Received 28 April 2024, accepted 10 May 2024, date of publication 17 May 2024, date of current version 24 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3402611

RESEARCH ARTICLE

An Online Simulated Annealing-Based Task Offloading Strategy for a Mobile Edge Architecture

AYEH MAHJOUBI, ARUNSELVAN RAMASWAMY, AND KARL-JOHAN GRINNEMO, (Senior Member, IEEE)
Department of Computer Science and Mathematics, Karlstad University, 651 88 Karlstad, Sweden

Corresponding author: Ayeh Mahjoubi (ayeh.mahjoubi@kau.se)

This work was supported in part by the Knowledge Foundation of Sweden through the Data-Driven Latency-Sensitive Mobile Services for a Digitalized Society (DRIVE) under Grant 20220072. The work of Arunselvan Ramaswamy was supported in part by the Knowledge Foundation under Grant 20200164.

ABSTRACT This paper introduces SATS, an innovative online task scheduling strategy tailored for hierarchical Mobile Edge Computing (MEC) architectures. Leveraging a Simulated Annealing-based approach, SATS extends the applicability of this method beyond offline scheduling, demonstrating its efficacy in real-time task allocation. Crucially, the paper underscores the pivotal role of accurate service request predictions in enhancing SATS' performance. Through a comparative analysis of three prediction models, neutral, conservative, and optimistic, the paper reveals that SATS achieves optimal outcomes when paired with a conservative predictor, which deliberately overestimates service request volumes. In fact, employing this predictor yields significantly higher acceptance rates and reduced processing times. Remarkably, SATS, coupled with a conservative predictor, achieves an acceptance ratio within a mere 5% deviation from an ideal scenario where the frequency of service request arrivals is known in advance, maintaining consistency across various experimental scenarios.

INDEX TERMS Online task scheduling, simulated annealing, mobile edge computing, task offloading.

I. INTRODUCTION

The 5G networks are being deployed worldwide, enabling advanced communication services that will transform society. Cellular IoT (CIoT) is a crucial element of 5G that will help digitize society by facilitating large-scale communication between machines and people. CIoT has many applications, from basic tasks like asset tracking and smart metering to more complex ones like AR/VR drones and even highly demanding ones like autonomous vehicles and collaborative robotics. CIoT devices collect and transmit vast amounts of data that need to be analyzed and processed somewhere. That is where Mobile Edge Computing (MEC) comes in. MEC extends cloud computing to the Edge of the 5G network, reducing the load on core cloud data centers and moving it closer to CIoT devices. The Edge is a growing business projected to reach $445 billion by 2030, with investment in Edge data centers set to exceed $140 billion by 2028 [1]. Efficient task-offloading is essential for the successful implementation of MEC in CIoT. It involves computing the optimal or near-optimal way of offloading service chain tasks between the CIoT device, Edge servers, and the core cloud to ensure efficient service processing time. Many task-offloading approaches have been proposed, but most are offline approaches that assume all services arriving at the CIoT devices are known in advance. This assumption is unrealistic in several cases since the arrival of services is unpredictable and varies with time.

This paper presents a new online strategy called SATS (Simulated Annealing Task Scheduling) that uses Simulated Annealing to determine the optimal or nearly optimal way to offload tasks in a three-layer MEC

The associate editor coordinating the review of this manuscript and approving it for publication was Valerio Freschi.

architecture. The architecture has a bottom layer housing the CIoT devices, a mid-layer with the Edge servers, and cloud data centers in the top layer. SATS is both theoretically sound and practically applicable, as it effectively determines how to offload tasks in services that run on CIoT devices, minimizing service processing times.

The main contributions of the paper are as follows:

- The paper examines a variety of user devices that invoke different numbers of services.
- Different types of services are considered, taking into account the number of tasks in each service and their specific requirements based on the service type.
- A computationally efficient strategy for online task offloading in a MEC architecture, SATS, is proposed.
- It is shown that Simulated Annealing can be used not only for offline task offloading but also for online task scheduling.
- SATS is demonstrated to optimally or near-optimally offload tasks in a three-layer MEC architecture.

The remainder of the paper is structured as follows: Section II provides preliminaries that include MEC, meta-heuristic optimization, and Simulated Annealing. Section III presents the three-layer MEC architecture used to assess SATS. The task offloading problem is mathematically formulated in Section IV. The proposed task offloading strategy, SATS, is described in Section V, and its evaluation is discussed in Sections VI and VII. In Section VIII, the authors survey related work and compare and contrast these works with SATS. Finally, the paper is concluded in Section IX with a summary and outlook for further research.

II. BACKGROUND

This section provides an introduction to MEC and meta-heuristic optimization. Section II-A explains Edge computation and MEC, while Section II-B provides an overview of metaheuristics, focusing on Simulated Annealing.

A. MOBILE EDGE COMPUTING

MEC is a cutting-edge technology under the 5G umbrella. It enables cloud-based network resources and services, such as processing, storage, and networking, to be provided at the network's Edge, i.e., closer to the CIoT devices. The Edge can refer to the base stations themselves, as well as data centers located near the radio network at aggregation points.

Figure 1 illustrates the general architecture of MEC. Cloud servers offer CIoT devices powerful computing resources and pre-installed software and libraries, such as machine learning support that allows mobile services to support more sophisticated and complex end-to-end applications that provide a better CIoT device experience. However, applications may experience higher end-to-end delays due to long network latencies between CIoT devices and cloud data servers.

The MEC layer is located between the mobile devices and the cloud. It includes several macro base stations that typically host city-level data centers containing considerable computation and storage resources that can host multiple virtual machines and containers simultaneously.

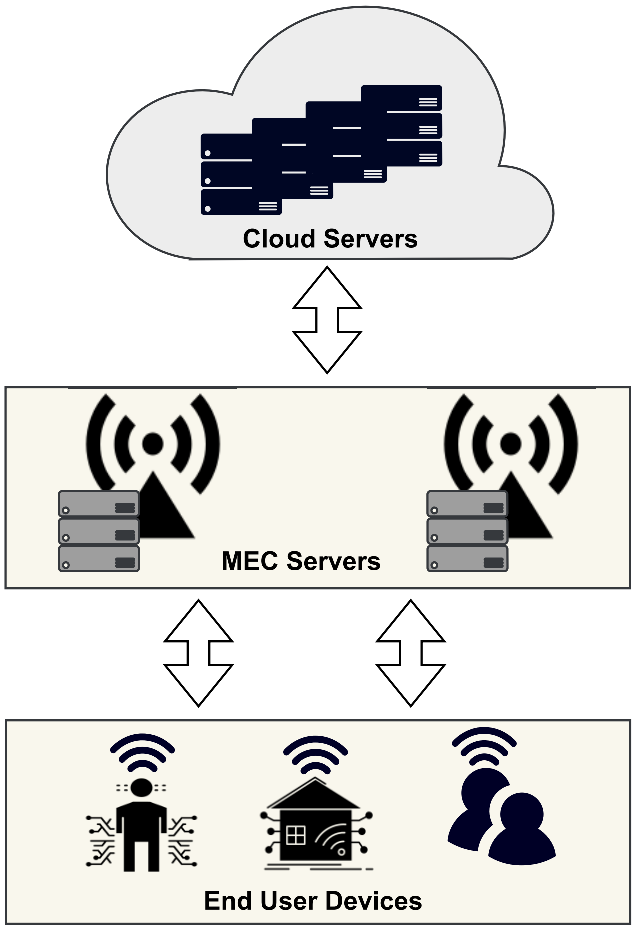


FIGURE 1. The general MEC architecture is structured into three layers: the bottom layer for CIoT devices, a mid-layer housing Edge servers, and the top layer comprising cloud data centers.

It mainly complements cloud computing to support and enhance the performance of mobile subscribers. The mobile Edge servers are computing equipment installed at or near base stations. Unlike centralized cloud servers or peer-to-peer mobile devices, MEC is managed locally by the network operator.

