electronics

MDPI

Article

EdgeUP: Utilization and Priority-Aware Load Balancing in Edge Computing

Lan Anh Nguyen [1], Sunggon Kim [2] and Yongseok Son [1,*]

[1] Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea; loglamo@cau.ac.kr
[2] Department of Computer Science and Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; sunggonkim@seoultech.ac.kr
* Correspondence: sysganda@cau.ac.kr

Abstract: This paper introduces a dynamic context-aware load balancing mechanism called `EdgeUP`, which leverages dynamic contexts of both edge nodes and IoT devices to balance tasks on edge nodes optimally. To perform this, we devised a lightweight load balancer to allocate tasks optimally based on CPU utilization across edge nodes while satisfying the task priority of IoT devices. We implemented `EdgeUP` on real edge devices (i.e., Raspberry Pi 400) and evaluated it in terms of the average CPU utilization of edge nodes, idle CPU distance, and execution time. The experimental results show that `EdgeUP` outperforms three conventional load balancing algorithms, including round-robin, hill climbing, and K-Means, by 5.6×, 2.1×, and 3.6×, respectively.

Keywords: edge computing; edge computational model; internet of things; load balancing

check for updates

Academic Editor: Javid Taheri

Received: 16 December 2024
Revised: 25 January 2025
Accepted: 27 January 2025
Published: 30 January 2025

Citation: Nguyen, L.A.; Kim, S.; Son, Y. EdgeUP: Utilization and Priority-Aware Load Balancing in Edge Computing. *Electronics* **2025**, *14*, 565. https://doi.org/10.3390/electronics14030565

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Prominence and necessity of edge computing. A massive amount of sensor data are generated and collected in Internet of Things (IoT) everyday [1–6]. Processing the massive sensor data has become one of the most prominent challenges in IoT [7–10] since it causes high latency [11–13]. Accordingly, cloud computing and fog/edge computing have emerged to efficiently handle this challenge in IoT [1]. Edge computing is the computer power that exists on the edge of a connected ecosystem [14–20]. The computing is close to endpoint devices generating data (e.g., sensors, mobile devices, etc.). The main advantages of edge computing are the reduction in latency, improvement in security, operational cost savings, increase in reliability, and scalability. Therefore, edge computing is a proper solution for massive data processing in IoT [21]. However, edge nodes in edge computing have limitations on resources (e.g., network, memory, processing, etc.) [22]. As a result, they can be overloaded or overworked when they exceed or run out of their resources. Load balancing mechanisms, which prevent edge nodes from being overloaded, play a critical role in edge computing [23–25].

Load balancing at application layer in edge computing. Load balancing mechanisms in edge computing exhibit a diversity based on the layers of the TCP/IP model to which they are applied, such as network access, network, transport, and application layers. Numerous studies are dedicated to proposing load balancing mechanisms across the layers. These studies design load balancing mechanisms to enhance the distribution of network traffic, aiming to maximize or minimize specific network metrics [25–28], such as latency, throughput, bandwidth, etc. It is clear to observe that load balancing mechanisms mostly focus on network metrics. Giving more consideration to computing metrics is crucial,

as the computing resource is also one of the constrained resources in edge computing. Moreover, there are limited studies at the application layer compared to the three lower layers (i.e., transport, network, and network access layers). Therefore, designing load balancing mechanisms while considering computing metrics at the application layer and also conducting practical experiments are essential in edge computing.

Prior work. Numerous studies have explored load balancing strategies for IoT systems. These studies differ widely in terms of the load balancing metrics employed, the IoT layer targeted, the system layers considered, and the solution approaches employed. Table 1 presents a comparative analysis of previous works and `EdgeUP`, focusing on their load balancing metrics, i.e., static (a static context refers to an object's state that remains constant over time) and dynamic (a dynamic context refers to an object's state that can change over time) contexts, IoT layers, system layers, and algorithm methods.

Table 1. Comparison between existing studies and `EdgeUP` (NA: computing node-aware; DA: user device-aware; ML: machine learning-based approach; Opt: optimization approach; Greedy: greedy approach).

Study	Static Context	Dynamic Context	IoT Layer	System Layer	ML	Opt	Greedy
Fan et al. [25]		NA	Fog	Physical		✓	
Li et al. [29]	✓	NA	Edge	Physical		✓	
Dong et al. [27]		NA	Edge	Physical		✓	
Yu et al. [30]		NA	Cloud	Application		✓	
Li et al. [31]		NA, DA	Edge	Physical	✓		
Baek et al. [32]		NA	Fog	Physical	✓		
Esmaeili et al. [33]	✓	NA, DA	Edge	Physical	✓		
`EdgeUP`		NA, DA	Edge	Application			✓

For load balancing metrics, most recent studies on load balancing metrics focus on dynamic contexts, which reflect the changing statuses of computing nodes [25,27,29–33], as indicated by 'Computing node-aware (NA)' in the table, and users' devices [31], as indicated by 'User device-aware (DA)' in the table. In terms of IoT layers, load balancing mechanisms can be implemented at three levels of IoT: cloud [30], fog [25,32], and edge [27,29,31,33]. In terms of system layers, most recent load balancing mechanisms are deployed at the physical layer [25,27,29,31–33]. Load balancing mechanisms implemented at the physical layer rely on physical data, such as precise locations, data rates, transmission power, and signal-to-noise ratios, making the deployment more complex. To address challenges associated with physical layer deployment, the study [30] proposed shifting load balancing mechanisms to the application layer, offering a simpler and faster implementation. Regarding approaches of load balancing mechanisms, load balancing mechanisms often employ three popular approaches: machine learning-based [31–33], optimization-based [25,27,29,30], and greedy approaches. Both machine learning-based and optimization-based approaches can provide optimal load balancing solutions. However, they are computationally intensive and time-consuming. In contrast, the greedy approach ensures computational efficiency.

Our study, `EdgeUP`, focuses on load balancing in edge computing, specifically considering the edge layer in IoT. Recognizing the impact of dynamic contexts on load balancing methods, we support both types of dynamic contexts in IoT: computing node and user device contexts. Furthermore, our system is designed at the application layer for more efficiency and flexibility than the physical layer. Lastly, we employ a greedy approach to provide efficient computation on resource-constrained devices.

To perform this, `EdgeUP` employs a novel load balancing metric, `ServingValue`, as an objective computing metric to allocate tasks to appropriate edge nodes. To handle the dynamic nature of an edge computing environment, `ServingValue` considers two key dynamic contexts: the dynamic contexts of IoT devices and edge nodes. The proposed load balancing mechanism operates as a module at the application layer of the edge layer in

IoT, making optimal decisions regarding task assignments to edge nodes to maximize the `ServingValue` metric. Our study offers the following contributions:

- First, we introduce a novel load balancing metric, `ServingValue`, which incorporates two dynamic contexts in edge computing: IoT device context (i.e., task priority) and edge node context (i.e., CPU utilization). Our goal is to maximize this metric by optimizing task assignments (i.e., pairs of tasks and corresponding edge nodes).
- Second, we employ a lightweight greedy approach, `EdgeUP`, to address the task assignment problem, making it suitable for edge nodes with limited resources.
- Third, we provide a practical implementation of a load balancing module at the application layer.
- Finally, our proposal outperforms three well-known load balancing methods, including round-robin, hill climbing, and K-Means.

2. Related Works

`EdgeUP` introduces a novel load balancing metric (i.e., `ServingValue`), which takes into account dynamic contexts in edge computing, such as IoT device and edge node contexts. Not only does it provide a dynamic context-aware load balancing mechanism using the proposed metric, but `EdgeUP` also guarantees a lightweight execution, making it suitable for edge computing by utilizing a greedy load balancing strategy.

