_{n+1}.

Step 4: Let $n = N_p$, the current $f_c^*[i]$, $i = 0, \ldots, N_p$, are considered to be the optimal frequencies that can minimize the task execution time.

It is worth noting that in Step 3, the cloudlet does not need to recalculate all CPU frequencies obtained in previous time slots since most of the frequency values are reusable. Using Fig. 8 as an example, if the optimization process stops in t_7, then the

optimal strings are $\overline{p_0p_1}$, $\overline{p_1p_2}$, and $\overline{p_2p_3}$. If the optimization process stops in t_8, then the optimal strings are $\overline{p_0p_1}$, $\overline{p_1p_2}$, and $\overline{p_2p_4}$. Obviously, the first two strings, $\overline{p_0p_1}$ and $\overline{p_1p_2}$, remain unchanged.

The above observation is mainly caused by the energy causality constraint. It can also be interpreted by the energy dam, which has been discussed in Section VI. As a result, in most cases, when newly received energy joins the feasible energy tunnel, the cloudlet only needs to recalculate the last two strings. This greatly reduces the computational complexity of optimal CPU frequency management.

In addition to reducing the computational complexity, reusable frequencies are also very useful for developing online CPU scaling strategies. Specifically, as analyzed in Section VI, significant changes in energy intensity are likely to create energy dams, which divides a long-term CPU management into multiple short-term strategies. Therefore, if the cloudlet can predict variations of energy strength in the near future, it can efficiently manage the CPU frequency to achieve good computing performance.

VIII. Performance Evaluation

In this section, we will show the effectiveness of the proposed optimal CPU frequency scaling strategy and demonstrate the impacts of various system parameters through MATLAB simulations. We first compare the proposed offline CPU frequency scaling policy with two benchmark strategies in terms of the average CPU clock rate in order to evaluate their efficiency of energy utilization. The average CPU clock rate will indicate the number of computing tasks that can be executed given the limited energy supply from renewable energy. Next, the impacts of the dynamic solar intensity and varying battery capacity on the system performance are illustrated. We also study the performance of the online CPU frequency management strategy and evaluate the performance degradation compared to offline optimal policy in different conditions. The results presented in the figures are the average of 10 independent tests for each configuration.

A. Performance of Offline Strategies

For comparison, we introduce two benchmark offline strategies that work as follows:

- **Average scaling strategy:** In this strategy, the cloudlet first calculates the average of the energy harvesting rate across all time slots, and then adjusts the CPU frequency so that the dynamic power is equal to the average energy harvesting rate.
- **Greedy scaling strategy:** In this strategy, the cloudlet adjusts the CPU frequency to consume all the collected energy by the end of each time slot.

Based on the experimental results reported in [39] and [40], we choose a half-sine function with random fading and fluctuation to model the intensity of solar radiation in a day. Taking Fig. 9 as an example, from 6:00 AM to 8:00 PM in summer, the average intensity of solar energy is a half-sine function with small random fluctuations. When the sun is obscured by clouds, the intensity of solar power decreases

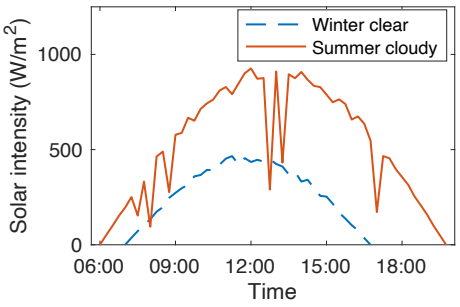


Figure 9: Changes of solar intensity in a day.

significantly, causing deep fadings on random places of the curve. In winter, the days are shorter (7:00 AM to 5:00 PM) and the average solar intensity is only half as strong as in summer. The peak intensity of solar energy in summer is 900 W/m^2 at 1:00 PM. In the simulation, the length of each time slot is $\Delta t = 10$ min. In each time slot, there is 10% probability that clouds block the sun and the solar intensity randomly drops to $1/5$ to $1/2$ of the original value. The frequency scaling coefficient is set to $\rho = 0.565 \times 10^9$. With this frequency scaling coefficient, when the dynamic power is 10 W and 150 W, the CPU clock frequencies are 1.2 GHz and 3.0 GHz, respectively.