With MEC, the end-to-end latency perceived by mobile CIoT devices can be significantly reduced, which is crucial for many 5G services like the tactile internet. MEC can be deployed in multiple ways, and MEC servers can be located in different areas of the radio access network based on technical and business requirements.

MEC provides an ultra-low-latency environment that delivers mission-critical and real-time services by performing analytics or caching content on the MEC server. MEC servers are not affected by congestion in other parts of the network, and data traffic can be minimized by reducing the volume of data transmitted to the core network for processing.

This results in the more efficient use of existing network bandwidth.

B. OPTIMIZATION, METAHEURISTICS, AND SIMULATED ANNEALING

Optimization is searching for designs and strategies that meet or exceed specific objectives while satisfying numerous constraints. In the simplest form, optimization aims to find an objective function's maximum or minimum value within a feasible range or space of variables. However, finding solutions close to the global optimum of a complex optimization problem with traditional methods is usually impossible. Therefore, metaheuristic algorithms are generally used to calculate near-optimal solutions. These algorithms are called metaheuristics because they are higher-level methods or heuristics used to discover, produce, or select a heuristic that is a good enough solution for the given optimization problem.

Simulated Annealing is a metaheuristic optimization technique that imitates the physical annealing of solids, as proposed by Kirkpatrick in 1983 [2]. The process involves heating and gradually cooling a solid until it achieves its most structured crystal lattice arrangement without any flaws.

The Simulated Annealing algorithm starts by creating a random feasible solution: the system's current state. A suitable procedure generates a new state and evaluates its fitness value. It is accepted if the new state is better than the current one. However, if the new state is inferior, it can still be accepted based on a probabilistic function called the acceptance function, which relies on the system's temperature.

The algorithm generates a certain number of new states at each temperature while progressively decreasing the temperature. Several new solutions are attempted until a thermal equilibrium criterion is satisfied at each temperature. At that point, the temperature decreases again. As the temperature reduces, the likelihood of selecting non-improving movements diminishes. This cycle of generating new solutions as the system cools down is repeated until the termination condition is met.

III. SYSTEM MODEL AND REPRESENTATION

This study considers a three-layer MEC architecture consisting of a CIoT device layer, a MEC layer, and a cloud layer. In the Edge layer the Edge node is located near a base station, which enables it to serve Cellular Internet of Things (CIoT) devices within its coverage area. Additionally, it is connected to the cloud, allowing all CIoT devices to access it via the Edge node.

In the framework, a heterogeneous network comprising various nodes is considered, including CIoT nodes, an Edge computing node, and a cloud node. The set of μ CIoT devices is represented by $\mathcal{M} = \{m_1, m_2, \ldots, m_\mu\}$, the Edge node is represented using $\mathcal{E}$, and $\mathcal{C}$ represents the cloud. A matrix - RN_n - is used to represent the available CPU (MIPS), RAM (Gb), and network (Gbps) resources for all nodes. The matrix rows represent the available CPU, RAM, and network resources for each node, while the columns represent the same resources for the nodes. For example, in RN_n,

$$RN_n = \begin{bmatrix} C_{n_1} & C_{n_2} & \cdots & C_{n_n} \\ R_{n_1} & R_{n_2} & \cdots & R_{n_n} \\ N_{n_1} & N_{n_2} & \cdots & N_{n_n} \end{bmatrix}, \tag{1}$$

where $n = m_1$, C_{n_1} represents the amount of CPU resource of the first CIoT device, and in RN_E, where $n = \mathcal{E}$, $C_{\mathcal{E}}$ represents the amount of CPU resource for the Edge node.

A matrix RL_l (2) is also defined, representing the available bandwidth, $\alpha(Gbps)$, and delay, $\beta(s)$, on the links between CIoT devices and the Edge node, as well as the links between the Edge node and the cloud. The rows correspond to the Edge nodes, while the columns correspond to CIoT devices. For example, $(\alpha, \beta)_{l_x, y_1}$ in matrix RL_l where $l = EM$ defines the amount of bandwidth and the delay of the link between the Edge node and the first CIoT device.

$$RL_l = \left[(\alpha, \beta)_{l_x, y_1} \cdots (\alpha, \beta)_{l_x, y_j}\right], \tag{2}$$

The set of available services in the system is denoted as $\mathcal{S} = (s_1, s_2, \ldots, s_s)$, while the set of sequential tasks comprising each service is represented by $S^t = \{1, 2, \ldots, t\}$. Additionally, each service is associated with a predefined deadline, d_{s_s}. The amount of CPU, RAM, and network resources required to complete a task in a service is denoted by matrix, RQ:

$$RQ = \begin{bmatrix} C_{s_1^1} & C_{s_1^2} & \cdots & C_{s_s^t} \\ R_{s_1^1} & R_{s_1^2} & \cdots & R_{s_s^t} \\ N_{s_1^1} & N_{s_1^2} & \cdots & N_{s_s^t} \end{bmatrix}. \tag{3}$$

As in RN_n, the rows represent the CPU, RAM, and network resources, and the columns represent the service tasks. For example, $C_{s_1^2}$ represents the CPU required to complete the second task of the first service.

In the system, time is divided into discrete time windows, w_t, where

$$w_t = \tau \times \iota. \tag{4}$$

Each time window, w_t, comprises multiple time slots, denoted as τ, with a fixed time interval, ι (seconds).

Each CIoT node requests several services at certain rates (frequencies). To manage these requests efficiently, the system incorporates a scheduler and each CIoT device is equipped with an associated execution queue. The scheduler receives IoT node requests at the start of each time window, and it processes these requests, allocating them across individual CIoT device queues based on their corresponding requirements.

In the study, two scenarios concerning the frequency of service requests from CIoT devices are explored. The first scenario considers a variable true frequency, f_t, which changes periodically across different time slots. This variability reflects a more dynamic model where the frequency of service requests is not constant but adapts over time, ensuring a more realistic representation of CIoT device behavior.

Consequently, the set of requests dispatched to the scheduler at the beginning of each time window is not identical but varies in accordance with the changes in f_t. In the second scenario, the impact of prediction errors on scheduling is simulated by introducing a degree of uncertainty into the true frequency. To model this, the true frequency is adjusted by adding a Gaussian noise component, resulting in a modified frequency, f_p. The equation for this adjustment is given by $f_p = f_t + \mathcal{N}(0, \sigma^2)$, where $\mathcal{N}(0, \sigma^2)$ represents the Gaussian noise with a mean of zero and a specified variance σ^2. This zero-mean Gaussian noise is added to the true frequency to simulate the inaccuracies and variability that might arise in predicting service request frequencies. The low variance σ^2 ensures that the introduced errors are subtle, reflecting realistic scenarios where predictions are generally close to the actual values but not exact. This method allows us to examine how such prediction errors might influence the allocation and scheduling of CIoT device requests across varying time windows.

A solution vector encapsulates the scheduled locations for each task corresponding to every service requested by CIoT devices.

For each task, t, in service, s, requested by CIoT device, m_i, a variable, $x_{m_i}^{s_s^t}$, is introduced, where $x_{m_i}^{s_s^t} \in \{0, 1, 2\}$. This categorization is achieved through a 3-digit coding system: 0 denotes task allocation to the CIoT device, 1 indicates task allocation to the Edge node, and 2 represents task allocation to the cloud.

IV. PROBLEM DEFINITION

As previously discussed, the system operates across multiple time windows. At the start of each window, requests from all CIoT devices are received. To manage these requests, the scheduler employs a First-In-First-Out (FIFO) mechanism, allocating each request to its respective execution queue. This FIFO approach ensures that requests are processed in the order they are received, with the earliest request being prioritized for execution.

