Future Generation Computer Systems 158 (2024) 501–515

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Elastic Federated Learning with Kubernetes Vertical Pod Autoscaler for edge computing

Khanh Quan Pham, Taehong Kim *

School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, Republic of Korea

ARTICLE INFO

Keywords:
Federated learning
Autoscaling
Kubernetes
KubeEdge
Resource allocation

ABSTRACT

Federated Learning (FL) is an emerging paradigm for training machine learning models across decentralized edge devices, ensuring data privacy and reducing computational tasks on the central cloud. However, the dynamic and resource-constrained nature of edge environments raises significant challenges in deploying FL applications efficiently. This paper introduces an Elastic Federated Learning framework that leverages Kubernetes Vertical Pod Autoscaler to improve the performance of FL applications. The framework enables dynamic adjustment of computational resources, facilitating effective resource utilization for model training and allowing quicker attainment of the targeted accuracy levels for the trained models. Our experiments demonstrate that the proposed framework significantly enhances FL efficiency, maintains model training progress during resource adjustments, and effectively addresses the resource allocation challenges in edge environments. This work advances the capabilities of FL in edge computing, opening doors to more efficient and scalable AI applications while preserving data privacy.

1. Introduction

Federated Learning (FL) has emerged as a promising alternative to traditional centralized model training methods [1], aiming to mitigate the privacy and security risks inherent in data aggregation. Unlike traditional approaches that necessitate centralizing data for model training, FL enables collaborative training across decentralized edge devices. By keeping sensitive data on individual devices, FL minimizes the privacy concerns associated with centralized data aggregation, thus ensuring greater data privacy and security.

As the demand for edge computing continues to grow, the integration of container orchestration platforms such as Kubernetes [2] and KubeEdge [3] plays an important role in efficiently deploying and managing FL tasks at the edge. However, several critical challenges [4] persist in managing resource utilization within dynamic and decentralized edge environments. Subsequent paragraphs will delve into these challenges in detail, including factors such as system heterogeneity and data heterogeneity.

System heterogeneity refers to the diversity of edge node's computational resources, ranging from lightweight devices with limited computational resources to nodes with abundant computational resources [5]. These capabilities include variations in processing capacities, memory, and storage, influencing their suitableness for different tasks. Edge nodes can be sensors, smartphones, Internet of Things (IoT) devices, or edge servers, each with separate computing resources and communication capabilities [6]. For instance, low-resource nodes may take longer to complete their training tasks compared to high-resource nodes, leading to training time distinctions across the FL system [7]. Besides, system heterogeneity can lead to challenges in aggregating models from nodes with different computation capabilities, potentially affecting the overall model convergence performance [7]. For system heterogeneity, in resource allocation, Vertical Pod Autoscalor (VPA) allows for dynamic adjustment of computing resources based on the capabilities of individual edge nodes. It ensures that lightweight devices receive appropriate computational resources, preventing overload, while more powerful edge nodes can handle larger and more complex tasks effectively. Importantly, it considers real-time allocations in the Central Processing Unit (CPU), Random Access Memory (RAM), and storage, improving the utilization of available resources for diverse tasks.

Data heterogeneity is also known as non-Independent and Identically Distributed (non-IID) data distribution [8] where the data available across different nodes in a network is not uniformly or identically distributed. This variability potentially impacts the model's performance as the number of training samples among clients also differs significantly. Addressing non-IID data is crucial in FL to ensure effective model training across diverse and distributed edge nodes. For example, nodes with more data may require higher computational resources, impacting the overall performance of FL, especially in

* Corresponding author.
E-mail addresses: khanhquan@cbnu.ac.kr (K.Q. Pham), taehongkim@cbnu.ac.kr (T. Kim).

https://doi.org/10.1016/j.future.2024.04.047
Received 19 January 2024; Received in revised form 27 March 2024; Accepted 25 April 2024
Available online 1 May 2024

resource-constrained environments [9]. Moreover, variability in node resources can lead to imbalances in resource utilization, where some nodes are underutilized, while others may be overutilized [10]. For data heterogeneity, by dynamically scaling resources based on pod history storage and real-time usage, VPA helps mitigate the challenges associated with non-IID and varying numbers of training data across edge devices. These above challenges underscore the critical need for advanced solutions in FL.