In Fig. 10, we compare the proposed optimal strategy with two benchmark policies under different battery capacity, E_{max}. As depicted in Fig. 10, the performance of the average CPU frequency scaling strategy is much lower than that of the other two strategies, especially when the energy storage capacity is low. The average scaling strategy that uses fixed computing power is not able to adapt to the high dynamics in the harvested energy: when the solar intensity is high, the battery tends to overflow in the average scaling policy causing waste of energy; when the solar energy is weak, the average policy drains the battery quickly resulting in inefficient energy utilization. Therefore, we observe obvious performance improvement for the average strategy with a larger energy buffer. By contrast, the change of E_{max} has much less impact on the optimal strategy and the greedy strategy than the average policy. As a result, a relative stable clock rate is achieved with the optimal and the greedy strategies regardless of E_{max}.

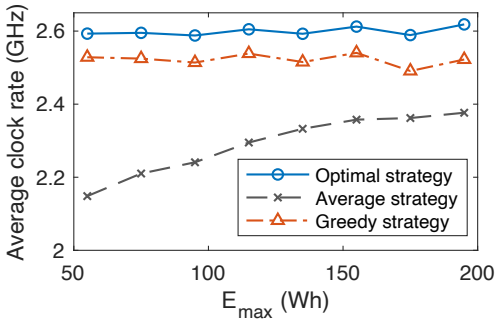


Figure 10: Impact of battery capacity on the performance of different CPU frequency management strategies in daylong tests.

The greedy strategy consumes all the harvested energy

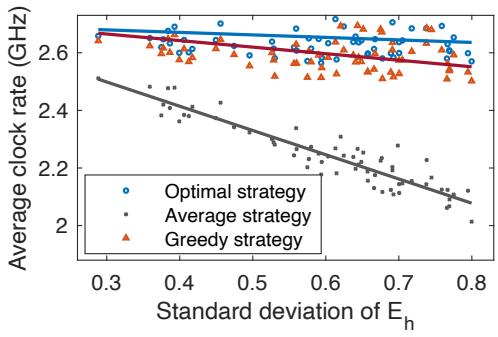


Figure 11: Impact of energy variation on the performance of different CPU frequency management strategies.

in each slot and can avoid waste of energy. However, the performance of the greedy policy is 5% lower than the optimal strategy due to inefficient energy utilization. Specifically, as shown in (2), the clock frequency of the CPU is a concave function of the dynamic power. Therefore, it is not efficient to improve the computing power of the cloudlet by increasing the dynamic power. In fact, if the maximum capacity of the energy storage is unlimited and all tasks have no deadline, the most efficient way to use the harvested energy is to perform the task at the lowest frequency, thereby maximizing the total clock cycles for a given energy consumption. However, the greedy strategy consumes all collected energy at the end of each time slot. The energy collected in past time slots cannot be saved for future use, resulting in high fluctuation in the computing capacity of cloudlets.

In addition to the daylong tests with time-varying average solar intensity, we also conducted short-term tests where the average energy intensity was constant during the experiment (i.e., $\bar{p}_h = 115\,\text{W}$), but the amount of energy harvested in each slot was dynamic representing the random fading caused by clouds. We vary the fluctuations of solar intensity and reveal the impact of energy variations on the performance of the three CPU frequency management strategies in Fig. 11, where the effectiveness of the proposed optimal strategy is again validated. In the figure, the battery capacity is set to $E_{max} = 115\,\text{Wh}$. Denote the relative standard deviation of E_h as σ_h, which indicates the variation of energy intensity between different time slots. In order to clearly show the relationship between the performance of the three strategies and σ_h, we perform the linear fit to all discrete points, and show the results with the solid line in the figure.

As shown in Fig. 11, when the fluctuation of energy intensity becomes large, the performance of all three CPU management strategies decreases linearly. According to Lemma 2, it can be realized that an ideal feasible energy tunnel should be the one that allows the CPU frequency to remain constant throughput the whole tunnel. In this case, the tightest string will be a single line segment connecting the start and end of the tunnel. To achieve this, the height of the steps in the feasible energy tunnel needs to be consistent, which requires the fluctuation of the energy intensity to be as small as possible. Otherwise, the slope of the tightest string will keep changing to meet the constraints of the energy causality and energy storage capacity, thereby reducing the efficiency of energy utilization.