Fan et al. [25] proposed a load balancing method at the fog layer in IoT, using CPU capacity, one of the dynamic contexts of computing nodes, as a key metric in their optimization-based mechanism to reduce user response latency. Dong et al. [27] took into account the dynamic contexts of computing nodes at the edge layer in IoT, specifically memory and CPU utilization, to assign tasks to either edge or cloud nodes. Their optimization-based load balancing mechanism aims to reduce task completion time. Li et al. [29] utilized both static and dynamic load balancing metrics. The static metrics include physical memory size, CPU frequency, number of cores, and disk size, while the dynamic metrics involve memory and CPU utilization. Using both types of contexts, their optimization-based load balancing mechanism in edge computing classifies edge nodes into different states (i.e., light, normal, or heavy) and assigns tasks to the most suitable nodes to reduce task completion time. Our study, `EdgeUP`, aligns with previous studies [25,27,29] in considering dynamic contexts of computing nodes (e.g., memory and CPU utilization) for load balancing. Additionally, `EdgeUP` shares similarities with [27,29] in addressing load balancing at the edge layer in IoT. However, while the load balancing mechanisms in [25,27,29] are deployed at the physical layer, `EdgeUP` is deployed at the application layer. Moreover, in addition to considering dynamic contexts of computing nodes like these studies, `EdgeUP` also takes into account another type of dynamic context, a dynamic context of user devices (i.e., user requests). Lastly, whereas these studies use optimization-based approaches for load balancing, `EdgeUP` employs a greedy approach to ensure lightweight execution of the mechanism.

Li et al. [31] incorporated both types of dynamic contexts for load balancing in edge computing, including the dynamic context of computing nodes (i.e., computing power of edge nodes) and dynamic context of user devices (i.e., task demands). Their approach aimed to enhance resource utilization while minimizing computational latency for tasks. Baek et al. [32] used both static and dynamic contexts for load balancing in fog computing. They considered CPU frequency as the static context and the number of remaining tasks in queues of computing nodes as the dynamic context to reduce task response latency. Esmaeili et al. [33] considered both static and dynamic contexts for load balancing in edge computing. They used memory and disk capacity of edge nodes as static contexts, and CPU utilization of edge nodes and task demands as dynamic contexts (which are

dynamic contexts of computing nodes and user devices, respectively). Their approach aimed to reduce user response latency. `EdgeUP` aligns with [31–33] by considering the dynamic contexts of computing nodes, such as memory and CPU utilization of edge nodes. Furthermore, `EdgeUP` is in line with [31,33] by addressing both types of dynamic contexts (i.e., computing node and user device contexts) and focusing on load balancing at the edge layer in IoT. In contrast to these studies that are deployed at the physical layer, `EdgeUP` operates at the application layer. Additionally, `EdgeUP` introduces a novel load balancing metric, `ServingValue`, which integrates both types of dynamic contexts (i.e., computing node and user device contexts), to efficiently enhance CPU utilization across all computing nodes. Moreover, unlike these studies that depend on machine learning-based methods for solving load balancing problems, `EdgeUP` employs a greedy approach to maintain the lightweight execution.

Yu et al. [30] addressed load balancing in microservice applications deployed in the cloud by incorporating dynamic contexts. Their study focuses on the dynamic contexts of computing nodes, particularly the external and internal demands within the microservice nodes. Our study, `EdgeUP`, aligns with this study in considering dynamic contexts of computing nodes and deploying the load balancing mechanism at the application layer. In contrast, our study tackles the load balancing problem at the edge layer in IoT, while Yu et al.'s study [30] concentrated on the cloud layer in IoT. Additionally, regarding load balancing metrics, while Yu et al. [30] focused only on the dynamic contexts of computing nodes, `EdgeUP` introduces a novel load balancing metric by combining the dynamic contexts of both computing nodes and user devices. Furthermore, while Yu et al. [30] proposed an optimization-based approach for their load balancing problem, our study employed a greedy approach to ensure the mechanism's lightweight execution.

3. Background and Motivation

3.1. Load Balancing-Aware Edge Computing in IoT

In IoT, the network architecture is typically divided into three layers, such as cloud layer, fog layer, and edge/device layer [34–36], based on their computing, storage capabilities, and their distance to end-users. The load balancing challenge in edge computing also pertains to the three-layer network architecture [28,29]. Figure 1 presents load balancing-aware edge computing in IoT based on the three-layer network architecture. In this study, we focus on load balancing within edge computing. As shown in Figure 1, we emphasize the edge/device layer in the three-layer network architecture by specifying it, while merging the fog and cloud layers into single upper layers.

Load balancing-aware edge computing in IoT is structured into three main layers:

- **Device layer:** This layer comprises IoT devices such as mobile phones, sensor nodes, autonomous cars, smart traffic lights, and robots. In the load balancing-aware edge computing architecture, we consider scenarios where IoT devices request services from the upper layers. For example, they can request tasks, such as data processing, data caching/storage, and IoT management, from the edge layer.
- **Edge layer:** This layer consists of a cluster of edge nodes, which are capable of computing and storage, and are located close to IoT devices. Edge nodes in the cluster are classified into two types: one master node (i.e., controller node) and multiple edge nodes (i.e., worker nodes). In this architecture, the master node works as a load balancer, collecting task requests from IoT devices, as well as information (e.g., data, status, etc.) about IoT devices and edge nodes. It then executes load balancing strategies to assign the IoT devices' task requests to the appropriate edge nodes. Edge nodes report their information (e.g., status (ACTIVE / NONACTIVE), memory, and CPU utilization, etc.) to the load balancer, receive task assignments from the load

balancer, and execute the assigned tasks. As depicted in Figure 1, the red arrows indicate the task assignments progressing from the load balancer to edge nodes. Dotted lines present task offloading from edge nodes to others if they are incapable of performing the assigned tasks. In the case of any significant tasks that cannot be processed at the edge layer, they are offloaded to the upper layer (i.e., fog/cloud layer).

- **Fog and Cloud layers (upper computing layer):** This layer consists of machines with higher computing and storage capabilities than the lower layer (i.e., edge layer). If the significant tasks cannot be processed at the edge layer, they will be offloaded to and executed at this layer.

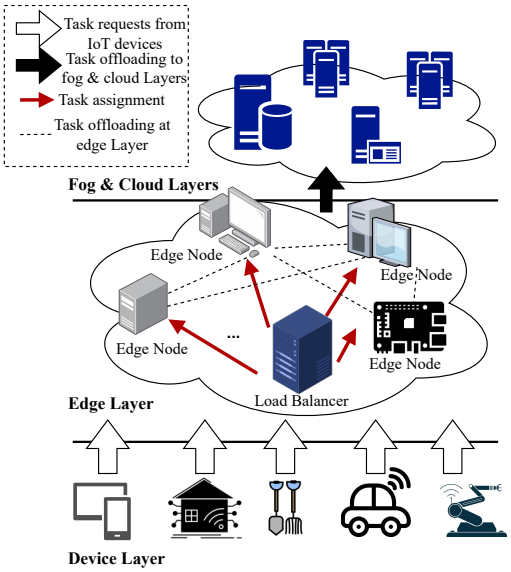


Figure 1. Load balancing-aware edge computing in IoT.

