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 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 of the central processing unit (CPU) to balance energy consumption and computing power. In this paper, we consider the limited energy storage of the cloudlet and the dynamic changes in sustainable energy strength, and then develop two offline optimal CPU frequency scaling policies to (a) maximize the overall computing power of the cloudlet within a certain period of time, and (b) minimize the execution time of tasks offloaded from mobile devices to the cloudlet. The tightest string policy and the directional water-filling strategy are adopted to solve the optimization problem from the graphical perspective and the algorithmic viewpoint, respectively. In addition to theoretical analysis, dynamic programming (DP) based computing method is introduced to verify the optimality of the developed frequency scaling policy. 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)

1 INTRODUCTION

In recent years, mobile edge computing (MEC) is emerging as a new computing paradigm for the mobile Internet of things (IoT) [1], [2]. 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. In this case, how to minimize the energy consumption of the cloudlet while meeting the deadline of each task is the main objective. To achieve this goal, strategies that offload tasks from mobile devices to the cloudlet and manage the computing power of the cloudlet have been comprehensively studied. [3]–[6].

Our work is different from conventional MEC in that it considers sustainable cloudlets, which can harvest energy from the surrounding environment. 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, sustainable edge computing (SEC) can greatly improve the scalability and sustainability of existing mobile IoT networks, because cloudlets can be deployed outside the coverage of the electric grid, (e.g., in the wild or on the mountains or even oceans) to provide resource-limited IoT devices with considerable computing power to perform complex tasks.

In SEC, managing the computing power of cloudlets is a new challenge. This is because the power density of wind and solar energy is not constant and varies with time [7]. Therefore, the energy harvested from the environment may not always allow cloudlets to run at full speed. We should carefully study how to manage power consumption of the cloudlet in a dynamic energy environment to maximize computing performance.

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 [8]. As a result, from an energy perspective, increasing the clock frequency to improve computing power is not an efficient strategy. Therefore, if the cloudlet has unlimited energy storage and the task has no deadline, the CPU should always run at the lowest clock frequency so as to save the energy.

After considering the limitations of task deadlines and energy storage capacity, it becomes difficult to accurately adjust CPU frequency. Specifically, if the cloudlet keeps running at minimum power, the energy storage may overflow, causing failed energy collection or missing the mission deadlines: the former reduces the efficiency of energy harvesting, while the latter introduces unexpected latency on the IoT network; both have a negative impact on network performance. Therefore, the cloudlet needs to manage its computing power to adapt to the changes in energy strength.

In this paper, we consider the scenario where the energy that can be collected in the future is known, or at least can be predicted at a certain level. In this case, two optimal offline CPU frequency scaling policies are developed 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 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, based on the results observed from the directional water-filling strategy, how to develop an online CPU frequency scaling strategy is briefly discussed. Moreover, a dynamic programming (DP) based computing method is introduced to convert the optimal CPU frequency management into a routing problem with weighted rewards on different paths. The results obtained from DP routing can verify the optimality of the frequency scaling policy we designed for the SEC cloudlet.

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 contributions of our work are four-fold:

a) We develop a system model to describe the CPU frequency scaling problem in the SEC cloudlet.

b) Based on the developed model, we convert the CPU frequency scaling problem into two optimization problems with different objectives and constraints.

c) We solve the optimization problem from a graphical perspective and an algorithmic point of view to give insight into the CPU frequency management. The DP-based computing method is applied to verify the optimality of the developed policy.

d) The 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 2 introduces the related work. The system model of the CUP frequency management for the SEC cloudlet is given in Section 3. In Section 4, we introduce the optimization problem to maximize the computing power of the cloudlet in a certain period of time; its solutions are provided in Sections 5 and 6 from the graphical perspective and the algorithmic viewpoint, respectively. How to minimize the time it takes for the cloudlet to execute a certain number of tasks is studied in Section 7. We conclude our work in Section 9.

2 Related Work

CPU frequency scaling and task offloading play a crucial role in MEC to minimize the energy consumption of cloudlets without missing task deadlines of resource-limited devices. How to properly allocate energy and time between computation and wireless communication is a hot topic in academia and industry [3], [5], [9]–[11].

In practice, IoT devices can offload all or partial computation task to the cloudlet [2]; the former and the latter are called the binary offload scheme and the partial offload scheme, respectively. The task offloading and CPU scaling strategy needs to comprehensively consider the deadline of each task, the quality of the wireless channel, the topology of the tasks (sequential, parallel, or general dependency [12], [13]), and the power consumption of communications and computations.