From Fig. 11, it can be observed that the average frequency management strategy is more sensitive to energy variation than the greedy strategy and the optimal policy. For instance, when the relative standard deviation of E_h increase from 0.3 to 0.8, the clock rate with the average frequency management strategy is reduced from 2.52 GHz to 2.08 GHz, that is, the computing power decreases by 18%. In the same situation, the average clock rate of the greedy strategy is reduced by 8%. This is because, with the increase of σ_h, the harvested energy is very likely to be underutilized. As a result, there is a high probability that the energy storage is fully charged in some time slots. This greatly reduces the energy harvesting efficiency because the arriving energy cannot be successfully stored by the cloudlet for future use.

B. Performance of Online Strategy

In the online CPU frequency scaling strategy, we assume the cloudlet is able to accurately predict the energy harvesting rates in the next N_f time slots. N_f is considered as the prediction ability of cloudlets. In the past few decades, how to predict the intensity of solar and wind energy has been extensively studied [41]–[43], which is out of the scope of this paper.

Compared to the offline policy, the whole feasible energy tunnel is equally divided into multiple sub-tunnels of N_f length in the online solution. The DP method is performed independently in each N_f-tunnel to calculate the dynamic power and the corresponding clock rate. The shortened energy tunnel and the prediction error lead to degraded performance of the online solution. We normalize the performance of the online method by dividing its clock frequency by the clock rate of the optimal offline strategy.

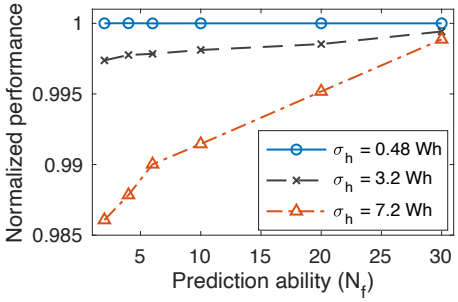
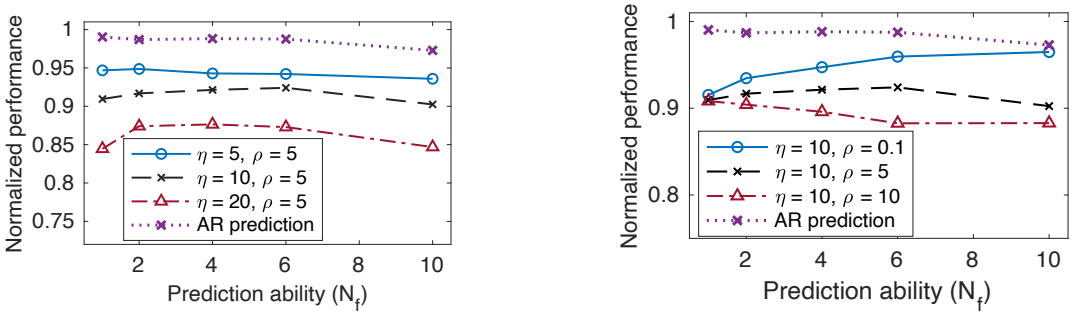


Figure 12: Performance of online CPU frequency scaling strategy with accurate prediction on energy harvesting rate.

For the online CPU frequency scaling, we study two different situations: (a) The cloudlet can accurately predict the future energy harvesting rate, and (b) the cloudlet can predict the energy harvesting rate with some errors. We start from the first case, where the cloudlet can perfectly predict the energy harvesting rates in the next N_f time slots.

Fig. 12 shows the performance of the online strategy with respect to N_f. As illustrated in the figure, the performance of the online strategy is proportional to N_f. With the increment of prediction capability, the performance of the online strategy will approach to the optimal solution quickly. Moreover, the



(a) Impact of initial value.

(b) Impact of slope.

Figure 13: Performance of online CPU frequency scaling strategy with prediction errors on energy harvesting rate.

variation of energy intensity has negative impact on the performance of the online strategy. The reason is the same as the low performance of the offline average frequency management strategy that is analyzed in Fig. 11. From Fig. 12, it can be observed that even if the cloudlet can only accurately predict the energy harvesting rate for the next several time slots, the performance of the online strategy could be very close to the optimal offline policy. When N_f is reduced to 1, the online CPU frequency scaling strategy degenerates to the greedy policy.