Subsequently, each task associated with a requested service in the CIoT device queue requires scheduling for either local execution by the CIoT device or potential offloading to the Edge or cloud, contingent upon available resources. Pending tasks await resource availability, remaining in the queue until resources are released by completed tasks. It's imperative to note that each service must be completed within its specified deadline for acceptance; otherwise, it will be dropped. This paper aims to establish an optimal scheduling solution to minimize the total system processing time while maximizing the acceptance ratio.

The acceptance ratio, denoted as η, quantifies the fraction of successfully processed requests within a given time window, given by:

$$\eta = \frac{\varphi}{\phi}, \tag{5}$$

where φ represents the number of requests completed within the time window, and ϕ denotes the total number of unfinished requests at the beginning of that window.

The initial step in calculating the system total delay, T_{total}, is to compute the combined delay of all tasks in a particular service, $T_d^{s_s^t}$, which is a function of the following components:

- *Waiting time:* The time spent in the queue before execution, which is computed as

$$t_q^{s_s^t} = t_{start}^{s_s^t} - t_{arrival}^{s_s^t}. \tag{6}$$

  Upon the arrival of a requested service, the scheduler marks the initiation by timestamping all tasks belonging to that service, $t_{arrival}^{s_s^t}$. As each task begins its execution, an additional timestamp is added to denote the task's start time, $t_{start}^{s_s^t}$.
- *Processing time:* The processing time of each task represents the time required to complete the task on the scheduled node (such as an IoT device, an Edge node, or in the cloud). The task processing time is computed as

$$t_{ex}^{s_s^t} = \frac{C_{s_s^t}}{C_{x_n}}, x_n \in \{0, 1, 2\}. \tag{7}$$

  For instance, when $x_n = 1$, this equation calculates the processing time of task, t, for service, s, on an Edge node, taking into account the CPU capacity of that Edge node.
- *Transmission delay:* A transmission delay occurs when a task is offloaded to a node different from the location of the previous task and is computes as

$$t_{tr}^{s_s^t} = \frac{N_{s_s^t}}{\alpha_l} + \beta_l. \tag{8}$$

  The task's transmission delay, $t_{tr}^{s_s^t}$, is computed as the quotient between the requested task's data size, $N_{s_s^t}$, and the available bandwidth of the link,α_l, added with the link delay, β_l.

Given that the transmission delay for a single task, $t_{tr}^{s_s^t}$, has been calculated, the overall delay of a single task t in a service s, $T_d^{s_s^t}$, as

$$T_d^{s_s^t} = t_q^{s_s^t} + t_{ex}^{s_s^t} + t_{tr}^{s_s^t}. \tag{9}$$

Next, the overall delay for a service, $T_d^{s_s}$, is calculated by summing up the overall delays for all tasks within that service,

$$T_d^{s_s} = \sum_{t \in \{1,2,\ldots,t\}} T_d^{s_s^t}. \tag{10}$$

Finally, T_{total} is calculated as the sum of all the delays of the services requested by all CIoT devices,

$$T_{total} = \sum_{m \in \mathcal{M}} \sum_{t \in \{1,2,\ldots,t\}} T_d^{s_s^t}. \tag{11}$$

Given the knowledge on how to calculate T_{total}, the optimization problem to be solved by SATS can be formulated. The problem is formulated using the parlance of consensus optimization, where the aim is to find x, a common solution that both maximizes the acceptance ratio, $\eta(x)$, and minimizes the total system delay, $T_{\text{total}}(x)$. The set of all feasible solutions is represented by $\mathcal{X}$. The optimization problem can then be written:

$$\text{Find } x \text{ such that: } \max_{x\in\mathcal{X}} \eta(x) \wedge \min_{x\in\mathcal{X}} T_{\text{total}}(x). \tag{12}$$

This optimization problems is governed by a set of constraints that ensure feasibility and adherence to practical limitations. These constraints are as follows:

$$\sum_{S\in n_i}\sum_{s_s\in S}\sum_{t\in s_s}\begin{bmatrix} C_{x_{n_i}^{s_s^t}} \\ R_{x_{n_i}^{s_s^t}} \\ N_{x_{n_i}^{s_s^t}} \end{bmatrix} \le \begin{bmatrix} C_{n_i} \\ R_{n_i} \\ N_{n_i} \end{bmatrix}, \forall x_{n_i}^{s_s^t} = 0, \forall n_i \in \mathcal{M} \tag{13}$$

$$\sum_{S\in n_i}\sum_{s_s\in S}\sum_{t\in s_s}\begin{bmatrix} C_{x_{n_i}^{s_s^t}} \\ R_{x_{n_i}^{s_s^t}} \\ N_{x_{n_i}^{s_s^t}} \end{bmatrix} \le \begin{bmatrix} C_{n_i} \\ R_{n_i} \\ N_{n_i} \end{bmatrix}, \forall x_{n_i}^{s_s^t} = 1, \forall n_i \in \mathcal{E} \tag{14}$$

$$\sum_{S\in n_i}\sum_{s_s\in S}\sum_{t\in s_s}\begin{bmatrix} C_{x_{n_i}^{s_s^t}} \\ R_{x_{n_i}^{s_s^t}} \\ N_{x_{n_i}^{s_s^t}} \end{bmatrix} \le \begin{bmatrix} C_{n_i} \\ R_{n_i} \\ N_{n_i} \end{bmatrix}, \forall x_{n_i}^{s_s^t} = 2, \forall n_i \in \mathcal{C} \tag{15}$$

$$\sum_{n_i\in\mathcal{M}}\sum_{S\in n_i}\sum_{s_s\in S}\sum_{t\in s_s}\frac{N_{s_s^t}}{\alpha_{x_{n_i}^{s_s^t}}} \le 1, \forall x_{n_i}^{s_s^t} \in \{0,1,2\} \tag{16}$$

$$T_d^{s_s} \le d_{s_s} \quad \forall s \in \mathcal{S} \tag{17}$$

Equations 13, 14, and 15 make sure that the total resources consumed by tasks from all CIoT device-requested services, do not surpass the resource capacity of any given node.

Equation 16 ensures that the resources available on all links are greater than or equal to those being utilized, and Equation 17 guarantees that all accepted requests are completed within their respective deadlines.

V. SIMULATED ANNEALING TASK SCHEDULING

As previously mentioned, at the start of each time window, CIoT device requests are received that should be scheduled in a way that maximizes the acceptance ratio while minimizing the processing time. The proposed task-scheduling strategy, SATS, is designed to be easy to implement, computationally lightweight, and be able to find near-optimal solutions to the optimization problem formulated in Equation 1.

Figure 2 provides a flow chart over the SATS strategy. As follows, SATS is initialized with an arbitrary feasible solution, and proceeds by improving its current solution in an iterative manner. Let χ denote the current solution. With χ as starting point, a potentially 'better' solution in the neighborhood of χ, Υ, is found. To find Υ, SATS modifies

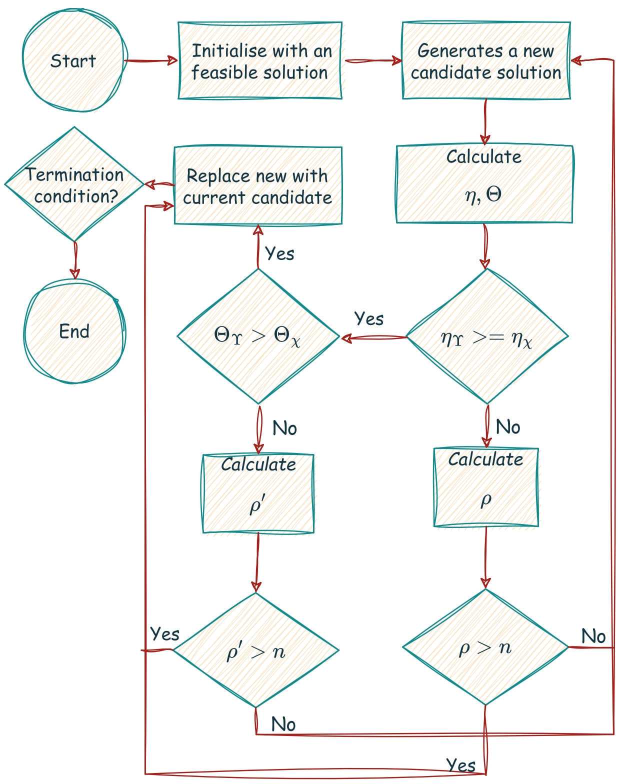

Start
Initialise with an feasible solution
Generates a new candidate solution
Termination condition?
Replace new with current candidate
Calculate η, Θ
End
Yes
$\Theta_\Upsilon > \Theta_\chi$
Yes
$\eta_\Upsilon >= \eta_\chi$
No
No
Calculate ρ'
Calculate ρ
Yes
$\rho' > n$
$\rho > n$
No
No
Yes