In this study, to address the aforementioned system and data heterogeneity in the edge computing environment, we build an edge computing-based FL framework using KubeEdge [7] for edge cluster orchestration and Flower [11] for FL service execution. Moreover, we apply a dynamic resource autoscaling mechanism called VPA to monitor the resource status of individual devices in real-time, capturing their computational capabilities and resource availability. By analyzing time-series historical and real-time data, our framework dynamically adjusts the resources allocated to container-based application services based on the actual application demands. To the best of our knowledge, this is the first study to propose applying dynamic autoscaling features to the edge computing-based FL framework. The following are the detailed contributions of this study:

- **VPA in Elastic Federated Learning framework**: We utilize a KubeEdge-based FL framework [7] to enable the deployment of containerized FL applications to edge nodes that are geographically distributed with heterogeneous hardware resources and data. We enhance elasticity in containerized resources upon this base framework by applying Kubernetes VPA. It can effectively balance the computational resources across nodes with different capabilities in an edge computing environment.
- **Experimental testbed in real edge node scenarios:** To evaluate the diverse effect of VPA on system and data heterogeneity, we establish an experimental testbed utilizing containerized FL applications within an edge computing environment, where each node may possess different computing resources and data. Additionally, we analyze diverse performance metrics such as local training time and convergence time, as well as recommended and actual CPU/RAM resource usage of individual nodes.
- **Lessons derived from performance analysis:** Our study derives diverse lessons from the performance analysis of deploying VPA in the FL scenarios. These lessons include best practices, insights into the most suitable configurations, and recommendations for adapting VPA to specific edge use cases.

The rest of this paper is structured as follows. reviews related works on dynamic resource management and autoscaling techniques that relate to FL in edge computing environments. Section 3 describes FL and autoscaling techniques in edge computing environments. Section 4 describes our Elastic Federated Learning (EFL) framework with Kubernetes VPA. Section 5 analyzes the performance evaluation results and summarizes the lessons considering VPA on system heterogeneity, data heterogeneity, and heterogeneous environments. Section 6 discusses the limitations of our EFL framework, focusing on the impact of diverse factors in resource allocation and the management overhead associated with VPA. Finally, we conclude our study and present the scope for future works in Section 7.

2. Related works

This section reviews related studies that address the existing challenges and evaluate the performance of dynamic resource management and autoscaling techniques that relate to FL in edge computing environments. A comprehensive comparison of the studies listed above is presented in Table 1.

Lina et al. [12] provided a comprehensive overview of Mobile Edge Computing (MEC) infrastructure in the context of emerging 5G cellular networks. They also emphasized the importance of virtualization, autoscaling mechanisms, MEC frameworks, and optimization strategies for meeting Quality of Service (QoS) thresholds in a dynamic mobile environment, as well as addressing the challenges in distributed resource allocation management. The dynamic nature of MEC systems, influenced by device mobility and changing computing application requirements, requires a multi-criteria resource management technique, which is complicated by the variability in application types, diverse user requirements, and different QoS needs for communication channels. Therefore, investigating reliable and distributed MEC resource allocation schemes to handle dynamic behaviors in heterogeneous environments is essential work.

Zhang et al. [13] provided a comprehensive review of existing research problems, solutions, and open challenges in MEC resource management, covering diverse performance requirements, dynamic environments, and application-specific solutions, thereby guiding future research directions in this emerging field. Besides, Leylane et al. [14] reviewed current advances in resource optimization for FL in IoT scenarios, addressing challenges related to limited edge device resources, communication issues, and optimizing processing and communication resources for effective FL implementation in IoT environments. However, as mentioned in [12–14], there is a noticeable absence of solutions addressing resource allocation in heterogeneous MEC environments, prompting the need for further research in this area.