In [9], joint task allocation and CPU frequency scaling is modeled as a stochastic optimization problem, which takes into account the dynamics of task arrival and variation of wireless channel states. The work aims at minimizing the energy consumption subject to the queue delay constraint of the cloudlet. In [11], it assumes that mobile devices can harvest energy from environments. How to manage the transmission power of the mobile device and the CPU frequency of the cloudlet is studied. The goal of that work is to minimize the execution delay and task dropping cost of the device. In [14], the authors consider an application that combines MEC with the wireless power transfer (WPT) technology, where a power station can transmit energy packets to charge mobile devices. In the work, both the binary offload scheme and the partial offload scheme are studied to maximize the computation efficiency (the ratio of the total computed bits to the consumed energy) of the cloudlet. This is achieved by jointly optimizing the cloudlet’s CPU frequency, energy harvesting time, offloading time, and transmission power of mobile devices.

Instead of offloading tasks to a single cloudlet, in [5], it is assumed that multiple cloudlets are deployed near mobile devices. The strategy of offloading tasks from a single device to plural cloudlets is studied to minimize the computing latency and the total energy consumption of cloudlets. In [15], the authors introduced an edge computing based mobile augmented reality (AR) system. The purpose of the system is to minimize the energy consumption of all cloudlets in processing a frame of images while satisfying user preferences. To achieve that, the cloudlet needs to dynamically change configurations, such as the CPU frequency and the size of the computation model, based on user preferences, camera sampling rate, and available radio resources on the edge server.

Recently, machine learning (ML) technology is applied to MEC to make resource allocation and task offloading more intelligent. In [3], the authors define an energy-time cost, which is the weighted sum of the total energy consumption of a mobile device and execution time of tasks on a cloudlet. The work aims at adjusting the CPU frequency and offloading strategy in a time-varying wireless fading channel to minimize the energy-time cost. To find the best strategy, a deep reinforcement learning (DRL) based actor-critic learn-
3 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.

3.1 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 m²), 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 [18]. These energy harvesting rates are sufficient to drive high-performance server processors, such as Intel Xeon Gold 6328HL [19] or AMD EPYC 7501 [20], 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/m² [21]. The power conversion efficiency of commercial solar panels is between 15% and 20% [22]. 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 [23]) 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.

3.2 Mathematical Model

Denote the clock frequency of the cloudlet CPU at time \( t \) by \( f_c(t) \). According to processor design [24], 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 circuit theory [8], the dynamic power at time \( t \), which is denoted by \( P_d(t) \), can be calculated as

\[
P_d(t) = \alpha f_c(t)V_c(t)^2,
\]

where \( \alpha \) 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 \( V_c \) [24]. By adopting dynamic voltage scaling (DVS) technology, modern processors can dynamically scale down the voltage based on the frequency requirement [25]. Consequently, the CPU clock frequency can be written as a function of the dynamic power:

\[
f_c(t) = \beta P_d^{1/2}(t), \quad P_d \geq 0,
\]

where \( \beta \) 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, the cloudlet cannot use the energy that has not yet arrived. As a result, the total energy consumed by cloudlet cannot exceed the cumulative harvested energy, i.e.,

\[
\int_0^t P_d(u) \, du \leq \int_0^t p_h(u) \, du, \quad \forall t \geq 0.
\]

Assume the maximum capacity of the energy storage used by the cloudlet is \( E_{max} \). 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 \( E_{\text{max}} \), i.e.,

\[
\int_0^t p_h(u) \, du - \int_0^t P_d(u) \, du \leq E_{\text{max}}, \quad \forall t \geq 0. \tag{4}
\]

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

**Optimal computing power:** It optimizes the overall computing power, so the total number of machine codes that can be executed by the cloudlet in a certain time period, \([0, T_p]\), is maximized:

\begin{align*}
\textbf{P1:} \quad & \arg \max_{P_d(t) \geq 0} \int_0^{T_p} \beta P^3_d(t) \, dt, \quad T_p > 0, \\
\text{s.t.} \quad & C1: \quad \int_0^t P_d(u) \, du \leq \int_0^t p_h(u) \, du, \quad \forall t \in [0, T_p], \\
& C2: \quad \int_0^t p_h(u) \, du - \int_0^t P_d(u) \, du \leq E_{\text{max}}, \quad \forall t \in [0, T_p].
\end{align*}

(5)