FIGURE 2. Flow chart over the SATS task-scheduling strategy.

the scheduling decisions of χ. Once found, Υ is evaluated as follows: First, Υ's processing time, Θ_Υ, and acceptance ratio, η_Υ, are calculated. Second, it is decided whether Υ should be the next current solution, i.e., the next χ. As follows, Υ is accepted as the next χ provided it improves the acceptance ratio over χ, or has an processing time that is at most that of χ. In those cases when the acceptance ratio of Υ is indeed higher than that of χ, but so is also the processing time, SATS accepts Υ with probability ρ, given by

$$\rho = \exp\left(\frac{\Theta_\chi - \Theta_\Upsilon}{\varsigma_{exc}/count}\right), \tag{18}$$

where ς_{exc} represents the initial processing time and is also the first of two 'temperature' parameters employed by SATS.

If the acceptance ratio of Υ is lower than that of χ, it is accepted as the new χ with a probability, ρ', which is computed as

$$\rho' = \exp\left(\frac{\eta_\Upsilon - \eta_\chi}{\varsigma_{ac}/count}\right), \tag{19}$$

where ς_{ac} is the initial acceptance ratio and the second 'temperature' parameter employed by SATS.

SATS iteratively refines solutions until reaching a pre-established termination condition, i.e., when (a) the acceptance ratio and the processing time stabilizes, exhibiting minimal variance over several iterations, or (b) when a predefined number of iterations is reached.

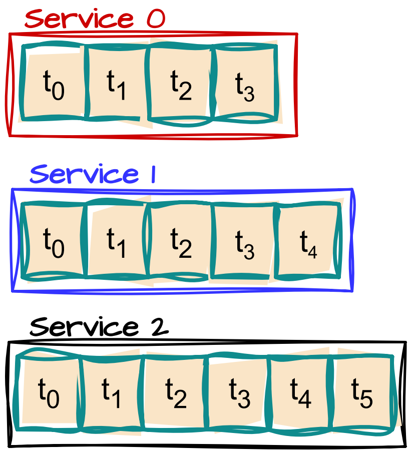


FIGURE 3. Three service types with 4, 5, and 6 tasks.

VI. EXPERIMENT SETUP

The system is structured into three distinct levels: the CIoT level, encompassing all CIoT devices; the edge level, housing the edge server; and the cloud level, representing the cloud server. Each CIoT device, edge server, and cloud server is equipped with three distinct resources: CPU, RAM, and Network. Furthermore, the links between CIoT devices and the edge server, as well as between the edge server and the cloud, are characterized by specific amounts of link bandwidth and delay. Throughout the experiments, the system was configured to accommodate three types of services, each comprising 4, 5, and 6 tasks, respectively (as depicted in Figure 3). The services were subject to strict deadlines, requiring completion within 2 - 3 time windows to avoid being discarded. The demand for services in the system fluctuated as CIoT devices made requests with varying frequencies.

An experiment comprised 20 time windows, each containing 100 time slots of 0.2 s each. All service request frequencies were considered to be established at the one set of each time window. The request frequencies were translated into an arrival sequence within the time window. As shown in Figure 4, requests were orderly queued in a FIFO manner within the CIoT devices' queues according to their arrival times, which enabled a structured approach to processing.

At the start of an experiment, the current solution of SATS was initialized to a randomly selected feasible solution, which was iteratively improved to maximize the acceptance ratio while simultaneously minimize the processing time. At the beginning of each time window, when the service requests arrived, the initial solution of SATS was set to the optimal solution found in the previous time window. In so doing, i.e., by utilizing the correlation between service request frequencies in consecutive time frames, the SATS algorithm was able to be more effective.

In the experiments, it was assumed that SATS employed a Machine Learning-based predictor, pre-trained on past service traffic data, to predict the service frequencies. The output of the Machine Learning-based predictor was modeled

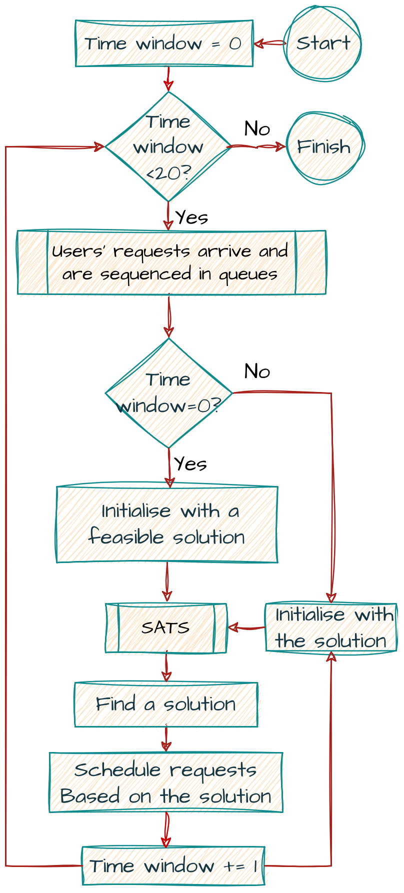

Start
Time window = 0
Time window <20?
No
Finish
Yes
Users' requests arrive and are sequenced in queues
Time window=0?
No
Yes
Initialise with a feasible solution
SATS
Initialise with the solution
Find a solution
Schedule requests Based on the solution
Time window += 1

FIGURE 4. Flow chart over the experiment.

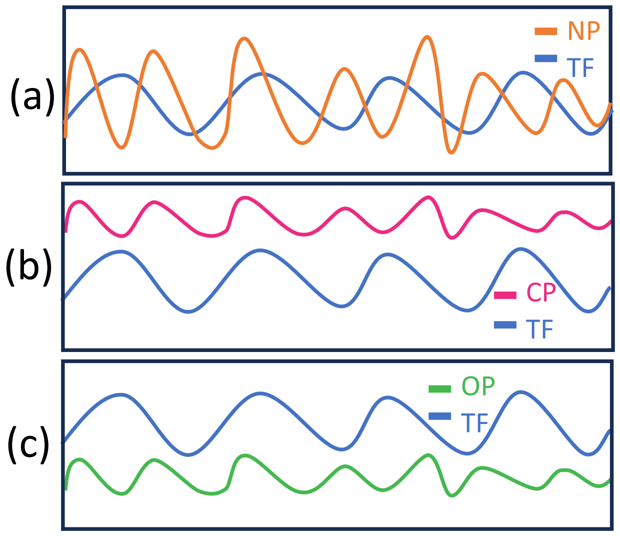


FIGURE 5. Three types of predictions of the true service frequency (TF) were considered in the experiments: 'neutral prediction' (NP), 'conservative prediction' (CP), and 'optimistic prediction' (OP).

as the sum of the actual or 'true' service frequency, f_{TF}, and a Gaussian error function, $\mathcal{N}(0, \sigma^2)$.