Thiago et al. [15] introduced an innovative auto-scaling subsystem for container-based processing services in edge computing environments, leveraging online machine learning. The proposed system, based on the MAPE-K control loop, dynamically adjusts the number of containers in response to workload changes. However, the authors neither evaluated their auto-scaling subsystem using real resource-constrained devices to measure CPU and memory usage nor deployed their proposed approach in a real application benchmark for practical validation.

Phuc et al. [2] introduced THPA, a traffic-aware horizontal pod autoscaling for Kubernetes, enhancing resource scaling for IoT applications in edge computing environments based on real-time network traffic information, significantly improving performance compared to the standard horizontal pod autoscaling.

Majid et al. [16] proposed node-based horizontal pod autoscaling (NHPA) to address unreliable communication links in KubeEdge. NHPA significantly improves the throughput and response time of KubeEdge-based edge computing environments by dynamically adjusting the number of pods based on incoming traffic.

Silvana et al. [17] provided a thorough overview of recent work on resource management at the edge, specifically focusing on challenges and future directions related to the execution of FL at the edge. The study addressed various aspects of resource management, including discovery, deployment, load balancing, migration, and energy efficiency, raising unsolved problems while presenting potential solutions. However, hardware resources like CPU and memory are not extensively explored in this survey.

Dong Yang et al. [18] proposed a multi-agent approach to optimize FL in distributed industrial IoT environments. They proposed optimal device selection and resource allocation decisions using multi-agent reinforcement learning, demonstrating its ability to improve FL accuracy while satisfying on-device energy consumption requirements. Similarly, Adeb et al. [19] proposed a resource allocation technique to optimize energy consumption while enhancing FL accuracy and reducing learning time in B5G-based IoT networks.

Xiao et al. [20] proposed a framework that optimizes vehicle selection and resource allocation decisions to enhance FL performance and efficiency for vehicular edge computing environments considering real-time requirements and distinct capabilities of individual vehicles. They proved that the optimization of the resource allocation reduces the system cost, energy consumption, and training time in an edge computing environment.

Table 1
Existing studies of resource allocation for FL in edge computing environments.

Domain	Paper	Contribution	DRA	FL	ECP
General	[12]	Provide insights into MEC infrastructure in 5G networks, emphasizing auto-scaling mechanisms, and optimization strategies.	–	–	–
	[13]	Survey MEC resource management issues, solutions, and applications.	–	–	–
	[14]	Explore optimizing resource usage for FL with IoT devices and consider constraints and communication issues.	–	–	–
Edge computing framework	[15]	Introduce an auto-scaling subsystem for container-based processing services using online machine learning.	✓	✗	✗
	[2]	Propose THPA, enhancing Kubernetes autoscaling in edge computing by considering real-time network traffic.	✓	✗	✓
	[16]	Propose NHPA to address communication issues and lack of load balancing in KubeEdge.	✓	✗	✓
Edge computing for FL	[17]	Addressing load balancing and energy issues by proposing resource management solutions in FL at the network edge.	✓	✓	✗
	[18]	Suggest a dynamic resource scheduling algorithm utilizing deep reinforcement learning to determine optimal resource allocation decisions.	✓	✓	✗
	[19]	Propose an edge intelligence-aided IoT network to optimize energy consumption and reduce computational complexity.	✓	✓	✗
	[20]	Propose a framework that optimizes vehicle selection and resource allocation decisions to enhance FL performance.	✓	✓	✗
	[21]	Explore resource allocation methods for FL in edge computing and computational constraints for diverse workflows.	✗	✓	✓
	[22]	Propose FedAvg-BiGRU, for proactive auto-scaling in edge computing.	✓	✓	✓

DRA: Dynamical Resource Adjustment, FL: Federated Learning, ECP: Edge Computing Platform.