**Optimal execution time:** It minimizes the time it takes for the cloudlet to execute a total number of \( M_p \) machine codes:

\begin{align*}
\textbf{P2:} \quad & \arg \min_{P_d(t) \geq 0} T_p, \quad T_p > 0, \\
\text{s.t.} \quad & C1: \quad \int_0^{T_p} \beta P^3_d(u) \, du = M_p, \\
& C2: \quad \int_0^t P_d(u) \, du \leq \int_0^t p_h(u) \, du, \quad \forall t \in [0, T_p], \\
& C3: \quad \int_0^t p_h(u) \, du - \int_0^t P_d(u) \, du \leq E_{\text{max}}, \quad \forall t \in [0, T_p].
\end{align*}

(6)

The objective function of \( \text{P1} \) enables the cloudlet to achieve the best computing performance in a certain period of time. Therefore, the CPU frequency scaling policy obtained from \( \text{P1} \) is suitable for IoT networks that run computation-intensive applications. In contrast, the objective function of \( \text{P2} \) allows the cloudlet to complete a certain number of tasks in the shortest time. As will be introduced in Section 7, the optimization problem \( \text{P2} \) can be converted into \( \text{P1} \). The solution of \( \text{P2} \) can be used to estimate the minimum computing latency that the cloudlet can achieve to perform specific tasks offloaded from mobile IoT devices.

4 Processing Power Optimization

Based on the optimization problem \( \text{P1} \) given in Section 3.2, 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 \( \text{P1} \), we first discretize \( p_h \) in (5). Let \( \Delta t \) represent a short time period, which is an aliquot part of \( T_p \); i.e., \( T_p \) can be divided by \( \Delta t \). 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 \( \Delta t \) is small, it is reasonable to assume that the incident power of sustainable energy remains constant within a time slot, then we have that

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

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

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

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 [26].

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

\[
f^*_c(t) = f^*_c[n], \quad t \in [n\Delta t, (n+1)\Delta t), \quad n = 0, 1, 2, \ldots \tag{9}
\]

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

\[
f_c[n] = \beta P^3_d[n], \quad n = 0, \ldots, N_p, \tag{10}
\]

where \( N_p = T_p/\Delta t - 1 \).

Through the above discretization process, the continuous optimization problem \( \text{P1} \) can be converted into a piece-wise optimization problem:

\begin{align*}
\textbf{P3:} \quad & \arg \max_{P_d[i] \geq 0} \sum_{i=0}^{N_p} \beta P^3_d[i] \Delta t, \quad N_p = 0, 1, 2, \ldots, \\
\text{s.t.} \quad & C1: \quad \sum_{i=0}^{n} P_d[i] \Delta t \leq \sum_{i=0}^{n} E_h[i], \quad n = 0, \ldots, N_p, \\
& C2: \quad \sum_{i=0}^{n} E_h[i] - \sum_{i=0}^{n} P_d[i] \Delta t \leq E_{\text{max}}, \quad n = 1, \ldots, N_p.
\end{align*}

(11)

In the optimization problem \( \text{P3} \), the objective function is concave because it is a linear combination of concave functions. In addition, \( \text{C1} \) and \( \text{C2} \) in (11) are composed of linear constraints, so they are convex. Consequently, there exist KKT multiplier sets \( \lambda = \{\lambda_0, \ldots, \lambda_{N_p}\} \) and \( \lambda = \{\lambda_1, \ldots, \lambda_{N_p+1}\} \) to make the following conditions hold:
Stationarity:

\[ \nabla P_{d}^{*}[i] \mathcal{L} = \nabla P_{d}^{*}[i] \left( \sum_{i=0}^{N_{p}} \beta P_{d}^{\frac{1}{2}}[i] \Delta t \right) \]

\[ - \sum_{n=0}^{N_{p}} \mu_{n} \nabla P_{d}^{*}[i] \left( \sum_{i=0}^{n} P_{d}[i] \Delta t - \sum_{i=0}^{n} E_{h}[i] \right) \]

\[ - \sum_{n=1}^{N_{p}} \lambda_{n} \nabla P_{d}^{*}[i] \left( \sum_{i=0}^{n} E_{h}[i] - \sum_{i=0}^{n} P_{d}[i] \Delta t - E_{\max} \right) = 0, \]

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

Complementary slackness:

\[
\begin{cases} 
\mu_{n} \left( \sum_{i=0}^{n} P_{d}[i] \Delta t - \sum_{i=0}^{n} E_{h}[i] \right) = 0, & n = 0, \ldots, N_{p} \\
\lambda_{n} \left( \sum_{i=0}^{n} E_{h}[i] - \sum_{i=0}^{n} P_{d}[i] \Delta t - E_{\max} \right) = 0, & n = 1, \ldots, N_{p}.
\end{cases}
\]

Dual feasibility:

\[
\begin{cases} 
\mu_{n} \geq 0, & n = 0, \ldots, N_{p}, \\
\lambda_{n} \geq 0, & n = 1, \ldots, N_{p}.
\end{cases}
\]

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.