In practice, prediction errors exists in the estimation of future energy harvesting rates, which will further reduce the performance of online CPU scaling strategy. We use a simple autoregressive (AR) [44] model to predict the incoming energy intensity and conduct numerical evaluations to evaluate the performance of the prediction-based online transmission strategy in Fig. 13. The prediction error is generally small while predicting the harvest rate in the next adjacent slot and grows with the slot increases. Therefore, we can observe a slight performance degradation with the growth of N_f for the AR prediction-based online strategy.

In order to further evaluate the impact of prediction error on the performance of prediction-based online CPU scaling policy, we manually add errors to the estimation of future energy harvesting rates. Here, we consider a linear error model where the prediction error, $e=\eta+\rho(n-1)$, with $n=1,\cdots,N_f$. We change η and ρ to generate varying prediction error and then plot the normalized performance in Fig. 13.

In Fig. 13(a), we fix the slope of the prediction error, ρ, but change the initial prediction error, η, from 5 Wh to 20 Wh. As shown in the figure, increasing the initial prediction error significantly reduces the performance of online strategy. The normalized performance is 0.997 when there is no prediction error, $N_f=1$, and $\sigma_h=3.2$ Wh (i.e. black curve in Fig. 12). It reduces to 0.947, 0.910, and 0.845 when η is 5 Wh, 10 Wh, and 15 Wh, respectively. From the figure it can be observed that if the initial prediction error is small (e.g., η is 5 Wh or 10 Wh), the performance of the online CPU scaling strategy can achieve a very good performance even if N_f is small.

Next, we investigate the impact of ρ on the performance of online strategy in Fig. 13(b). When $\rho=0.1$ Wh, the growth of the prediction error with time slots is negligible; the estimation of energy harvesting rate in all time slots has comparable prediction error. In this case, the benefit from larger prediction ability (i.e., larger N_f) leads to better normalized performance. When $\rho=5$ Wh, the prediction errors in estimating the harvest rate in the end slots become significant with large N_f. In this case, the negative impact of growing prediction error surpasses the benefit from better prediction ability. Therefore, a moderate prediction ability is suggested, e.g., $N_f=6$. When ρ becomes very large (e.g., $\rho=10$ Wh), the greedy strategy (i.e., $N_f=1$) is most favorable.

Combining Fig. 12 with Fig. 13, we realize that the variation of the energy intensity (σ) and the initial prediction error (η) have the most significant impact on the performance of the online CPU scaling strategy. To reduce σ and η, the most efficient way is to decrease the length of each time slot (Δt). This is because the change of energy intensity is usually a continuous process, reducing Δt will make the adjacent measures of the energy intensity highly correlated. In this case, using the greed strategy can achieve similar a performance as the offline optimal CPU frequency scaling policy.

IX. Conclusions and Future Work

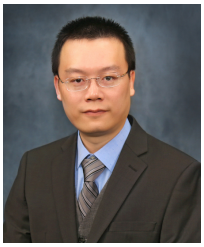
In this paper, we studied the CPU frequency scaling problem of the cloudlet with energy harvesting capability. Two optimization problems subject to the constraints of energy causality and energy storage capacity are developed to (a) maximize the computing power of the CPU within a certain period of time, or (b) minimize the execution time of tasks offloaded from mobile IoT devices. As analyzed in the paper, the second optimization problem can be transformed into the first one. To give insight into efficient CPU frequency management, the optimization problem is solved through the graphical method (tightest string) and the algorithmic method (directional water-filling). In addition, we also introduced a suboptimal DP-based solution by converting the optimal frequency scaling problem into a shortest path problem. The DP method is a practical and general solution that can be easily applied to the optimal CPU frequency scaling problems. Finally, how to design an online strategy for frequency management is discussed. Simulation results verified the efficiency of the proposed CPU frequency scaling policies.