Fotis et al. [21] investigated resource allocation methods for federated learning in a virtualized managed environment considering computational capability, network bandwidth, data complexity, and FL workflow characteristics. They developed an experimental testbed with Kubernetes container orchestration framework and analyzed CPU, memory, and network resource usage for diverse datasets such as CIFAR-10 and MNIST in virtualized environments. This work provides insights into how resource allocation can be tailored to enhance the performance of FL systems in an edge computing framework, however, they applied manual CPU adjustment, which is a lack of dynamic adjustment of resources according to workload demands.

Javad et al. [22] proposed FedAvg-BiGRU, for proactive auto-scaling in edge computing using federated averaging (FedAvg) and bidirectional gated recurrent unit (BiGRU) for multi-step workload prediction, determining the number of Kubernetes pods based on the cool-down time (CDT) concept. However, their approach faces challenges in implementing and managing the FedAvg-BiGRU method in real-world edge computing systems, which may impact its practical deployment and scalability.

In the edge computing framework domain, several works have applied dynamic resource adjustment in edge computing environments using Kubernetes [2] and KubeEdge [16] platforms, but they have primarily evaluated general edge computing applications without considering the specific requirements of FL scenarios. Besides, [15] does not consider using any real edge computing platform for the experimental environment.

Transitioning to edge computing for the FL domain, on the other hand, there have been several studies [17–22] proposing resource allocation algorithms for FL in edge computing settings. However, most of these works focused solely on suggesting algorithms without actually applying and verifying the proposed techniques through practical implementations of edge computing platforms. Only [21,22] developed experimental testbeds using Kubernetes or other edge computing frameworks, but [21] has not applied the dynamical resource adjustment scheme; it just evaluated the effect of resource allocation on FL performances. Furthermore, [22] applied an autoscaling mechanism, but their mechanism concentrated solely on adjusting the number of Kubernetes pods, overlooking the fine-grained adjustment of each pod's resources. In summary, there has been a gap in efforts to apply dynamic autoscaling techniques to improve the performances of FL workloads running on edge computing platforms.

To the best of our knowledge, this study is the first to apply VPA in an edge computing platform to improve the performance of FL. As proven by existing works that evaluated FL performances without edge computing platforms, research on FL over the edge computing platform requires deep knowledge of both domains. It is not easy to implement FL applications over an edge computing platform without a comprehensive understanding of the edge computing platform. Especially, applying the VPA methodology can be achieved through extensive experience with dynamic resource allocation strategies of the platform, together with a full understanding of the hardware capabilities of edge nodes integrated with existing FL frameworks.

In summary, the goal of our study is to pave the way for a more realistic and efficient approach to FL execution at the edge by leveraging VPA. Through this approach, the study seeks to develop improved resource management strategies and expand the capabilities of edge computing environments, addressing limitations observed in the above existing studies and promoting the implementation of VPA for enhanced FL performance in heterogeneous edge computing environments.

3. Preliminaries

3.1. Federated learning in edge computing

FL is a machine learning approach that allows a centralized model to be trained across multiple devices or edge nodes, such as smartphones, IoT devices, or local servers, without sharing raw data. Instead of sending data to a central server for training, FL keeps data on edge devices, reserving privacy and security [23]. Here's an overview of each stage in the FL process:

- **Task initialization:** In this initial stage, a central server defines a machine learning model, which can be the convolutional neural

networks (CNN) [24], long short-term memory (LSTM) [25], etc. The server selects a subset of available edge devices or clients that will participate in the FL process. These clients could be smartphones, sensors, or any device with data relevant to the task.