3.2. Load Balancing Methods in Edge Computing

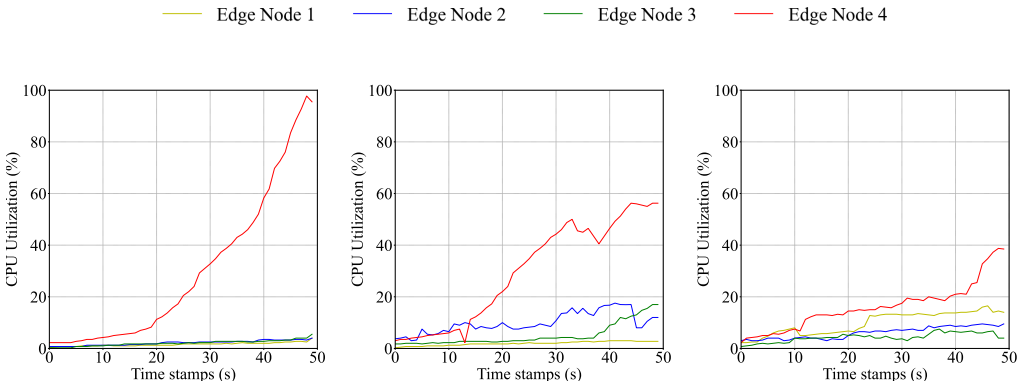
The concept of load balancing in edge computing is similar to that in fog and cloud computing. Since computing nodes in edge computing clusters contend with resource constraints (e.g., network, memory, processing, etc.) [23,25,37], load balancing mechanisms play a pivotal role in preventing computing nodes from becoming overloaded.

Load balancing in edge computing employs various metrics, such as resource utilization, processing time, throughput, availability, security, energy consumption, cost, and response time [23,25,38]. It is obvious that conventional load balancing metrics mostly focus on the dynamic contexts of computing nodes (e.g., CPU utilization, memory usage, etc.), while omitting or not directly addressing the dynamic contexts of IoT devices, such as the volume of task requests and their computational requirements. The volume of task requests from IoT devices fluctuates over time. Additionally, the computational requirements of these task requests can vary significantly. These contexts of IoT devices directly impact the operation of computing nodes. For instance, the types of task requests from IoT devices can indicate how tasks affect the operation of computing nodes during or after their execution.

To underscore the limitation of conventional load balancing metrics in handling the dynamic context of IoT devices, we conducted an experiment using three load balancing mechanisms that only consider computing node context (e.g., CPU utilization), as shown in Figure 2.

In the experiment, the dynamic context of IoT devices is represented by three types of tasks (i.e., reading, writing, and model training) generated randomly. Considering only computing node context (i.e., CPU utilization), load balancing mechanisms mostly fail to prevent edge nodes from becoming overloaded. As shown in Figure 2a, the CPU

utilization of Edge Node 4 reaches approximately 100% with the round-robin load balancing mechanism. As shown in Figure 2b,c, even when employing K-Means and hill climbing, significant gaps in CPU utilization among edge nodes persist, making it easy to push some edge nodes into an overloaded state.



(**a**) Round-robin (**b**) K-Means (**c**) Hill climbing

Figure 2. Performance of load balancing mechanisms on a conventional load balancing metric (i.e., CPU utilization).

In conclusion, a novel load balancing metric, which is aware of both the dynamic contexts of computing nodes and IoT devices, should be efficiently proposed for load balancing mechanisms in edge computing. To address this challenge, this study introduces `EdgeUP`, which takes into account the dynamic contexts of both computing nodes and IoT devices to make decisions of task assignments while ensuring its lightweight execution for resource-constrained devices in edge computing.

4. EdgeUP: Utilization and Priority-Aware Load Balancing in Edge Computing

4.1. Overview of EdgeUP

Our load balancing mechanism aims to improve task assignment within edge computing. Our load balancing mechanism, `EdgeUP`, addresses the dynamic change in edge computing by taking into account the dynamic contexts of both IoT devices and edge nodes at the application layer. Regarding IoT devices, we consider priorities of tasks requested by users for execution on edge nodes. Regarding edge nodes, we consider their computational resource: CPU utilization. The product of these factors (i.e., priorities of tasks and CPU utilization) is introduced as a new metric called `ServingValue`. Eventually, our load balancing algorithm identifies the optimal (task, edge node) pairs to maximize the total `ServingValue`. By maximizing `ServingValue` with each task assignment, `EdgeUP` balances CPU utilization across edge nodes, preventing them from becoming overloaded. `EdgeUP` directly emphasizes the task assignments rather than offloading tasks between edge nodes.

Figure 3 provides an overview of our load balancing system at the application layer. The system consists of four primary components:

- **IoT devices:** Send task requests to "Task" topic on the message broker.
- **Message broker:** A module that facilitates communication among components in the system, allowing them to exchange information through message queue topics. It includes predefined topics containing message queues. The "Task" topic holds messages about task requests from IoT devices. The "Utilization" topic stores messages about CPU utilization, which are reported by the edge nodes. The "Task Assignment" topic contains messages about the assignment results, specifically the (task, edge node) pairs determined by the load balancing algorithm.

- **Balancer:** Receives task requests and CPU utilization from the corresponding topics on the message broker, executes the load balancing algorithm (i.e., `EdgeUP`), and sends the assignment results to the "Task Assignment" topic on the message broker.
- **Edge nodes:** Receive task assignments from the "Task Assignment" topic on the message broker and execute the assigned tasks. Additionally, edge nodes periodically report their CPU utilization to the "Utilization" topic on the message broker.

In the following part, we formulate the load balancing problem in `EdgeUP`.

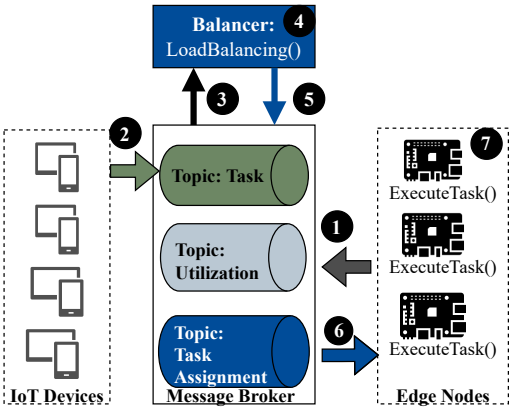


Figure 3. System model.

4.2. Problem Formulation

The system comprises a cluster $\mathcal{E}$ of E edge nodes with computational capabilities, $\mathcal{E} = \{1, 2, \ldots, E\}$, a load balancer $\mathcal{B}$, and a set $\mathcal{N}$ of N IoT devices, $\mathcal{N} = \{1, 2, \ldots, N\}$. Communication among IoT devices, edge nodes, and the balancer is facilitated through a message broker $\mathcal{Q}$.

Each IoT device n at time t has the following:

- A binary variable $v_n^t \in \{0, 1\}$, which denotes whether the IoT device n requests a task at time t or not.
- Priority value $V_n^t \in [0, 1]$ represents the importance or criticality of a task requested by IoT device n at time t. The priority values should be predefined, with higher values indicating tasks of greater importance and requiring more computational resources for execution.

Each edge node e at time t has the following:

- A_e^t, called "idle CPU" of edge node e at time t, represents a normalized value in the range $[0, 1]$ of idle CPU percentage (i.e., idle CPU percentage is the complement of CPU utilization percentage, idle CPU = 100% - CPU utilization).
- A binary variable $a_e^t \in \{0, 1\}$, which denotes whether edge node e at time t is assigned to a task or not.