The future work will focus on two directions. 1) We aim to implement the optimal CPU frequency scaling policy on a real sustainable cloudlet and conduct field experiments. More

practical issues, such as the piecewise nonlinear lithium-ion battery charging and the power consumption model considering system power and CPU leakage power, will be considered when developing the optimal CPU frequency scaling policies. 2) We will integrate the proposed sustainable cloudlet CPU frequency scaling and task allocation in order to develop a more comprehensive sustainable edge computing system. The stochastic task arrival, dynamic wireless channel, CPU frequency scaling, and dynamics in renewable energy will be taken into account in the policy design.

ACKNOWLEDGEMENT

This work is supported in part by the US National Science Foundation under Awards CIF-2051356, CNS-2122167, CNS-2122159, CNS-2006453, CNS-2126151, and ECCS-2210106.



Dr. Yu Luo received the B.S. degree and the M.S. degree in electrical engineering from the Northwestern Polytechnical University, China, in 2009 and 2012, respectively. In 2015, he received the Ph.D. degree in computer science and engineering from University of Connecticut, Storrs. Dr. Luo is currently an Assistant Professor at Mississippi State University. His major research focus on the sustainable wireless networks for emerging IoT, RF energy harvesting hardware, security in RF energy harvesting wireless networks, and underwater wireless networks. He is a Co-recipient of the Best Paper Award in IFIP Networking 2013 and Chinacom 2016.

Dr. Lina Pu received the B.S. degree in electrical engineering from the Northwestern Polytechnical University, Xi'an, China in 2009 and the Ph.D. degree in Computer Science and Engineering from University of Connecticut, Storrs. Dr. Pu is currently an Assistant Professor at University of Alabama. Her research interests lie in the area of edge computing, RF energy harvesting wireless networks, security in the sustainable IoT, and underwater acoustic/VL networks. She owned IFIP Networking 2013 best paper award.

Dr. Chun-Hung Liu (Senior Member, IEEE) received the B.S. degree in Mechanical Engineering from National Taiwan University, Taipei, Taiwan, the M.S. degree in Mechanical Engineering from Massachusetts Institute of Technology, Cambridge MA, USA, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, TX, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering at Mississippi State University, Starkville MS, USA. He was with the University of Michigan, Ann Arbor MI, USA, and National (Yang Ming) Chiao Tung University, Hsinchu, Taiwan. His research interests lie in the areas of wireless networking, machine learning, high-dimensional data analysis, stochastic control, and optimization theory. He was a recipient of the Best Paper Award from the IEEE Global Communications Conference (Globecom) in 2008 and 2014, a recipient of the Excellent Young Researcher Award from the Ministry of Science and Technology of Taiwan in 2015, a recipient of the Outstanding Advisor Award from the Taiwan Institute of Electrical and Electronic Engineering in 2016.

REFERENCES

[1] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[2] J. Wang, W. Wu, Z. Liao, R. S. Sherratt, G.-J. Kim, O. Alfarraj, A. Alzubi, and A. Tolba, "A probability preferred priori offloading mechanism in mobile edge computing," *IEEE Access*, vol. 8, pp. 39 758–39 767, 2020.

[3] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4132–4150, 2019.

[4] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: collaborative intelligence between the cloud and mobile edge," *Proceedings of ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[6] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6774–6785, 2019.

[7] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[8] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "TOFFEE: task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Transactions on Cloud Computing*, 2019.

[9] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[10] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[11] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[12] L. P. Yu Luo and C.-H. Liu, "Optimal CPU Frequency Scaling Policies for Sustainable Edge Computing," in *proceedings of IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) (Under review)*. IEEE, 2021, pp. 1–6.

[13] J. Cochran, M. Miller, O. Zinaman, M. Milligan, D. Arent, B. Palmintier, M. O'Malley, S. Mueller, E. Lannoye, A. Tuohy *et al.*, "Flexibility in 21st century power systems," National Renewable Energy Lab.(NREL), Golden, CO (United States), Tech. Rep., 2014.

[14] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: modeling and experimental validation on mobile devices," in *proceedings of International Conference on Parallel Processing and Applied Mathematics*. Springer, 2013, pp. 793–803.

[15] R. F. da Silva, H. Casanova, A.-C. Orgerie, R. Tanaka, E. Deelman, and F. Suter, "Characterizing, modeling, and accurately simulating power and energy consumption of i/o-intensive scientific workflows," *Journal of computational science*, vol. 44, p. 101157, 2020.