Three types of predictions were considered, as depicted in Figure 5: (a) a 'neutral prediction' (NP),

$f_{NPF} = f_{TF} + \mathcal{N}(0, \sigma^2)$, which permitted both over- and underestimation of the true service frequency; (b) a 'conservative prediction' (CP), $f_{CPF} = f_{TF} + max\{0, \mathcal{N}(0, \sigma^2)\}$, that overestimated the true service frequency; and (c) an 'optimistic prediction' (OP), $f_{OPF} = f_{TF} + min\{\mathcal{N}(0, \sigma^2), 0\}$, that underestimated the true service frequency.

All system setups were implemented using Python 3.11, utilizing the features of a macOS Monterey environment running on an M1 chip with 16 GB of memory.

VII. NUMERICAL STUDIES

This section discusses the results of the experiments conducted to evaluate the performance of SATS using the previously mentioned three types of service frequency predictions: neutral, conservative, and optimistic. The experiments were based on two evaluation metrics - total processing time and acceptance ratio. The experiments were divided into two categories. SATS and the true frequency were used as benchmarks, assuming that the predictor could always predict the actual service frequency.

The first category of experiments, Category A, aimed to evaluate the performance of SATS across successive time windows by varying the number of CIoT devices over time. This was done to simulate the dynamics of the system.

The second category of experiments, Category B, studied the system's response to fluctuations in service demand by keeping the number of CIoT devices constant and varying the service frequency.

One significant difference between the two categories of experiments was that in Category A, the amount of available resources increased as the number of CIoT devices increased. However in category B, the amount of available resources remained constant.

A. VARYING THE NUMBER OF CIOT DEVICES

In the first experiment, the performance of SATS was observed by decreasing the number of CIoT devices in the system. The frequency of service requests was decreased by removing more devices at every fourth time window, from 240 to 160. Subsequently, the number of devices was increased by adding them at every fourth time window, from 160 to 240.

Figure 6 (a) displays the arrival rate of service requests over time, while Figures 6 (b) and (c) show the acceptance ratio and processing time for different predictors.

It was found that the conservative predictor performed the best. When this predictor was used, the acceptance ratio never deviated by more than 8% from the optimal value, and the processing time was never more than 10% longer than that obtained with the true frequency. In contrast, the optimistic predictor performed the worst, with acceptance ratios sometimes 26% worse than optimal.

These results can be explained as follows. The conservative predictor overestimated traffic, which caused SATS to allocate more resources than were necessary. The optimistic predictor, on the other hand, underestimated traffic, which resulted in SATS allocating fewer resources than were required.

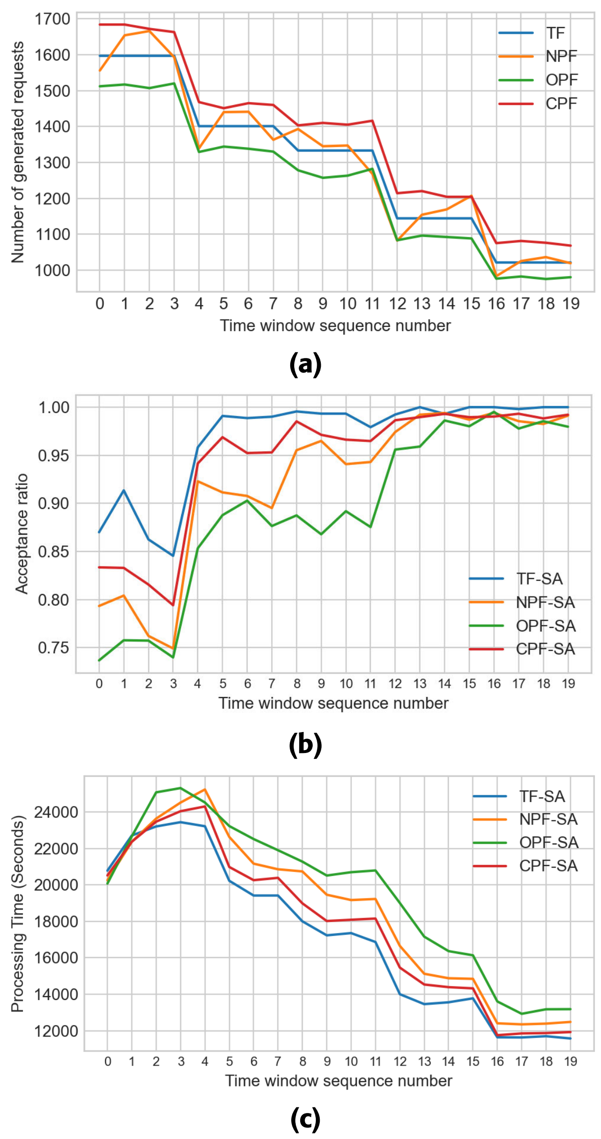


FIGURE 6. In the first Category A experiment, the frequency of service requests decreased by removing CIoT devices in every four time windows. The graphs depict (a) the number of requests that arrived in each time window, (b) the acceptance ratio, and (c) the processing time.

In the second experiment, the performance of SATS was tested as more CIoT devices were gradually added to the system. The process started with 160 devices for the first three time windows (0-(3), increased to 180 devices for the following four time windows (4)-(7), and continued in this manner until reaching 240 devices in the last four time windows (16)-(19). The results of this study are presented in Figure 7, following the same presentation format as for the first experiment.

It was observed that the conservative predictor performed the best, while the optimistic predictor performed the worst.

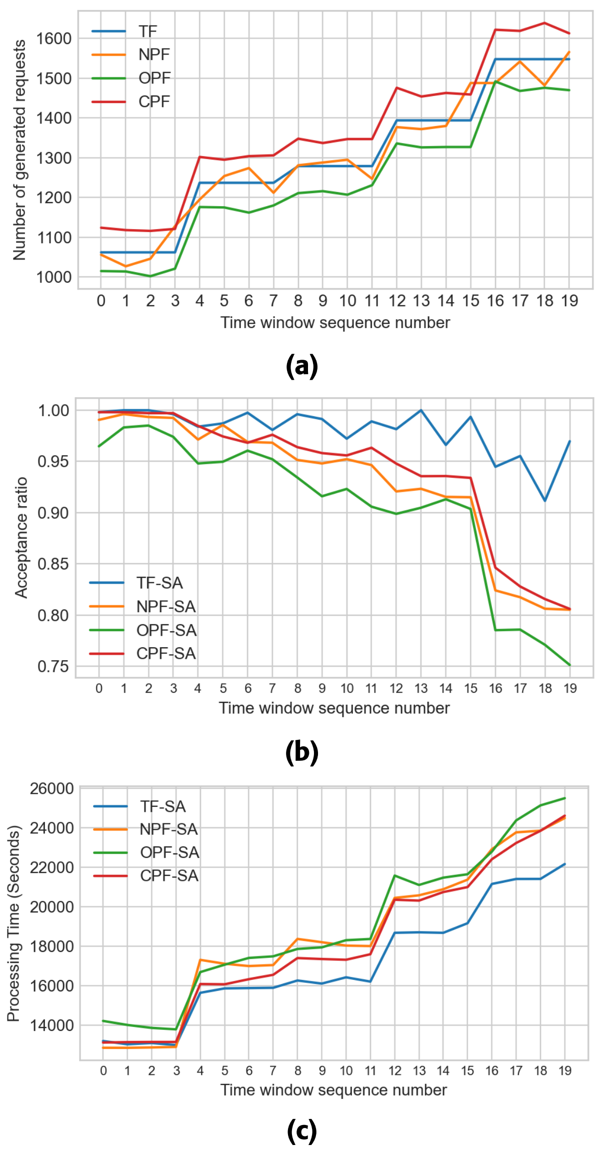


FIGURE 7. In the second Category A experiment, the frequency of service requests gradually increased by adding more CIoT devices in every four time windows. The graphs depict (a) the number of requests that arrived in each time window, (b) the acceptance ratio, and (c) the processing time.