As shown in Figure 3, at step (1), edge nodes periodically perform their CPU utilization by reporting to the "Utilization" topic on the message broker. At step (2), IoT devices send task requests (e.g., reading, writing files, training models, etc.) to the message broker. Task requests from IoT devices are sent to the corresponding topic on the message broker, "Task". Each time, there are task request messages from IoT devices; the balancer will receive them by subscribing the "Task" topic on the message broker as shown at step (3). It also obtains the latest CPU utilization message from the "Utilization" topic as shown at step (3). After consuming task requests and CPU utilization messages, the balancer performs its load balancing algorithm as shown at step (4). The results of the load balancing algorithm from (4) are sent to the "Task Assignment" topic on the message broker as shown at step (5).

Edge nodes always subscribe the "Task Assignment" topic, so every time there is a new message (i.e., task assignment message from the balancer), edge nodes will receive (6) and execute the task assigned to them as shown at step (7).

Particularly, tasks from IoT devices $\mathcal{N}$ are pushed into the "Task" topic. CPU utilization from edge nodes $\mathcal{E}$ are pushed into the "Utilization" topic. Balancer $\mathcal{B}$ uses two this information for the load balancing algorithm to output task assignments. As the result, we obtain pairs of (task, edge node), (v_n^t, a_e^t), at each time t. We denote a new load balancing metric as `ServingValue`. It indicates the relationship between a pair comprising a task and an edge node.

The serving value at time t of the pair (i.e., task n and edge node e, (v_n^t, a_e^t)) is

$$S(v_n^t, a_e^t) = a_e^t A_e^t v_n^t V_n^t \tag{1}$$

The total serving value at time t is

$$S^t = \sum_{\mathcal{E}} \sum_{\mathcal{N}} S(v_n^t, a_e^t) = \sum_{\mathcal{E}} \sum_{\mathcal{N}} a_e^t A_e^t v_n^t V_n^t \tag{2}$$

With a constraint of serving limitation of each edge node at each time,

$$\sum_{\mathcal{E}} \sum_{\mathcal{N}} a_e^t \leq E \tag{3}$$

The objective of this study was to maximize the total serving value. This objective helps to obtain optimal task assignments to prevent edge nodes from being overloaded.

The problem formulation is

$$\max \sum_{\mathcal{E}} \sum_{\mathcal{N}} a_e^t A_e^t v_n^t V_n^t \tag{4}$$

$$st. \sum_{\mathcal{E}} \sum_{\mathcal{N}} a_e^t \leq E \tag{5}$$

To maximize the total serving value at each timestamp, it is necessary to determine the optimal task and edge node pairs at each moment. Due to the limitation on the computation of edge nodes, we employ a lightweight approach to solve (4). The `EdgeUP` algorithm is presented in Algorithm 1.

Firstly, sets of edge nodes, IoT devices, and task priority values are set up (line 1). Three empty arrays are initialized for edge node indexes after an idle CPU value-sorting step (i.e., A), for task indexes after a priority value-sorting step (i.e., V), and for task assignment results (i.e., P), respectively (line 2). In a running process of T requesting times, at time t, the balancer collects idle CPU values of E edge nodes and priority values of N IoT devices (line 4 and line 5). Next, these values of idle CPU and task priority are sorted in ascending order into the A and V arrays, respectively (line 6 and line 7). Items with the same indexes in both sorted arrays from the previous step are paired together, forming pairs (i.e., P) of edge node index and task index (line 9). These pairs are the task assignments at the tth requesting time to edge nodes, $P^t = \{e, n\}^N$ (line 11). It means that the edge node e is assigned to perform the task n for each pair of P^t. The assignment is produced to the corresponding topic (i.e., "Task Assignment" topic) on the message broker (line 11). Consequently, all edge nodes can receive task assignment messages. Next, the three temporary arrays A, V, P are emptied (line 12), and the process continues with the next request time (line 13).

The total serving value at each timestamp is maximized in Algorithm 1 by applying the rearrangement inequality, which is detailed in Appendix A. The maximization problem in Equation (4) involves two key components: idle CPU values of edge nodes (represented

as array A in Algorithm 1) and task priority values of IoT devices (represented as array V in Algorithm 1). According to the rearrangement inequality, the highest serving value at each timestamp is achieved by pairing the largest elements of A with the largest elements of V. Thus, sorting both arrays A and V in ascending order and pairing elements with the same indices ensure the maximization described in Equation (4).

Algorithm 1 EdgeUP: Utilization and priority-aware load balancing in edge computing.

Initialize $\mathcal{E}$(Set of edge nodes.) $= \{1, \ldots, E\}$, $\mathcal{N}$(Set of IoT devices.) $= \{1, \ldots, N\}$, and $\{V_n$(Task priority value of IoT device n.)$\}^N$
Initialize A (Array of edge node indexes after sorting idle CPU of edge nodes in ascending order.) = [$empty$], V (Array of task indexes after sorting priority of tasks requested from IoT devices in ascending order.) = [$empty$], P (Array of (e, n) pairs; The variables e and n represent the indexes of the edge node and IoT device, respectively.) = [$empty$]
for $t = 0$ to $(T - 1)$ **do**
    Collect set of idle CPU of edge nodes $\{A_e^t\}^E$
    Collect set of priority values of tasks $\{V_n^t\}^N$
    A^t = Sort($\{A_e^t\}^E$) in ascending order
    V^t = Sort($\{V_n^t\}^N$) in ascending order
    for $i = 1$ to N **do**
        $P^t[i] = [e$ from $A^t[i], n$ from $V^t[i]]$
    end for
    Produce task assignments at t with $P^t = \{(e, n)\}^N$
    $A^{t+1} = [empty], V^{t+1} = [empty], P^{t+1} = [empty]$
    $t++$
end for

4.3. `EdgeUP` Procedure

The `EdgeUP` procedure is presented in Figure 4. The three primary stages of the `EdgeUP` procedure, derived from the seven steps outlined in Figure 3, are illustrated in Figure 4. Each stage reflects the status of the three key components of the system (i.e., edge nodes, message broker, and balancer). The numbered sequence from Figure 3 is preserved, demonstrating the procedural flow of `EdgeUP`. The orange color is used to show the active states of functions and messages in each stage.

- At stage (1, 2, 3), edge nodes report their utilization to the "Utilization" topic on the message broker (1). Simultaneously, IoT devices send task requests to the "Task" topic on the message broker, represented by (2). Subsequently, the balancer retrieves task requests and utilization reports from the "Task" and "Utilization" topics on the message broker, as shown at (3.1) and (3.2), respectively. The retrieved data serve as input for the load balancing function, illustrated at (3.3).
- At stage (4, 5, 6), the balancer uses the results from the previous step as inputs and executes the LoadBalancing function to determine task assignments, as indicated by (4). The task assignments, represented as pairs of (edge node index, task index), are processed by SendAssignment in preparation for delivery to the message broker, shown at (5.1). For example, in Figure 4, at stage (4, 5, 6), task assignments are EN_1:1, …, EN_E:N, meaning that the edge node 1 is assigned to task 1 and the edge node E is assigned to task N. Subsequently, these assignments are sent to the "Task Assignment" topic by the SendAssignment function, as illustrated at (5.2). Finally, the ReadAssignment functions at the edge nodes retrieve the assignments by subscribing to the topic, as shown at (6).
- At stage (7), edge nodes execute tasks assigned to them, as depicted at (7). For example, EdgeNode1 carries out Task1, while EdgeNodeE executes TaskN. During this stage, edge

nodes continue to report their utilization to the "Utilization" topic on the message broker at regular intervals, as indicated by (1). Simultaneously, task requests are continuously generated, represented by (2). The balancer retrieves task requests and utilization reports through the ReadTaskRequest and ReadUtilizationReport functions, which read from the relevant topics on the message broker, as shown at (3.1) and (3.2), respectively.