5 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. To achieve this, we first construct an energy feasibility tunnel based on the constraints of the energy causality and the energy storage capacity. Afterward, based on Lemmas and Corollary, the solution of \( P3 \) is given. Then, the optimal policy, called the tightest string policy, is introduced to efficiently manage the CPU frequency. Finally, we introduce a DP-based method to verify the optimality of the tightest string policy.

5.1 Energy Feasibility Tunnel

In Fig.2, X-axis is the time and Y-axis is the dynamic energy consumed by the cloudlet, i.e., \( \sum_{i=0}^{n} P_{d}[i] \Delta t \). According to the constraints of \( P3 \), an energy feasibility tunnel can be formed to restrict the dynamic energy, as shown in the figure. At any time, the dynamic energy cannot fall outside the tunnel: If it exceeds the upper bound of the tunnel, the energy causality constraint will be violated; if it is below the lower bound of the tunnel, the constraint of the energy storage capacity will not hold.

The dynamic energy curve is admissible if it is in the energy feasibility tunnel. In the optimization problem \( P3 \), we aim at finding an admissible curve to maximize \( \mathcal{F}(P_{d}) \).

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 energy feasibility 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 energy feasibility tunnel at time \( T_{m} \), where \( T_{m} \in (m \Delta t, (m + 1) \Delta t) \), the curve must be below the lower bound of the energy feasibility tunnel between \( (m \Delta t, T_{m}) \), 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 energy feasibility tunnel at time \( T_{a} \), then the curve must exceed the upper bound of the energy feasibility tunnel between \( (T_{a}, (m + 1) \Delta t) \), which violates the energy causality constraint, as shown in curve (d) of Fig.3.

According to Corollary 2 and the constrains of energy causality and energy storage capacity, at the end of \( T_{m} \), the optimal dynamic energy curve has only three potential states: (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. The states (d) and (e) will not occur.

![Figure 3: Three possible states of the dynamic energy curve and the end of time slot m.](image-url)
5.2 Solution of Precessing Power Optimization

By solving the KKT stationarity condition in (12), the following result can be obtained:

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

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

**Lemma 2.** Under the optimal policy, the CPU clock frequency remains unchanged when the energy storage becomes full, i.e., \(E_h[m] - P_d[i] \Delta t < E_{max}, \forall m \in [0, N_p - 1]: f^*_m[m + 1] \geq f^*_m[m + 1]. \)

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

\[
\sum_{i=0}^{m} P_d[i] \Delta t - \sum_{i=0}^{m} E_h[i] \neq 0,
\]

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

\[
P_d^*[m] = \left(\frac{3}{\beta} \left(\sum_{n=m+1}^{N_p} \mu_n - \sum_{n=m+1}^{N_p} \lambda_n\right)\right)^{-\frac{3}{2}}
\]

\[
= P_d^*[m + 1].
\]

From (20) and the relation between the CPU clock frequency and the dynamic power given in (10), we have that \(f^*_m[m + 1] \geq f^*_m[m + 1]. \)

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

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

\[
\sum_{i=0}^{m} P_d[i] \Delta t - \sum_{i=0}^{m} E_h[i] = 0,
\]

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 \(\mu_m \geq 0\). Finally, according to (17), it can be obtained that

\[
P_d^*[m] = \left[\frac{3}{\beta} \left(\sum_{n=m+1}^{N_p} \mu_n - \sum_{n=m+1}^{N_p} \lambda_n\right)\right]^{-\frac{3}{2}}
\]

\[
= P_d^*[m + 1].
\]

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 that \(f^*_m[m + 1] \geq f^*_m[m]\).

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

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

\[
\sum_{i=0}^{m} P_d[i] \Delta t - \sum_{i=0}^{m} E_h[i] \neq 0,
\]

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 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

\[
P_d^*[m] = \left[\frac{3}{\beta} \left(\sum_{n=m+1}^{N_p} \mu_n - \sum_{n=m+1}^{N_p} \lambda_n\right)\right]^{-\frac{3}{2}}
\]

\[
= P_d^*[m + 1].
\]

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 that \(f^*_m[m + 1] \leq f^*_m[m]\).
Lemma 5. Under the optimal policy, the cloudlet consumes all the harvested energy by the end of the last slot (i.e., \(\sum_{i=0}^{N_p} P_a[i] \Delta t = \sum_{i=0}^{N_p} E_h[i]\)).