When SATS used the conservative predictor, the acceptance ratio was never more than 16% different from the acceptance ratio obtained with the true frequency. However, when SATS relied on the optimistic predictor, it sometimes resulted in an acceptance ratio that was 21% less than the one obtained with the true frequency. Additionally, when SATS used the optimistic predictor, it consistently had longer processing times than with the other two predictors.

Interestingly, the neutral predictor resulted in almost as high acceptance rates as the ones obtained with the conservative predictor. The explanation for this result is In Figure 7 and particularly in subfigure (a), it's observed that the NFT closely aligns with the TF, allowing for accurate request predictions despite occasional underestimations or overestimations within specific time windows. This similarity results in the performance of NPF-SA closely approaching that of CPF-SA. However, any degree of underestimation, even if minimal, detracts from achieving the optimal performance exhibited by CPF-SA.

B. VARYING THE SERVICE FREQUENCY

In the first experiment, the frequency of service requests was gradually increased while keeping the number of CIoT devices constant at 220. The results of this experiment are shown in Figure 8.

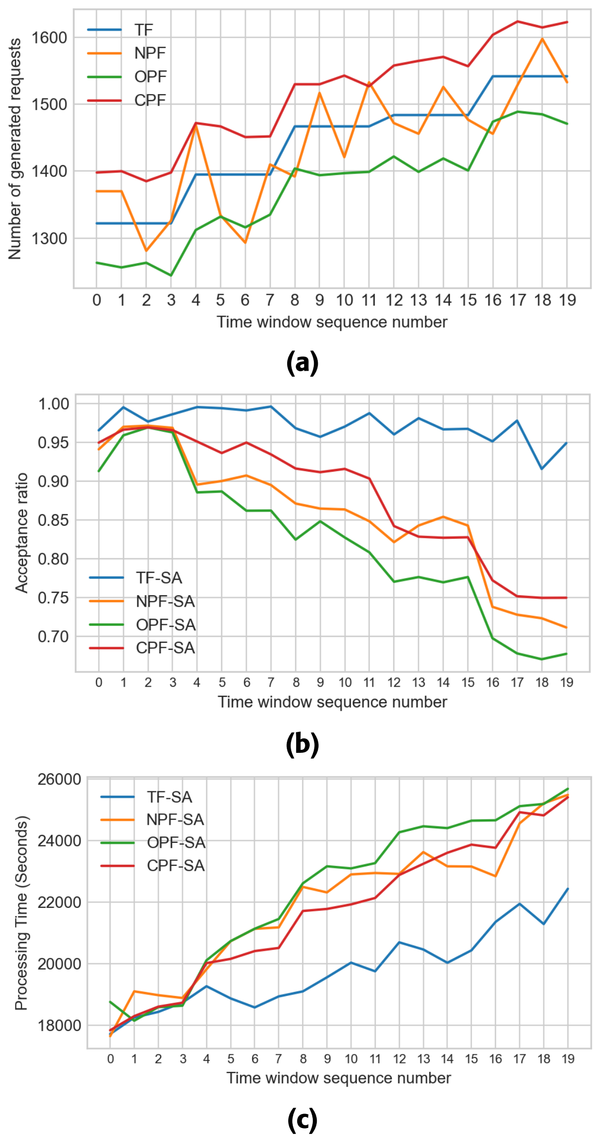


FIGURE 8. In the first Category B experiment, the frequency of service requests was increased while the number of CIoT devices was kept constant. The graphs depict (a) the number of requests that arrived in each time window, (b) the acceptance ratio, and (c) the processing time.

As seen from Figure 8, the results of the first Category B experiment are similar to those of the Category A experiments. SATS achieved the highest acceptance ratio and lowest processing time when it used the conservative predictor and the lowest acceptance ratio and longest processing time when the optimistic predictor was used. When the conservative predictor was used, the acceptance ratio for SATS only fell below 90% when the service request frequency exceeded 1500 service requests, which occurred in the 11th time window. On the other hand, when the optimistic predictor was used, SATS went below an acceptance ratio of 90% at slightly more than 1300 service requests. With the conservative predictor, the acceptance ratio for SATS was never more than 13% lower than that obtained with the true frequency. However, with the optimistic predictor, SATS sometimes exhibited an acceptance ratio 26% lower than that obtained with the true predictor. When SATS used the neutral predictor, its acceptance ratio fell between that obtained with the conservative and optimistic predictors. In fact, SATS overestimated the service request frequency 63% of the time and underestimated it 37% of the time.

In the second experiment, the number of CIoT devices was maintained at 220 while the frequency of service requests was gradually decreased. The results of this experiment are shown in Figure 9.

Similar to previous experiments, it was observed that SATS performed best when using a conservative predictor, which led to the highest acceptance ratio and shortest processing time. It is worth noting that the acceptance ratio provided by SATS with a conservative predictor was always at least 95% of that obtained with the true frequency. In contrast, using neutral or optimistic predictors resulted in SATS exhibiting a much more variable acceptance ratio.

Comparing the proposed method results with those reported in the recent work [3], it is important to note significant differences in performance. While the authors focused on scheduling a limited number of tasks, their results indicate that increasing the task arrival rate from 4 to 7 reduces the task acceptance ratio from over 50% to less than 30%. In contrast, the proposed method can efficiently manage the scheduling of more than 1000 tasks per time slot while maintaining an acceptance ratio of no less than 70% in worst case. This substantial increase in acceptance ratio, even under high load conditions, demonstrates that the proposed method is markedly more practical and effective for dynamic task scheduling in complex computing environments.

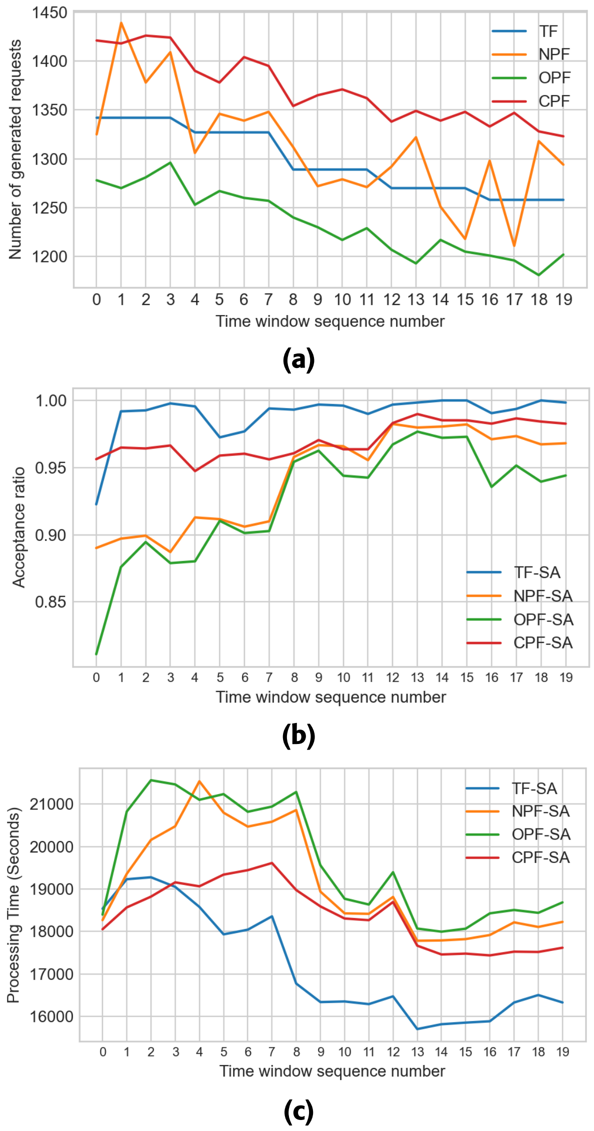


FIGURE 9. In the second Category B experiment, the frequency of service requests was decreased while the number of CIoT devices was kept constant. The graphs depict (a) the number of requests that arrived in each time window, (b) the acceptance ratio, and (c) the processing time.