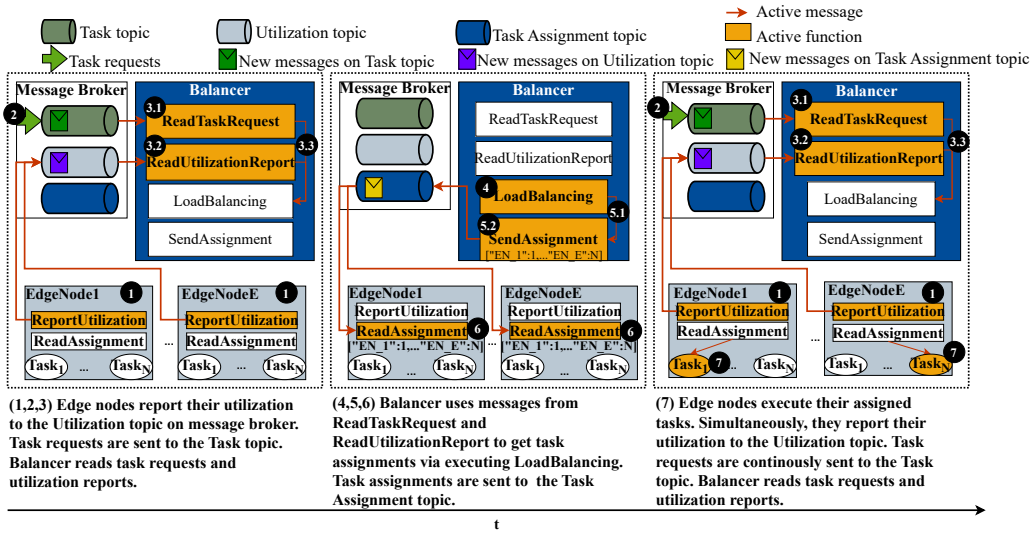


Figure 4. EdgeUP procedure.

4.4. Task Generator

A task generator is a critical component in computing frameworks and simulators that samples events/tasks in an interval of time. The generating process is under the control of probability models, which can be Gaussian, Poisson, Dirichlet probability models, etc. Tasks in cloud, fog, and edge computing experiments have been popularly simulated as Poisson-based events or Poisson processes [39–41]. The usage of the Poisson process for events/task generators is in cloud, fog, and edge simulators, such as Cloudsim [42], CloudSimPy [43], EdgeCloudSim [44], etc. A Poisson process is defined as follows.

Suppose that events/tasks occur at a fixed rate $\lambda > 0$ throughout an interval. The interval can be divided into sub-intervals t, expressed as $t \in [0, \infty)$. The counting process, resulting in the number of events/tasks at the sub-intervals, is called a Poisson process and denoted as $\{N(t), t \in [0, \infty)\}$. This process must satisfy all three conditions:

1. $P(0) = 0$: The probability of events/tasks occurring at the first sub-interval is zero;
2. An event/task occurring in any given sub-interval is independent of other sub-intervals;
3. $P(t, \tau) \propto Poisson(\lambda\tau)$: The probability of one event occurring in a sub-interval is proportional to the length τ of the sub-interval.

The word "process" is used to suggest that the simulation takes place over time. The random variable X representing the actual number of events/tasks occurring in the interval has a Poisson distribution. According to the Poisson model, the probability of r occurrences in the interval is given as follows:

$$P(X = r) = e^{-\lambda} \frac{\lambda^r}{r!}, r = 0, 1, 2, 3, \ldots \quad (6)$$

This research adopts methods in simulation from previous works in cloud, fog, and edge computing, so the Poisson model is used to sample events/tasks in the experiment.

5. Experiments and Results

5.1. Experimental Setup

In our experiment, we set up a cluster of 5 edge computing nodes with 1 master node working as a balancer and 4 edge nodes working as workers. All 5 edge computing nodes used in our experiment were Raspberry Pi 400. Configuration and installation of the edge computing nodes are shown in Table 2.

Table 2. Configuration and installation.

	Master Node	**Edge Node**
Quantity	1	4
Configuration	Raspbian OS 4 CPUs	Raspbian OS 4 CPUs
Tools	Apache Kafka	Apache Kafka, Python Threading, TensorFlow Keras
Modules	Kafka broker, Kafka producer, Kafka consumer, LB mechanisms, Task generator	Kafka producer, Kafka consumer, Tasks (reading, writing, MLP training)

We simulated an edge computing system by adhering to a widely recognized reference architecture for edge computing [45,46]. This architecture specifies that both storage and intelligence operations are conducted at the edge layer. Key operations include reading and writing for storage and machine learning model training for intelligence. To reflect this, we simulated three types of tasks: reading, writing, and machine learning model training. Specifically, reading and writing tasks involve accessing and modifying rows within a time-series dataset, a common format in IoT applications. For our experiments, we utilized SensorScope datasets [47], which contain time-series data rows comprising timestamp, temperature, and light with the data types string, float, and float.

Each reading task retrieves a single data row, while each writing task appends a new data row to the file. For the model training task, we trained a basic neural network, specifically a multi-layer perceptron (MLP). The training configuration included an MLP with a (12, 8) structure (e.g., 12 input features fed into the network; the network with 1 hidden layer with 8 neurons; the output layer with Softmax activation function for classification), 1 training round, the Adam optimizer, 500 epochs, and a batch size of 10. Approximately 2800 KB of time-series data from the SensorScope dataset [47] were used as the training dataset.

The task generator in our simulation is responsible for generating task requests at regular intervals. The rationale behind incorporating a task generator is detailed in Section 4.4. To simulate task requests originating from IoT devices, we employed a Poisson-based request generator at the load balancer, ensuring realistic task generation patterns.

We used Apache Kafka as the message broker. Furthermore, at the edge nodes, we utilized the Threading module in Python to provide them with asynchronous task running. Since tasks assigned to each edge node occur randomly and have lengthy execution times, each edge node was capable of running tasks asynchronously. Additionally, edge nodes were required to perform tasks such as reading, writing, and multi-layer perceptron (MLP) training. For the MLP training task, the edge nodes installed TensorFlow Keras to create and perform the training model. To conclude, 5 functional modules were installed on the master node, while 3 functional modules were installed on edge nodes as shown in Table 2.

The master node runs 5 modules:

- Kafka broker: Works as a message broker in Figure 3.
- Task generator: Generates 5 tasks of 3 task types (i.e., reading, writing, and MLP training) within each interval based on the Poisson process at the rate $\lambda = 0.5$.
- Balancer: Performs 4 load balancing mechanisms (i.e., `EdgeUP`, round-robin, hill climbing, K-Means). Priorities of tasks are predefined (e.g., $V_{reading} = 0.2, V_{writing} = 0.2, V_{training} = 0.6$).
- Kafka producer: Produces messages to their topics on the Kafka broker. In particular, after obtaining tasks from the task generator, the Kafka producer on the master node produces tasks to the "Task" topic as shown in Figure 3. The balancer obtains messages from the "Task" topic as one of two inputs. Additionally, after the load balancing mechanisms are executed and the task assignment results are provided, the Kafka producer produces those task assignments to the "Task assignment" topic.
- Kafka consumer: The master node can also work as a message consumer. With this role, it consumes messages on the "Utilization" topic as the other input of the balancer.