Proof. If energy is not exhausted in the last time slot with the optimal \(P_a[i], i = 1, \ldots, N_p\), we can always find \(P_a[N_p] > P_a^*\) that consumes all the collected energy. Because \(\mathcal{F}\) in (16) is a monotonically increasing function of \(P_a\), we thus have \(\mathcal{F}(P_a^* N_p)) > \mathcal{F}(P_a[N_p])\), which means that \(P_a[N_p]\) is not optimal. Therefore, the optimal policy must consume all harvested energy in the last time slot.

5.3 Tightest String Policy

Here, we use an example to introduce how to utilize the Lemmas and Corollaries proved in Sections 4 and 5.2 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.

**Step 1:** Connect \(p_0\) with all turning points in the energy feasibility tunnel. Remove the strings that have any portion fall outside the tunnel. The rest of the strings, which are \(p_0p_1, p_0p_2,\) and \(p_0p_3,\) 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 \(p_0p_2\) and \(p_0p_3\) in Fig. 4, and then go to the next step to examine each retained string.

**Step 3:** We check \(p_0p_2\) first. Let \(p_2\) be the new starting point, and then repeat Step 1 and Step 2 to obtain all admissible strings, \(p_0p_2\) and \(p_0p_3\). Because \(p_0p_2\) hits the lower bound of the energy feasibility tunnel, the energy storage is full accordingly. 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 \(p_0p_2\) must be smaller than that of \(p_0p_3\). However, it can be observed from Fig. 4 that the slopes of \(p_0p_2\) and \(p_0p_3\) are both greater than the slope of \(p_0p_3\). Therefore, \(p_0p_2\) needs to be removed from the admissible strings, and only \(p_0p_3\) 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 \(p_0p_3\) and \(p_0p_7\). We first check \(p_0p_3\). 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 \(p_7\). As shown in the figure, \(p_0p_3\) reaches the upper bound of the energy feasibility 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 \(p_0p_7\) should be greater than that of the slope of \(p_0p_3\), which is unsatisfactory. Therefore, \(p_0p_3\) is removed from the admissible strings. Finally, \(p_0p_3\) and \(p_0p_7\) 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 energy feasibility 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 [26].

5.4 Dynamic Programming Method

In Section 5.3, we introduced how to get the tightest string in the energy feasibility tunnel. Next, we study how to obtain the optimal CPU frequency scaling strategy by the DP algorithm, a computer programming method, without using any Lemmas and Corollaries given in Sections 4 and 5.2. The results obtained from the DP algorithm can verify that the tightest string policy is indeed the optimal solution for managing the CPU frequency of the SCE cloudlet.

The DP algorithm was developed by Richard Bellman in the 1950s [27]. It can decompose a complex optimization problem into multiple sub-problems to find the optimal solution in a recursive manner. An important application of DP is to find the optimal path between source and destination in the network routing design [28].
Let $V_{i,j}$ be the vertex in the row $i$ and column $j$. A segment that connects two vertexes in adjacent columns is called the edge. Let $E_{i,j}^{t} \Delta t, j+1$ be the edge connecting $V_{i,j}$ and $V_{i+\Delta t, j+1}$, where $\Delta t$ is zero or a positive integer.

In order to find the optimal CPU frequency scaling strategy, suppose that a virtual point is placed at the starting point $V_{0,0}$. The point moves along the edge to the end point, which is a vertex located at the end of the energy feasibility tunnel. In the moving process, the cloudlet consumes dynamic power and gets CPU frequency as a reward. Specifically, according to the definition of the energy feasibility tunnel, the slope of the edge is equal to the dynamic power, which determines the reward by (2).

Essentially, the route in the energy feasibility tunnel is the dynamic energy curve, which increases monotonically with time. Therefore, the virtual point is not allowed to move backward or downward. In addition, the point cannot move vertically; otherwise, the dynamic power will become infinite, which is impractical in the real world. For example, in Fig. 5, assuming that the current position of the virtual point is at $V_{10,4}$, then the next hop of this point can only be selected from $V_{11,4}$ to $V_{11,8}$.

Following the above description, maximizing the computing power 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 rewards. To achieve this goal, DP-based methods like the Dijkstra’s algorithm can be applied to find the optimal solution. The computational complexity of the Dijkstra’s algorithm is $O(n \log n)$, where $n$ is the number of grids in the energy feasibility tunnel. More details about the DP algorithm can be found in [29].