- **Local model training:** After task initialization, the central server sends the machine learning model to the selected edge devices. Each edge device independently trains the model on its local data. This training typically occurs over multiple epochs, allowing the model to learn from the device's data. The training process updates the model's parameters, adapting it to the specific characteristics of the data on each device.
- **Local model updating:** Once the local model training is complete, each edge device calculates an update to the model's parameters. This update represents the device's knowledge gained during training. Importantly, no raw data is transmitted from the edge devices. Instead, only the model update or weight value, which is a set of numerical values, is shared with the central server.
- **Global model aggregation:** The central server collects the model updates from all participating edge devices. It performs an aggregation process, typically averaging or combining these updates, to create a new, global model with improved parameters. The global model now benefits from the collective knowledge of all the edge devices without ever seeing their raw data.

The FL process goes through multiple rounds of local model training, updating, and global model aggregation until the model has achieved the expected accuracy level.

In each round, the global model becomes more accurate and refined, as it learns from diverse data sources across different edge devices. One of the famous aggregation techniques is FedAvg [26], which is a weighted average of the individual clients' losses, derived as follows:

$$w_{\text{new}} = \frac{1}{K}\sum_{k=1}^{K}\left(w_k - \eta \nabla L(w_k, X_k, Y_k)\right) \tag{1}$$

where w_{new} represents the updated global model, K is the number of selected clients for the current communication round, $\sum_{k=1}^{K}$ is a summation notation that sums over K clients or devices, w_k represents the local model of client k, η is the learning rate, and $\nabla L(w_k, X_k, Y_k)$ represents the gradient of the loss function concerning the local model w_k on the data samples X_k and labels Y_k of client k.

3.2. Autoscaling techniques in edge computing environments

When choosing an edge computing platform, it is crucial to assess its strengths and limitations. For instance, Docker Swarm [27] provides seamless orchestration of container infrastructures but lacks built-in support for auto-scaling and load-balancing [28]. Apache Mesos [29] excels in large-scale clustering but presents challenges in terms of complexity and management overhead due to its lower level of abstraction compared to Kubernetes [30]. Kubernetes [30], managed by CNCF, offers comprehensive container management features with fewer third-party dependencies. However, Kubernetes may not be suitable for edge scenarios and needs to be installed on high-resource devices [31]. On the other hand, KubeEdge [3] is an extension of Kubernetes that provides lightweight deployment [32], efficient networking [33], and data synchronization [34], making it well-suited for resource-constrained edge environments.

In the Kubernetes and KubeEdge platforms, three key components play a crucial role in resource allocation and management, especially in the context of FL in edge computing. These components are Cluster Autoscaler (CA) [35], Horizontal Pod Autoscalor (HPA) [36], and VPA [37]. This section discusses these components in terms of resource allocation in FL within edge environments and derives the reason why VPA offers significant benefits compared to other methods in boosting FL performance.

CA [35] is a Kubernetes component that automatically adjusts the number of nodes in a cluster based on resource demands and node availability. It adds nodes when resources are insufficient and removes underutilized nodes to optimize resource allocation to ensure efficient application performance. However, scaling up nodes in FL for edge computing may raise privacy concerns as it involves sharing local data. If not adequately protected during transfer, data security may be compromised.

HPA [36] in Kubernetes automatically adjusts the number of pod replicas based on CPU and memory utilization to maintain application performance. It scales up when metrics exceed defined targets and scales down during decreased load, helping manage resource allocation. However, integrating HPA into FL applications introduces challenges, especially when increasing the number of pods via HPA, a critical concern is the duplication of training tasks on the same local node, resulting in unnecessary CPU and memory resource consumption.