Edge nodes run 3 main modules:

- Task: Executes tasks (i.e., reading, writing, and MLP training).
- Kafka producer: Periodically conducts CPU utilization reporting by gathering CPU utilization values and subsequently sending them to the "Utilization" topic on the Kafka broker. In experiments, we set up various reporting periods of edge nodes, such as $p = 5, p = 15, p = 30, p = 50$ in seconds.
- Kafka consumer: Consumes the "Task assignment" topic on the Kafka broker to identify the tasks assigned to edge nodes by the master node.

In our experiments, we focused on evaluating how well our proposal could improve idle CPU distance and the CPU utilization of edge nodes across various reporting periods of edge nodes and also be compared to 3 load balancing mechanisms with a conventional load balancing metric (i.e., CPU utilization). Additionally, we compared the execution time of our proposal to that of other load balancing mechanisms.

5.2. Results and Analysis

5.2.1. Impact of Reporting Period on `ServingValue`

First, we examined how the reporting periods of edge nodes affect average serving values (i.e., proposed load balancing metric) in our proposal. Figure 5 shows how the reporting period affects `ServingValue`.

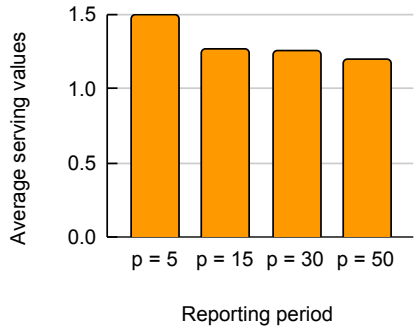


Figure 5. Average serving value of `EdgeUP` on various reporting intervals (p: reporting period in seconds).

`ServingValue` reflects the effectiveness of a load balancing method in our study. A higher `ServingValue` indicates better performance of the load balancing method. Figure 5 presents the average `ServingValue` across different reporting periods. As shown, `EdgeUP`

achieves the highest `ServingValue` at $p = 5$. This suggests that shorter reporting periods lead to improved `ServingValue`.

This can be explained by the fact that shorter reporting periods provide more up-to-date information on the CPU utilization of edge nodes within load balancing mechanisms. In other words, load balancing mechanisms can leverage the latest CPU utilization of edge nodes for their operations, being aware of the dynamic status of CPU utilization. Conversely, a prolonged reporting period can degrade the performance of load balancing mechanisms, as it prevents the accurate observation of the dynamic CPU utilization status of edge nodes. Consequently, for subsequent experiments, $p = 5$ is employed as the reporting period setup.

5.2.2. Comparison of CPU Utilization

The primary focus of this study is to introduce an effective load balancing mechanism that minimizes CPU utilization discrepancies among edge nodes. This is achieved by maximizing the proposed load balancing metric, `ServingValue`, thereby assisting edge nodes in avoiding overloaded states.

Figure 6 shows the CPU utilization of edge nodes over 50 rounds of task generation across different load balancing mechanisms (i.e., `EdgeUP`, round-robin, hill climbing, K-Means) with $p = 5$. The figure demonstrates that `EdgeUP` successfully maintains a high level of similarity in CPU utilization among edge nodes within a computing cluster. In contrast, the round-robin method results in the worst similarity in CPU utilization (Figure 6a). With K-Means and hill climbing, the similarity in CPU utilization is more improved than with the round-robin. However, it still raises up to over 30%, 15% of CPU utilization gaps between 2 edge nodes with K-Means (Figure 6b) and hill climbing (Figure 6c), respectively, whereas, as shown in Figure 6d, `EdgeUP` limits CPU utilization gaps to no more than 10%.

The large disparities in CPU utilization among edge nodes when using the three conventional methods stem from their failure to account for the current status of edge nodes (i.e., CPU utilization) and the nature of incoming task requests (i.e., reading, writing, and MLP training). For instance, in our experiments, MLP training demands the highest CPU utilization compared to the other two task types. In the worst-case scenario, an edge node already engaged in an MLP training task is assigned an additional MLP training task, causing its CPU utilization to rise further. This scenario frequently occurs with the round-robin method, as it assigns tasks sequentially without considering the edge nodes' current workload or the type of task. As shown in Figure 6a, Edge Node 4 exhibits an increasing CPU utilization, nearing an overloaded state. This is because Edge Node 4, already handling an MLP training task with high CPU demand, continues to receive similar tasks, compounding its CPU load.

Additionally, `EdgeUP` can consistently maintain the similarity in CPU utilization among edge nodes. The higher the similarity in CPU utilization, the more effective the load balancing mechanism. This is because a high similarity in utilization among edge nodes is equal to minimize CPU utilization discrepancies by maximizing `ServingValue`, reducing the likelihood of edge nodes becoming overloaded. `ServingValue` is a combination of idle CPU (i.e., the complementary part of CPU utilization, idle CPU = 100% − CPU utilization) and task priority. `EdgeUP` tries to assign tasks with high priority to edge nodes with high idle CPU. Because tasks with high priority will assume high utilization, they should be assigned to edge nodes with high idle CPU. As the result, the CPU utilization values of edge nodes converge towards each other. In other words, whenever an edge node has a low idle CPU, `EdgeUP` strives to balance the load by assigning heavier tasks to edge nodes with higher idle CPU or by avoiding assigning heavy tasks to those with low idle CPU.

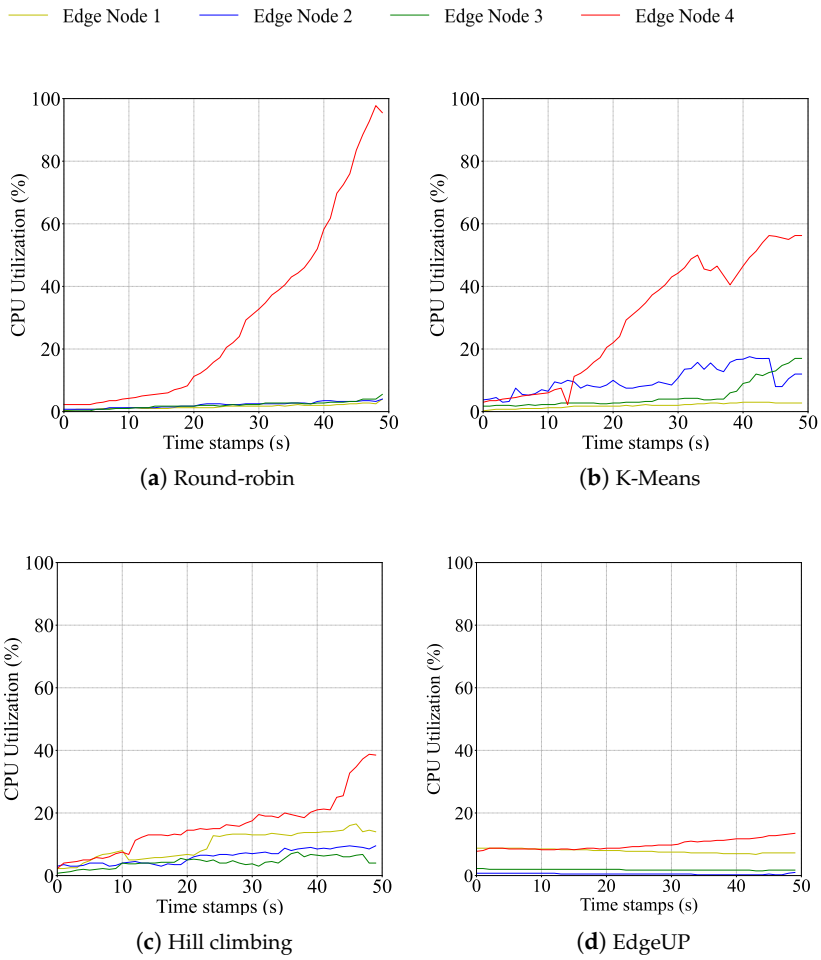


Figure 6. CPU utilization of edge nodes across different load balancing mechanisms, $p = 5$.