By increasing grids density in the tunnel, the results obtained from the DP method can gradually approach the optimal solution. Therefore, we can compare the CPU frequency scaling strategy obtained in Section 5.3 with the results obtained from the DP algorithm with high grid density to verify the optimality of the tightest string policy. More results about verification will be shown in Section 8.

6 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 [30]. It has been widely used for resource (e.g., frequency and power) allocation, transceiver optimization, and training optimization in communication systems [31], [32]. The direction water-filling algorithm was first proposed in [33] 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. 6, 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. 6 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. 6, 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 har-
vesting rate to achieve sub-optimal computing performance.

7 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 3.2. To solve P2, we first discretize $p_i$ as introduced in Section 4. 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 $\Delta t$ be 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:

$$P4: \text{arg min}_{P_d[i] \geq 0} N_p,$$

s.t. \begin{align*}
C1: & \sum_{i=0}^{N_p} \beta P_d[i] = M_p, \\
C2: & \sum_{i=0}^{n} P_d[i] \Delta t \leq \sum_{i=0}^{n} E_h[i], \quad n = 0, \ldots, N_p, \\
C3: & \sum_{i=0}^{N_p} E_h[i] - \sum_{i=0}^{n} P_d[i] \Delta t \leq E_{max}, \quad n = 1, \ldots, N_p.
\end{align*}

We can solve P4 based on the following Lemma:

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

Proof. Denote $\mathcal{Z}(P_d, N) = \sum_{i=0}^{N_p} \beta P_d[i]$. Assuming that $P_d[i]$ do not maximize $\mathcal{Z}(P_d[N], N)$, then we can always find $P_d[i]$ making $\mathcal{Z}(P_d[N], N) - \mathcal{Z}(P_d[i], N) = \Delta M_p$, 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 $\mathcal{Z}(P_d[N], N_x) = \mathcal{Z}(P_d[i], N_x)$, $\sum_{j=N_x}^{N_p} \beta P_d[j]$, where $\mathcal{Z}(P_d[i], N_x) = M_p$ and $\sum_{j=N_x}^{N_p} \beta P_d[j] = \Delta M_p$. 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 $\sum_{i=0}^{N_p} \beta P_d[i] \Delta t$.

Based on Lemma 6, the optimization problem P4 can be converted into the optimization problem P3. The only difference between 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 5.3 to find the optimal CPU frequency from $t_0$ to $t_n$.

Step 2: The cloudlet calculates the cumulative frequency. If $\sum_{i=0}^{n} f_2[i] < M_p$, 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 $\sum_{i=0}^{n} f_2[i] = M_p$, 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 $f_2[i], i = 0, \ldots, N_p$, 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. 7 as an example, if the optimization process stops in $t_2$, then the optimal strings are $p_0p_1$, $p_1p_2$, and $p_2p_3$. If the optimization process stops in $t_3$, then the optimal strings are $p_0p_1$, $p_1p_2$, and $p_2p_3$. Obviously, the first two strings, $p_0p_1$ and $p_2p_3$, 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 6. As a result, in most cases, when newly received energy joins the energy feasibility 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 6, 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.

8 Performance Evaluation

This section evaluates the performance of different CPU frequency scaling strategies. We first briefly introduce the simulation configuration. After that, the performance of the DP algorithm under different grid densities is evaluated.
Then, we compare the performance of three different offline strategies. Finally, we study the performance of the online CPU frequency management strategy in different conditions.

8.1 Simulation Setup
In the simulation, we will evaluate the performance of the optimal CPU frequency scaling policy in different conditions, and compare it with the following two frequency management strategies:

- **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 CPU frequency in each time slot is optimized independently. Consequently, the cloudlet will spend all the collected energy at the end of each time slot to maximize short-term computing power. The greedy strategy is essentially an online method because the CPU frequency in the current time slot is only affected by the remaining energy in the energy storage but not related to the energy arriving in the future.

The simulations are carried out on MATLAB. In the simulation, the length of each time slot is $\Delta t = 10$ min. The default value of the average energy harvesting rate is $\bar{p}_h = 96$ W, thus the average amount of energy that the cloudlet can receive in each time slot is $\bar{E}_h = 16$ Wh or $5.76 \times 10^4$ J. 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.

8.2 Performance of DP Algorithm
In Fig. 8, 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 $E_{\text{max}} = 100$ 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 energy feasibility tunnel. The grid density on the X-axis refers to the number of vertexes distributed in each time slot. 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 $10$ simulations.