VPA [37] is a component in the Kubernetes ecosystem that helps manage and improve the resource allocation for individual pods within a cluster. Unlike HPA [36], which focuses on adjusting the number of pods or replicas to meet traffic demands, VPA is concerned with fine-tuning the resource requests and limits of individual pods to ensure they have the right amount of CPU and memory resources allocated. VPA constantly monitors the resource utilization of pods and can automatically adjust their resource requests and limits based on observed usage patterns. If a pod is consistently using more or less CPU or memory than initially requested, VPA can dynamically modify these resource parameters to improve resource efficiency, reduce resource wastage, and achieve better overall cluster performance. Based on these characteristics, VPA can potentially improve FL performance in an edge node by optimizing resource allocation for the pods involved in FL tasks. Here's how the proposed framework with VPA-integrated can contribute to FL performance improvement:

- **Resource Efficiency:** VPA continuously monitors the resource utilization of FL pods and adjusts their resource requests and limits dynamically. This means that FL pods are more likely to have resource settings that match their actual resource needs. In FL, where multiple pods may perform training simultaneously, efficient resource allocation is crucial.
- **Stability:** FL relies on the synchronized execution of training tasks across multiple edge nodes. Any instability or resource bottlenecks can interrupt this synchronization, leading to delays in model updates. VPA's resource optimization helps maintain cluster stability by preventing Out-of-memory (OOM) issues and potential pod evictions due to resource constraints. Besides, we implement the *Deployment* rolling update mechanism, ensuring that newly created pods seamlessly take over the FL task of the evicted pods, eliminating the need to start training from scratch.
- **Scalability:** FL applications may require scaling up or down the pod resources requirement to accommodate increased or decreased workloads or data volumes. VPA can help with the dynamic scaling of resources, ensuring that new pods added during scaling activities are appropriately allocated resources. This scalability is essential for FL applications that need to adapt to changing data and workload demands.

In summary, VPA stands out for its resource allocation capabilities, offering benefits in terms of better utilization, stability, and scalability, making it a promising solution to enhance FL performance in edge environments. VPA's ability to dynamically adjust resources aligns well with FL's requirements, ensuring efficient training and adaptability to varying workloads.

4. Elastic federated learning framework with Kubernetes VPA

In our previous work [7], we proposed a KubeEdge-based FL framework that deploys containerized FL applications in the edge computing environment. In this study, we advance our work by proposing an EFL framework, achieved by integrating Kubernetes VPA into the existing KubeEdge-based FL framework. VPA, when deployed in a KubeEdge-based environment, can dynamically adjust resource allocation for individual pods to improve performance, making it more suitable for edge nodes.

This section explains the architecture of the EFL framework and VPA algorithm to show how VPA works to dynamically manage resource allocations for pods within a KubeEdge cluster.

4.1. Architecture of EFL framework

Fig. 1 illustrates the architecture of the EFL framework, consisting of FL cloud node and FL edge node. While FL edge node runs the FL application on KubeEdge, FL cloud node provides management of pod resources adjustment on Kubernetes.

FL edge node consisted of FL pod and VPA API object. While FL pod manages the execution of FL application, handling resource allocation such as CPU request/limit and memory request/limit, VPA API object is used to configure and manage VPA operation on FL pods. Each VPA API object monitors an FL pod on an edge node. It gives instructions for specifying vertical autoscaling behavior and policies.

FL cloud node comprises four key modules: Metrics Server, History Storage, VPA Admission Plugin, and VPA Controller.

Metrics Server collects real-time resource information from FL nodes/pods and then archives this data in the History Storage using a time-series data structure. It provides a way to query and access this data, allowing users and other containers to make informed decisions about resource allocation and scaling. These data points are obtained from Metrics Server and are stored securely on Prometheus [38] for future reference.

History Storage refers to resource utilization data from edge nodes or pods. This historical data, including CPU usage and memory usage, is crucial for improving VPA performance. By analyzing historical patterns, VPA can estimate future resource demands and evaluate performance over time. Ultimately, History Storage enables data-driven decision-making in resource management, leading to improved efficiency and stability in edge computing environments.