To better discern the similarity in CPU utilization among edge nodes, we provide a comparison of average idle CPU distance among edge nodes over 50 rounds of task generation across multiple load balancing mechanisms in Figure 7.

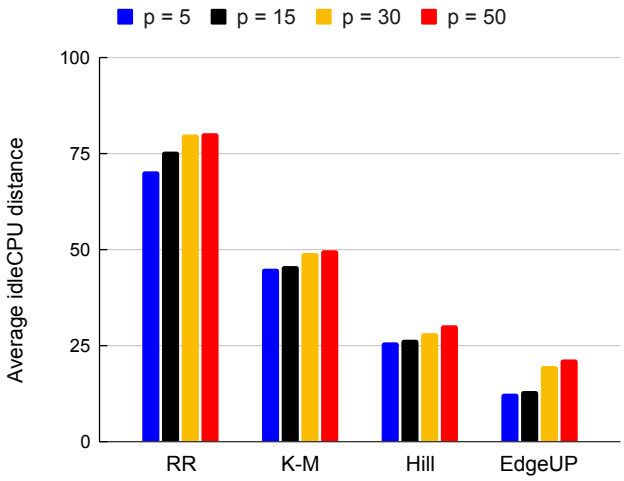


Figure 7. Average idle CPU distance of load balancing mechanisms on various reporting periods (p: reporting period in seconds; RR: round-robin; K-M: K-Means; Hill: hill climbing).

The idle CPU distance among edge nodes is calculated by

$$d(\mathcal{E}) = \sum d(e_i, e_j), \tag{7}$$

with

$$e_i, e_j \in \mathcal{E} \tag{8}$$

$$i < j \tag{9}$$

$$d(e_i, e_j) = |idleCPU_{e_i} - idleCPU_{e_j}| \tag{10}$$

where $\mathcal{E}$ is the set of edge nodes. e_i, e_j are edge node i and edge node j.

The idle CPU distance between 2 edge nodes is calculated as Equation (10). The lower the idle CPU distance, the higher the similarity of CPU utilization. Figure 7 shows that `EdgeUP` raises the lowest idle CPU distance among the load balancing mechanisms across various reporting periods. For example, with the shortest reporting period $p = 5$, `EdgeUP` outperforms round-robin, hill climbing, and K-Means by 5.6x, 2.1x, and 3.6x, respectively.

5.2.3. Execution Time

Finally, Table 3 presents a comparison of the execution time between `EdgeUP` and other load balancing methods.

Table 3. Execution time of load balancing mechanisms.

	Round-Robin	**K-Means**	**Hill Climbing**	**EdgeUP**
Execution time	0.21 µs	1.1 ms	0.03 s	0.6 ms

The table shows the execution time for each load balancing mechanism to produce a task assignment on a Raspberry Pi 400. Among the methods, round-robin demonstrates the shortest execution time, with `EdgeUP` following closely behind. Although `EdgeUP` takes slightly longer to execute than round-robin, the difference is minimal. Notably, `EdgeUP` outperforms other load balancing mechanisms (i.e., K-Means and hill climbing) in terms of execution time, underscoring its lightweight nature.

This outcome aligns with expectations, as round-robin is a fundamental load balancing approach that requires minimal computation to generate task assignments. In contrast, K-Means and hill climbing exhibit longer execution times compared to round-robin and `EdgeUP`. As learning and optimization-based methods, they involve more complex operations to achieve optimal results, contributing to their extended execution time.

6. Conclusions

This study proposes `EdgeUP`, a lightweight load balancing mechanism for edge computing in IoT applications. `EdgeUP` introduces a new load balancing metric, named `ServingValue`, which considers both dynamic contexts of edge nodes (i.e., CPU utilization) and IoT devices (i.e., task priority). `EdgeUP` allocates tasks to edge nodes by identifying pairs of tasks and edge nodes that result in the highest `ServingValue`. By maximizing `ServingValue` at each task assignment, `EdgeUP` converges the CPU utilization of edge nodes, avoiding the overloaded state of edge nodes in edge computing. `EdgeUP` outperforms load balancing mechanisms using conventional load balancing metrics in terms of CPU utilization similarity, idle CPU distance, and execution time.

In addition to considering the dynamic context of IoT devices (i.e., task priority), `EdgeUP` currently prioritizes a single computing context (i.e., CPU utilization) when balancing workloads. To improve its effectiveness, it could benefit from incorporating additional computing factors such as memory consumption, input/output performance, and other aspects of the load balancing, including response latency, task completion time, cost, etc.

Author Contributions: L.A.N., conceptualization, methodology, software, validation, data curation, writing—original draft preparation, visualization; S.K., formal analysis, investigation, writing–review and editing; Y.S., writing–review and editing, resources, supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Research Foundation of Korea (NRF) (No. NRF-2022R1A4A5034130) and the Korea Institute for Advancement of Technology (KIAT) (No. KIAT-P0012724) grant funded by the Korean Government. This research was supported by the Chung-Ang University research grant in 2024 (Corresponding Author: Yongseok Son).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author(s).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Rearrangement Inequality

In mathematics, the rearrangement inequality states that
For $x_1 \leq \ldots \leq x_n$ and $y_1 \leq \ldots \leq y_n$
and every permutation σ of the numbers $1, 2, \ldots, n$, we have
$x_1 y_n + \ldots + x_n y_1 \leq x_1 y_{\sigma(1)} + \ldots + x_n y_{\sigma(n)} \leq x_1 y_1 + \ldots + x_n y_n$

References

1. Kumar, S.; Tiwari, P.; Zymbler, M. Internet of Things is a revolutionary approach for future technology enhancement: A review. *J. Big Data* **2019**, *6*, 111. [CrossRef]
2. Maiti, P.; Shukla, J.; Sahoo, B.; Turuk, A.K. Efficient data collection for IoT services in edge computing environment. In Proceedings of the 2017 International Conference on Information Technology (ICIT), Singapore, 27–29 December 2017; pp. 101–106.
3. Zhu, G.; Xu, J.; Huang, K.; Cui, S. Over-the-air computing for wireless data aggregation in massive IoT. *IEEE Wirel. Commun.* **2021**, *28*, 57–65. [CrossRef]
4. Lukaj, V.; Martella, F.; Fazio, M.; Galletta, A.; Celesti, A.; Villari, M. Gateway-Based Certification Approach to Include IoT Nodes in a Trusted Edge/Cloud Environment. In Proceedings of the 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW), Bangalore, India, 1–4 May 2023; pp. 237–241.
5. Ferrag, M.A.; Debbah, M.; Al-Hawawreh, M. Generative ai for cyber threat-hunting in 6g-enabled iot networks. *arXiv* **2023**, arXiv:2303.11751.
6. Cecchinel, C.; Jimenez, M.; Mosser, S.; Riveill, M. An architecture to support the collection of big data in the internet of things. In Proceedings of the 2014 IEEE World Congress on Services, Anchorage, AK, USA, 27 June–2 July 2014; pp. 442–449.
7. Khare, S.; Totaro, M. Big data in IoT. In Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019; pp. 1–7.
8. Sasaki, Y. A survey on IoT big data analytic systems: Current and future. *IEEE Internet Things J.* **2021**, *9*, 1024–1036. [CrossRef]
9. Verma, S.; Kawamoto, Y.; Fadlullah, Z.M.; Nishiyama, H.; Kato, N. A survey on network methodologies for real-time analytics of massive IoT data and open research issues. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1457–1477. [CrossRef]
10. Taherkordi, A.; Eliassen, F.; Horn, G. From IoT big data to IoT big services. In Proceedings of the Symposium on Applied Computing, Marrakech, Morocco, 4–6 April 2017; pp. 485–491.
11. Jang, H.S.; Jin, H.; Jung, B.C.; Quek, T.Q. Versatile access control for massive IoT: Throughput, latency, and energy efficiency. *IEEE Trans. Mob. Comput.* **2019**, *19*, 1984–1997. [CrossRef]
12. Shukla, S.; Hassan, M.F.; Tran, D.C.; Akbar, R.; Paputungan, I.V.; Khan, M.K. Improving latency in Internet-of-Things and cloud computing for real-time data transmission: A systematic literature review (SLR). In *Cluster Computing*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 1–24.
13. Pereira, C.; Pinto, A.; Ferreira, D.; Aguiar, A. Experimental characterization of mobile IoT application latency. *IEEE Internet Things J.* **2017**, *4*, 1082–1094. [CrossRef]
14. Chen, B.; Wan, J.; Celesti, A.; Li, D.; Abbas, H.; Zhang, Q. Edge computing in IoT-based manufacturing. *IEEE Commun. Mag.* **2018**, *56*, 103–109. [CrossRef]
15. Sun, X.; Ansari, N. EdgeIoT: Mobile edge computing for the Internet of Things. *IEEE Commun. Mag.* **2016**, *54*, 22–29. [CrossRef]
16. Chen, X.; Shi, Q.; Yang, L.; Xu, J. ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications. *IEEE Netw.* **2018**, *32*, 61–65. [CrossRef]