VIII. RELATED WORK

Several strategies have been proposed for offloading tasks from IoT devices online. For instance, Oo and Ko [4] proposed a service-aware offloading scheme that considers the Quality of Service (QoS) requirements of IoT services. According to their scheme, low-latency tasks are executed on IoT devices, while less latency-sensitive tasks are offloaded to edge servers. Another service-aware offloading scheme, Priority Aware Task Scheduling (PaTS), was proposed by Bali et al. [5]. The PaTS scheme is designed to offload IoT sensor data on a factory floor. It categorizes the incoming tasks produced by sensor devices using a utilization function and determines whether to offload the task based on priority. In [6], Han et al. uses two-timescale optimization to reduce remote IoT task offloading delays. In this case, work is dynamically split between terrestrial and satellite networks using a hierarchical Markov decision process (H-MDP) with reinforcement learning. Rural areas with satellite and terrestrial backhaul are the target use cases for this method. These use cases are a good choice for low-latency applications. Utilizing hybrid proximal policy optimization (H-PPO), the system controls discrete and continuous action spaces across timelines. The simulation results show that the proposed design works, even with a restricted spectrum and significant traffic. The job dynamically handles incoming tasks and adapts to online network conditions. Brik et al. [7] investigate how multi-access Edge computing applications can be deployed across a federated Edge infrastructure. It addresses optimal deployment location selection to meet application-specific requirements for computing resources, latency, and service availability. They use an integer linear programming model to balance computational loads and propose a tabu search meta-heuristic algorithm to manage the placement problem efficiently. The approach dynamically adapts to network conditions and application demands, ensuring compliance with ETSI MEC standards and providing an online solution. Among these strategies, the SATS approach is noteworthy because it employs the Simulated Annealing metaheuristic optimization technique, commonly used for offline task offloading. In [8], they propose the SIaTS framework for task scheduling in computing power networks (CPNs). This approach enhances resource efficiency and service performance by integrating service intent awareness. This system utilizes intent-based networking principles to overcome the limitations of conventional task scheduling. It accomplishes this by employing an auction-based mechanism that synchronizes task scheduling with user service intentions.

Researchers have proposed several task scheduling algorithms that leverage machine learning methods such as Deep Reinforcement Learning (DRL) to address the computational challenges IoT applications pose within heterogeneous MEC networks. The goal is to optimize task execution order and location, thereby minimizing latency and enhancing service delivery efficiency.

For instance, Shang et al. [9] introduce a Deep Q-Network-based (DQN-based) task scheduling algorithm. The approach models the scheduling problem as a Markov decision process, leveraging DRL to optimize task execution order and location adaptively. Lu et al. [10] propose a task scheduling framework for container-based Edge computing environments, utilizing the Proximal Policy Optimization (PPO) algorithm to balance completion time and energy consumption. Two distinct algorithms for independent (PPOTS) and dependent (PPODTS) task scheduling were developed and tested against traditional methods. Results indicate a significant improvement in utility reduction, showcasing the potential of DRL in optimizing Edge computing operations. Min et al. [11] explore a learning-based offloading strategy for IoT devices that incorporates energy harvesting in a dynamic MEC environment. The study utilizes reinforcement learning to optimize the offloading of computational tasks to Edge devices without prior knowledge of the energy consumption or computation latency models. A DRL model is also proposed to improve learning speed and decision accuracy by managing large state spaces efficiently. In [12], Wang et al. investigate the incorporation of DRL into the scheduling of cellular networks in order to enhance the process of decision-making and flexibility. Compared to conventional techniques, it addresses how expert information can be included to improve DRL agent training, yielding a more reliable and effective result. Sellami et al. [13] present a DRL approach for energy-efficient task scheduling in SDN-based IoT networks. The proposed DRL model aims to minimize network latency and energy usage, demonstrating superior simulation performance compared to deterministic and random scheduling methods. Sheng et al. [3] propose a task scheduling framework for IoT applications utilizing a DRL technique, specifically focusing on Edge computing environments. The system utilizes a policy-based reinforcement algorithm to improve the scheduling of tasks and allocation of resources across virtual machines (VMs) on Edge servers. In [14], Tang et al. explore distributed task scheduling within serverless Edge computing networks for IoT, presenting a learning-based approach. It introduces a multiagent DRL algorithm, specifically a dueling double deep recurrent Q-network (D3RQN), to optimize task scheduling and resource allocation by considering Edge computing resources' dynamic and heterogeneous nature. The proposed method aims to enhance the efficiency and utility of Edge computing nodes by enabling them to make autonomous scheduling decisions based on local observations, demonstrating significant improvements over conventional methods through extensive simulations.

Also, several offline task-offloading strategies have been proposed for IoT devices.

Authors in [15] suggested using a Genetic algorithm as an offline task offloading strategy. Al-Habob et al. [16] also proposed a Genetic-based task offloading strategy. Authors in [17] proposed employing a Simulated Annealing algorithm to schedule tasks within a MEC architecture. Alameddine et al. [18], Alnoman et al. [19], and Lee et al. [20] have proposed offload strategies that address task offloading, application resource allocation, and task scheduling problems. These strategies ensure that the deadlines of the tasks for IoT applications are met. Some studies have focused explicitly on MEC. For instance, Ning et al. [21] have proposed an iterative heuristic resource allocation for delay-aware task offloading in a MEC architecture. Zhang et al. [22] have suggested a time-sensitive MEC task scheduling algorithm that reduces the average CIoT device processing time and saves IoT device energy consumption.

Previous studies did not consider the varying sizes of service requests or the frequency at which end devices invoke them. Unlike these works, which typically model requests as singular tasks with specific resource requirements, this research attempts to model a diverse array of requests, each with unique requirements. Furthermore, unlike the cited studies, this paper considers a substantial number of users, each frequently requesting services, something which leads to the scheduling of a large volume of different requests within each time window, necessitating complex partial scheduling across CIoT, Edge, and Cloud layers rather than merely between Edge and Cloud. Lastly, this study explores metaheuristics in online systems, evaluating their performance as faster, simpler, and resource-efficient alternatives to machine learning or reinforcement learning methods.

IX. CONCLUSION

This research paper introduces a new task scheduling strategy named SATS for MEC architecture. SATS is an online task scheduling technique that shows that Simulated Annealing is applicable beyond traditional offline task scheduling. During the evaluation of SATS, it was found that the accuracy of the service request frequency predictor played a crucial role in its performance. SATS showed excellent results when used with a conservative predictor that consistently overestimated the actual service request frequency by a small margin. This predictor could achieve an acceptance ratio comparable to knowing the frequency of service request arrivals beforehand, with a difference of no more than 5% and a consistent deviation within 20% in all experiments. However, there is a scope for improvement in MEC environments with diurnal traffic patterns, where the trends are predictable, but the individual service requests are not. In such cases, improving the service request frequency prediction may be possible by using model-based reinforcement learning. Therefore, applying reinforcement learning (RL) methods will be considered more extensively. Specifically, a method based on RL will be proposed and evaluated to see how effectively it can enhance prediction accuracy and overall system performance in these contexts.

TABLE 1. Listing of notations.