From Fig. 8, it can be observed that the performance of the DP-based frequency scaling strategy does not always improve as the grid density increases. This is a counterintuitive observation. Specifically, as shown in Fig. 8(a), placing more vertexes on the X-axis of the energy feasibility tunnel will hurt the performance of the DP-based strategy. For example, when the grid density on the Y-axis is $3$, and the grid density on the X-axis is increased from $1$ to $5$, the normalized performance of the DP-based method decreases from $1$ to $0.42$.

The above observations can be explained by the characteristics of the optimal dynamic energy curve. According to Lemma 2, if the dynamic energy curve neither touches the upper bound nor the lower bound of the energy feasibility tunnel, the CPU will not change its clock frequency. Therefore, we only need to place vertexes at the end of each time slot. In this case, the grid density on the X-axis is equal to $1$, and the dynamic energy curve obtained from the DP algorithm will approach the tightest string with the increase of the grid density on the Y-axis, as illustrated in Fig. 9(a). By contrast, as shown in Fig. 9(b), if we increase the grid density on the X-axis to $5$, the dynamic energy curve obtained from the DP algorithm will not perfectly follow the optimal path, resulting in a reduction in the computing power.

For the DP-based frequency scaling strategy, increasing the grid density on the Y-axis can slow down the performance degradation caused by the high grid density on the X-axis. As can be observed from Fig. 8(a), when the grid density on the Y-axis is increased from $1$ to $40$, the normalized performance of the DP-based method decreases from $1$ to $0.42$. However, as the grid density on the X-axis is increased from $1$ to $5$, the normalized performance of the DP-based method decreases from $1$ to $0.42$. This is because the dynamic energy curve obtained from the DP algorithm will approach the tightest string with the increase of the grid density on the Y-axis, as illustrated in Fig. 9(a). By contrast, as shown in Fig. 9(b), if we increase the grid density on the X-axis to $5$, the dynamic energy curve obtained from the DP algorithm will not perfectly follow the optimal path, resulting in a reduction in the computing power.
densities on the X-axis and Y-axis are 4 and 3, respectively, the normalized performance of the DP-based method is only 0.5. However, if the grid density on the X-axis remains unchanged, and the grid densities on the Y-axis are increased to 10 and 40, the normalized performance of the DP-based method can be improved to 0.65 and 0.98, respectively.

Unlike the grid density on the X-axis, increasing the grid density on the Y-axis can always improve the performance of the DP-based frequency scaling strategy. As shown in Fig. 8(b), when the grid densities on the X-axis and the Y-axis are both 3, the normalized performance of the DP-based method can only reach 0.61. However, if we increase the grid density on the Y-axis from 3 to 40, the normalized performance rapidly increases to 0.99.

As shown in Fig. 8, no matter how we change the grid density, the normalized performance of the DP-based frequency scaling strategy is less than 1, which means that the performance of the tightest string policy is always higher than that of DP-based method. This indicates that the tightest string is indeed the optimal policy to maximize the computing power of the SCE cloudlet.

8.3 Performance of Offline Strategies

In Fig. 10, we evaluate the impact of the energy storage capacity on the average clock rate (\(\bar{f}_c\)) of different CPU scaling strategies. The amount of energy harvested in each time slot follows a truncated Gaussian distribution that lies between 0 Wh and 50 Wh with a mean of 16 Wh and a standard deviation of 10 Wh. The truncated Gaussian distribution guarantees that the energy harvesting rate is a positive value less than 300 W, which is the peak energy harvesting rate that a 2.5 m² solar panel with 20% energy conversion efficiency can be achieved in an unshadowed area. The results presented in the figure are the average of 10 simulations.

As illustrated 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. In addition, the change of \(E_{\text{max}}\) has much less impact on the optimal strategy and the greedy strategy than the average policy. As a result, in the optimal strategy and the greedy strategy, \(\bar{f}_c\) is almost a constant relative to \(E_{\text{max}}\). For example, when the maximum capacity of energy storage increases from 55 Wh to 130 Wh, \(\bar{f}_c\) with the average frequency manage strategy increases from 2.22 GHz to 2.43 GHz. In this case, the computing power of the cloudlet is improved by 8.6%. On the contrast, if the optimal strategy is adopted, \(\bar{f}_c\) slightly increases from 2.65 GHz to 2.71 GHz. As a result, there is only a 2.5% improvement in computing power.

Compared with the optimal strategy, the performance of the greedy strategy is relatively low due to the 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 advantage of this strategy is that energy never overflows from the energy storage. Therefore, all arriving energy can be successfully received by the cloudlet, thus achieving high energy harvesting efficiency. The disadvantage is that the energy collected in past time slots cannot be saved for future use, resulting in low energy utilization.