17. Naveen, S.; Kounte, M.R. Key technologies and challenges in IoT edge computing. In Proceedings of the 2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 12–14 December 2019; pp. 61–65.
18. Alnoman, A.; Sharma, S.K.; Ejaz, W.; Anpalagan, A. Emerging edge computing technologies for distributed IoT systems. *IEEE Netw.* **2019**, *33*, 140–147. [CrossRef]
19. Liu, X.; Yu, J.; Wang, J.; Gao, Y. Resource allocation with edge computing in IoT networks via machine learning. *IEEE Internet Things J.* **2020**, *7*, 3415–3426. [CrossRef]
20. Long, C.; Cao, Y.; Jiang, T.; Zhang, Q. Edge computing framework for cooperative video processing in multimedia IoT systems. *IEEE Trans. Multimed.* **2017**, *20*, 1126–1139. [CrossRef]
21. Sivarajah, U.; Kamal, M.M.; Irani, Z.; Weerakkody, V. Critical analysis of Big Data challenges and analytical methods. *J. Bus. Res.* **2017**, *70*, 263–286. [CrossRef]
22. Satyanarayanan, M. The Emergence of Edge Computing. *Computer* **2017**, *50*, 30–39. [CrossRef]
23. Kashani, M.H.; Ahmadzadeh, A.; Mahdipour, E. Load balancing mechanisms in fog computing: A systematic review. *arXiv* **2020**, arXiv:2011.14706.
24. Ashouri, M.; Davidsson, P.; Spalazzese, R. Quality attributes in edge computing for the Internet of Things: A systematic mapping study. *Internet Things* **2021**, *13*, 100346. [CrossRef]
25. Fan, Q.; Ansari, N. Towards Workload Balancing in Fog Computing Empowered IoT. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 253–262. [CrossRef]
26. Zhang, J.; Guo, H.; Liu, J.; Zhang, Y. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2092–2104. [CrossRef]
27. Dong, Y.; Xu, G.; Ding, Y.; Meng, X.; Zhao, J. A 'Joint-Me' Task Deployment Strategy for Load Balancing in Edge Computing. *IEEE Access* **2019**, *7*, 99658–99669. [CrossRef]
28. Yu, Y.; Li, X.; Qian, C. SDLB: A Scalable and Dynamic Software Load Balancer for Fog and Mobile Edge Computing. In Proceedings of the Workshop on Mobile Edge Communications, New York, NY, USA, 21 August 2017; MECOMM '17; pp. 55–60. [CrossRef]
29. Li, G.; Yao, Y.; Wu, J.; Liu, X.; Sheng, X.; Lin, Q. A new load balancing strategy by task allocation in edge computing based on intermediary nodes. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 3. [CrossRef]
30. Yu, R.; Kilari, V.T.; Xue, G.; Yang, D. Load balancing for interdependent IoT microservices. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 298–306.
31. Li, P.; Xie, W.; Yuan, Y.; Chen, C.; Wan, S. Deep reinforcement learning for load balancing of edge servers in iov. *Mob. Netw. Appl.* **2022**, *27*, 1461–1474. [CrossRef]
32. Baek, J.y.; Kaddoum, G.; Garg, S.; Kaur, K.; Gravel, V. Managing fog networks using reinforcement learning based load balancing algorithm. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019; pp. 1–7.
33. Esmaeili, M.E.; Khonsari, A.; Sohrabi, V.; Dadlani, A. Reinforcement learning-based dynamic load balancing in edge computing networks. *Comput. Commun.* **2024**, *222*, 188–197. [CrossRef]
34. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [CrossRef]
35. Ray, P.P. A survey on Internet of Things architectures. *J. King Saud Univ. Comput. Inf. Sci.* **2016**, *30*, 291–319. [CrossRef]
36. Jin, J.; Gubbi, J.; Marusic, S.; Palaniswami, M.S. An Information Framework for Creating a Smart City Through Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 112–121. [CrossRef]
37. Pydi, H.P.R.; Iyer, G.N. Analytical Review and Study on Load Balancing in Edge Computing Platform. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 180–187.
38. Khan, W.; Ahmed, E.; Hakak, S.; Yaqoob, I.; Ahmed, A. Edge computing: A survey. *Future Gener. Comput. Syst.* **2019**, *97*. [CrossRef]
39. Zhang, J.; Elnikety, S.; Zarar, S.; Gupta, A.; Garg, S. {Model-Switching}: Dealing with Fluctuating Workloads in {Machine-Learning-as-a-Service} Systems. In Proceedings of the 12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20), Online, 13–14 July 2020.
40. Curiel, M.; Pont, A. Workload generators for web-based systems: Characteristics, current status, and challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1526–1546. [CrossRef]
41. Fischer, W.; Meier-Hellstern, K. The Markov-modulated Poisson process (MMPP) cookbook. *Perform. Eval.* **1993**, *18*, 149–171. [CrossRef]
42. Computing, T.C.; The Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory. CloudSim: A Framework For Modeling and Simulation of Cloud Computing Infrastructures and Services. Available online: http://www.cloudbus.org/cloudsim/ (accessed on 27 November 2023).

43. Li, F. CloudSimPy. Available online: https://github.com/FengcunLi/CloudSimPy (accessed on 27 November 2023).
44. Sonmez, C.; Ozgovde, A.; Ersoy, C. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3493. [CrossRef]
45. Kong, X.; Wu, Y.; Wang, H.; Xia, F. Edge Computing for Internet of Everything: A Survey. *IEEE Internet Things J.* **2022**, *9*, 23472–23485. [CrossRef]
46. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
47. De Bruijn, B.; Nguyen, T.A.; Bucur, D.; Tei, K. Benchmark datasets for fault detection and classification in sensor data. In Proceedings of the International Confererence on Sensor Networks, Vienna, Austria, 11–14 April 2016; pp. 185–195.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.