Notation	Specification
$\mathcal{M}$	Set of CIoT devices
$\mathcal{E}$	Edge server
$\mathcal{C}$	Cloud server
$Rsrc^{nodes}$	Matrix of CPU, RAM, and network resources for nodes
C_i	CPU resource of the i^{th} CIoT device
R_i	RAM resource of the i^{th} CIoT device
N_i	Network resource of the i^{th} CIoT device
$Rsrc^{links}$	Matrix of available bandwidth and delay on each link
$(\alpha, \beta)_{l_{x,y}}$	Bandwidth and delay of link between nodes x and y
$\mathcal{S}$	Set of available services
S^t	Set of sequential tasks for each service
d_{s_s}	Predefined deadline for service s
$Rqst$	Matrix of resources required for each task in a service
$C_{s_i^j}$	CPU required for task j of service i
$R_{s_i^j}$	RAM required for task j of service i
$N_{s_i^j}$	Network resources required for task j of service i
w_t	Discrete time windows
τ	Time slots within each time window
ι	Fixed time interval
f_t	Fixed frequency for each service
f_p	Modified frequency with Gaussian error
$\mathcal{N}(\mu, \sigma^2)$	Gaussian error function
x_n	Variable indicating the scheduled location for each task
$T_d^{s_s^t}$	Total delay of task t in service s
$t_q^{s_s^t}$	Waiting time for task t in service s
$t_{ex}^{s_s^t}$	processing time for task t in service s
$t_{tr}^{s_s^t}$	Transmission time for task t in service s
T_{total}	Total system delay
η	Acceptance ratio
φ	Number of requests responded within the time window
ϕ	Total number of requests at the start of the time window
Θ_χ	Processing time of the current solution
Θ_Υ	Processing time of the new solution
η_χ	Acceptance ratio of the current solution
η_Υ	Acceptance ratio of the new solution
ρ, ρ'	Probabilities for accepting new solutions
$\varsigma_{ac}, \varsigma_{exc}$	initial temperature of acceptance ratio, Processing time

APPENDIX
TABLE OF NOTATIONS

A comprehensive summary of the notations utilized throughout the paper is provided in Table 1.

REFERENCES

[1] *Edge Deployment Strategies for Communications Service providers*, Ericsson AB, Stockholm, Sweden, 2023.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 4598, doi: 10.1126/science.220.4598.671.

[3] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in IoT edge computing," *Sensors*, vol. 21, no. 5, p. 1666, Feb. 2021.

[4] T. Oo and Y.-B. Ko, "Application-aware task scheduling in heterogeneous edge cloud," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 1316–1320.

[5] M. S. Bali, K. Gupta, D. Gupta, G. Srivastava, S. Juneja, and A. Nauman, "An effective technique to schedule priority aware tasks to offload data on edge and cloud servers," *Meas., Sensors*, vol. 26, Apr. 2023, Art. no. 100670.

[6] D. Han, Q. Ye, H. Peng, W. Wu, H. Wu, W. Liao, and X. Shen, "Two-timescale learning-based task offloading for remote IoT in integrated satellite-terrestrial networks," *IEEE Internet Things J.*, vol. 10, no. 12, pp. 10131–10145, Jun. 2023.

[7] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-oriented MEC applications placement in a federated edge cloud architecture," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[8] Q. Tang, R. Xie, L. Feng, F. R. Yu, T. Chen, R. Zhang, and T. Huang, "SIaTS: A service intent-aware task scheduling framework for computing power networks," *IEEE Netw.*, early access, Oct. 20, 2024, doi: 10.1109/MNET.2023.3326239.

[9] Y. Shang, J. Li, M. Qin, and Q. Yang, "Deep reinforcement learning-based task scheduling in heterogeneous MEC networks," in *Proc. IEEE 95th Veh. Technol. Conf.*, Jun. 2022, pp. 1–6.

[10] T. Lu, F. Zeng, J. Shen, G. Chen, W. Shu, and W. Zhang, "A scheduling scheme in a container-based edge computing environment using deep reinforcement learning approach," in *Proc. 17th Int. Conf. Mobility, Sens. Netw. (MSN)*, Dec. 2021, pp. 56–65.

[11] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[12] J. Wang, C. Xu, Y. Huangfu, R. Li, Y. Ge, and J. Wang, "Deep reinforcement learning for scheduling in cellular networks," in *Proc. 11th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2019, pp. 1–6.

[13] B. Sellami, A. Hakiri, S. Ben Yahia, and P. Berthou, "Deep reinforcement learning for energy-efficient task scheduling in SDN-based IoT network," in *Proc. IEEE 19th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2020, pp. 1–4.

[14] Q. Tang, R. Xie, F. R. Yu, T. Chen, R. Zhang, T. Huang, and Y. Liu, "Distributed task scheduling in serverless edge computing networks for the Internet of Things: A learning approach," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 19634–19648, Oct. 2022.

[15] A. Mahjoubi, K.-J. Grinnemo, and J. Taheri, "EHGA: A genetic algorithm based approach for scheduling tasks on distributed edge-cloud infrastructures," in *Proc. 13th Int. Conf. Netw. Future (NoF)*, Oct. 2022, pp. 1–5.

[16] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, "Task scheduling for mobile edge computing using genetic algorithm and conflict graphs," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8805–8819, Aug. 2020.

[17] A. Mahjoubi, K.-J. Grinnemo, and J. Taheri, "An efficient simulated annealing-based task scheduling technique for task offloading in a mobile edge architecture," in *Proc. IEEE 11th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2022, pp. 159–167.

[18] 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 J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

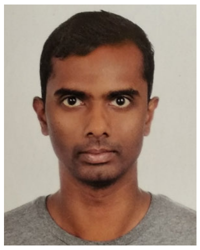
[19] A. Alnoman, "Delay-aware scheduling scheme for ubiquitous IoT applications in edge computing," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Oct. 2021, pp. 1–4.

[20] S. Lee, S. Lee, and S.-S. Lee, "Deadline-aware task scheduling for IoT applications in collaborative edge computing," *IEEE Wireless Commun. Lett.*, vol. 10, no. 10, pp. 2175–2179, Oct. 2021.

[21] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[22] J. Zhang, B. Jiang, H. Zhao, and Y. Xu, "Time-sensitive multi-user oriented mobile edge computing task scheduling algorithm," in *Proc. 2nd Int. Conf. Comput. Commun. Internet (ICCCI)*, Jun. 2020, pp. 145–149.

AYEH MAHJOUBI received the Master of Science degree in computer networking from Isfahan University, in 2017, establishing a strong foundation in the realms of computing and networking. She is currently pursuing the Ph.D. degree with Karlstad University, where she specializes in optimization, particularly focusing on task scheduling within mobile edge computing (MEC) environments. This niche field of study not only showcases her passion for technology and innovation but also places her at the forefront of cutting-edge research aimed at enhancing the efficiency and performance of computing resources in edge devices.



ARUNSELVAN RAMASWAMY received the Ph.D. degree from the Indian Institute of Science, in 2018.

He was a Lecturer with the Department of Computer Science, Paderborn University, Germany, and a Junior Research Head (Data Science and Artificial Intelligence) with the Heinz Nixdorf Institute, Paderborn University. He is currently an Associate Professor with the Department of Computer Science and Mathematics, Karlstad University, Sweden. Currently, he works on various aspects of deep reinforcement learning, stochastic optimization, dynamical systems theory, statistical learning theory, and multi-agent systems. His thesis was on stochastic approximation algorithms and reinforcement learning, which was recognized by the institute for its "outstanding research contribution." In 2022, the DLF Projekttragger awarded him the Best Young Researcher Award "Innovative Ideas in AI."

KARL-JOHAN GRINNEMO (Senior Member, IEEE) received the Ph.D. degree in computer science from Karlstad University, in 2006. With nearly 15 years of work experience, he was an Engineer in the telecom industry, first at Ericsson and later as a Consultant at Tieto. He has worked on Ericsson's mobile core and radio access network, specifically its signaling systems. From 2009 to 2010, he was an acting Associate Professor with the School of Information and Communication Technology, KTH Royal Institute of Technology, while on leave from Tieto. Since 2014, he has been a Senior Lecturer with Karlstad University. He has published over 90 papers for conferences and journals, with a focus on enhancing reliability, and throughput and reducing latency in IP networks through multipath transport protocols like multipath TCP. In addition, he has been researching energy-efficient cellular IoT networks, service placement in mobile edge networks, and telemetry in software-defined and programmable networks.