[16] M. Travers, "CPU power consumption experiments and results analysis of intel i7-4820k," *Newcastle University, Newcastle*, 2015.

[17] K. D. Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2013, pp. 793–803.

[18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[19] F. Zhou and R. Q. Hu, "Computation efficiency maximization in wireless-powered mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3170–3184, 2020.

[20] A. Gougeon, B. Camus, and A.-C. Orgerie, "Optimizing green energy consumption of fog computing architectures," in *2020 IEEE 32nd Inter-*

national Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, 2020, pp. 75–82.

[21] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[22] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 301–313, 2016.

[23] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2014, pp. 352–357.

[24] H. Mamur, "Design, application, and power performance analyses of a micro wind turbine," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 23, no. 6, pp. 1619–1637, 2015.

[25] Intel Company, "Intel Xeon Gold 6328HL Processor," intel.com, 2021, [Accessed: Feb, 2021]. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/gold-processors/gold-6328hl.html

[26] AMD Company, "AMD EPYC 7501," amd.com, 2021, [Accessed: Feb, 2021]. [Online]. Available: https://www.amd.com/en/products/cpu/amd-epyc-7501

[27] G. Papadakis, P. Tsamis, and S. Kyritsis, "An experimental investigation of the effect of shading with plants for solar control of buildings," *Energy and buildings*, vol. 33, no. 8, pp. 831–836, 2001.

[28] Ossila Company, "Perovskites and perovskite solar cells: an introduction," ossila.com, 2018, [Accessed: Feb, 2021]. [Online]. Available: https://www.ossila.com/pages/perovskites-and-perovskite-solar-cells-an-introduction

[29] SpaceX Company, "Starlink," starlink.com, 2021, [Accessed: Feb, 2021]. [Online]. Available: https://www.starlink.com

[30] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 13, no. 2, pp. 203–221, 1996.

[31] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *proceedings of the 41st Annual Design Automation Conference*, 2004, pp. 868–873.

[32] M. A. Zafer, "Dynamic rate-control and scheduling algorithms for quality-of-service in wireless networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[33] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[34] M. Sniedovich, *Dynamic programming: foundations and principles*. CRC press, 2010.

[35] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[36] C. Xing, Y. Jing, S. Wang, S. Ma, and H. V. Poor, "New viewpoint and algorithms for water-filling solutions in wireless communications," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1618–1634, 2020.

[37] P. He, L. Zhao, S. Zhou, and Z. Niu, "Water-filling: a geometric approach and its application to solve generalized radio resource allocation problems," *IEEE transactions on Wireless Communications*, vol. 12, no. 7, pp. 3637–3647, 2013.

[38] O. Ozel, K. Tutuncuoglu, J. Yang, S. Ulukus, and A. Yener, "Transmission with energy harvesting nodes in fading wireless channels: optimal policies," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 8, pp. 1732–1743, 2011.

[39] F. Wang, Y. Zhu, and J. Yan, "Performance of solar PV micro-grid systems: a comparison study," *Energy Procedia*, vol. 145, pp. 570–575, 2018.

[40] A. Avila, P. R. Vizcaya, and R. Diez, "Daily irradiance test signal for photovoltaic systems by selection from long-term data," *Renewable Energy*, vol. 131, pp. 755–762, 2019.

[41] A. Cammarano, C. Petrioli, and D. Spenza, "Pro-energy: a novel energy prediction model for solar and wind energy-harvesting wireless sensor networks," in *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*. IEEE, 2012, pp. 75–83.

[42] T. Khatib, A. Mohamed, and K. Sopian, "A review of solar energy modeling techniques," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 5, pp. 2864–2869, 2012.

[43] F. Cassola and M. Burlando, "Wind speed and wind energy forecast through kalman filtering of numerical weather prediction model output," *Applied energy*, vol. 99, pp. 154–166, 2012.

[44] K. E. Baddour and N. C. Beaulieu, "Autoregressive modeling for fading channel simulation," *IEEE Transactions on Wireless Communications*, vol. 4, no. 4, pp. 1650–1662, 2005.