Compared with the greedy strategy, the average CPU frequency scaling policy has higher energy efficiency due to the lower average dynamic power. However, the energy harvesting efficiency of the average strategy is relatively low because it uses the collected energy in a conservative manner. As a consequence, the dynamic power of the average strategy may be lower than the energy harvesting rate for a long time. In this situation, the energy storage may be fully charged, causing energy reception failure in the following time slot. Increasing the maximum capacity of the energy storage can improve the performance of the average CPU frequency scaling strategy, making it gradually approach the optimal solution by reducing the probability of the energy storage being completely filled.

In Fig. 11, we evaluate the impact of energy changes on the performance of the three CPU frequency management strategies. In the figure, the maximum capacity of the energy storage and the average energy harvesting rate are set to \(E_{\text{max}} = 85\text{ Wh}\) and \(p_h = 16\text{ Wh}\), respectively. Denote the standard deviation of \(E_h\) as \(\sigma_{h}\), which can indicate the variation of energy intensity between different time slots. In order to clearly show the relationship between the performance of the three strategies and \(\sigma_{h}\), 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 energy feasibility tunnel should be the one that allows the CPU frequency to remain constant throughout 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 energy feasibility tunnel needs to be consistent, which requires the fluctuation of the energy intensity to be as small as possible. Otherwise, the slope of
computing power is only reduced by 6.1%. This is because, the average clock rate that is, the computing power decreases by 16.7%. In the same situation, the average clock rate of the greedy strategy is slightly reduced from 2.61 GHz to 2.45 GHz, that is, the computing power is only reduced by 6.1%. 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.

8.4 Performance of Online Strategy

For the online CPU frequency scaling strategy, 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 but there are some errors. In the past few decades, how to predict the intensity of solar and wind energy has been extensively studied [34]–[36], which is out of the scope of this paper.

We start from the first case, where the cloudlet can perfectly predict the energy harvesting rates in the next N_f time slots. To manage the CPU frequency, the energy feasibility tunnel is equally divided into multiple sub-tunnels, and each of them contains N_f time slots. Then, the online strategy runs the tightest string policy independently in each sub-tunnel to calculate the dynamic power and the corresponding clock rate.

Fig. 12 shows the performance of the online strategy with respect to N_f. In the figure, the maximum capacity of the energy storage is set to E_{max} = 85 Wh. Moreover, the normalized performance is defined as the ratio of the average clock rate obtained by the online strategy to that obtained by the optimal offline strategy.

As illustrated in Fig. 12, the performance of the online strategy is proportional to N_f. With the increase N_f, the performance of the online strategy will approach to the optimal solution quickly. Moreover, the 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 this case, the online strategy can still achieve a good performance. For instance, when σ_h are 3.2 Wh and 7.2 Wh, the normalized performance of the online strategy can reach 0.997 and 0.986, respectively, almost the same as the optimal offline 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. The prediction error is generally small while predicting the harvest rate in the next adjacent slot and grows with the slot increases. Here, we consider a linear error model where the prediction error, e = η + ρ(n − 1), with n = 1, · · · , N_f. In order to investigate the impact of prediction error on the performance of online CPU scaling strategy, we vary η and ρ, then plot the normalized performance in Fig. 13. In the figure, the maximum capacity of the energy storage is E_{max} = 85 Wh and the average energy variation is σ_h = 3.2 Wh.

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 σ_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.

Now, we investigate the impact of ρ on the performance of online strategy in Fig. 13(b). When ρ = 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 ρ = 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 $\rho$ 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 ($\sigma$) and the initial prediction error ($\eta$) have the most significant impact on the performance of the online CPU scaling strategy. To reduce $\sigma$ and $\eta$, the most efficient way is to decrease the length of each time slot ($\Delta t$). This is because the change of energy intensity is usually a continuous process, reducing $\Delta t$ will make the adjacent measures of the energy intensity highly correlated. In this case, using the greedy strategy can achieve similar a performance as the offline optimal CPU frequency scaling policy.

9 Conclusions

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 how to convert the optimal frequency scaling problem into a routing problem that can be solved by the DP algorithm. Finally, how to design an online strategy for frequency management is discussed. Simulation results show that the performance of the online strategy is affected by the prediction accuracy of the energy intensity and the variation of the sustainable energy in the surrounding environment. Reducing the interval of the time slot helps to improve the performance of the online CPU scaling policy.

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

**REFERENCES**