VPA Admission Plugin collaborates with Recommender to apply resource recommendations to new pod creation requests based on the

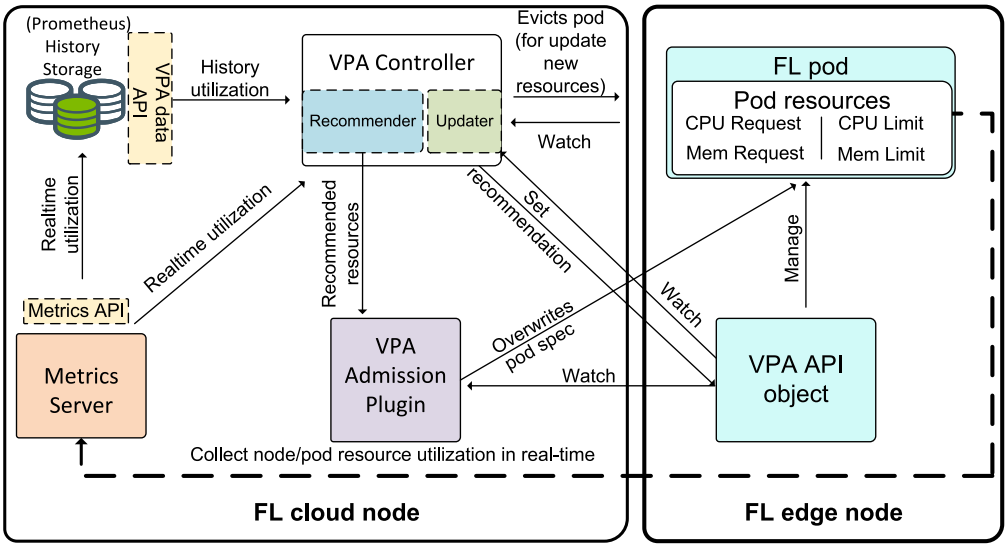


Fig. 1. Architecture of EFL framework.

Table 2
List of parameters in VPA algorithm.

Parameter	Description
$upaList$	List of VPA API objects from all nodes.
$FLpod$	FL pod in the iteration.
$historyDict$	Resource utilization history data.
$historyList$	Extracted resource list from historyDict.
$sortHistoryList$	Resource list sorted in ascending based on CPU and RAM values.
min_{CPU}, min_{RAM}	Minimum CPU and RAM for a pod recommended by VPA.
$SafetyMargin(SM)$	Safety margin or scaling multiplier.
$ratio_{CPU}$, $ratio_{RAM}$	Ratio between initial CPU/RAM limit and request set by user.
$iniLimitCPU$, $iniLimitRAM$	Initial CPU/RAM limit set by user.
$iniRequestCPU$, $iniRequestRAM$	Initial CPU/RAM request set by user.
$rec_{CPU,RAM}$	Recommended CPU and RAM resource for the FL pod.
$req_{CPU,RAM}$	Current CPU and RAM resource requests for FL pod.
$lim_{CPU,RAM}$	Current CPU and RAM resource limits for FL pod.
$diff_{CPU}$, $diff_{RAM}$	Differences between current and recommended resources.
$target$	Parameters for computing target estimate.
$targetEstimate_{CPU,RAM}$	Target resources (CPU and RAM) for the recommendations.
$minResources_{CPU,RAM}$	Minimum resources for handling no history situation.
$threshold$	Threshold parameter for the Updater function.

VPA API object. This plugin ensures that the resource recommendations from VPA are appropriately applied to the creation of new pods. Additionally, it can overwrite the pod specifications based on the configurations defined in the VPA configuration.

VPA Controller, including Recommender and Updater modules, utilizes real-time and historical resource consumption data to generate CPU and memory request recommendations for pods, with Updater managing pod resource settings and implementing recommendations through pod termination and recreation. Three main modes that control VPA's behavior are described below:

- **Off**: In this mode, VPA is disabled but the recommender still sets the recommended resources in the VPA API object. However, FL pods remain unaffected by VPA's resource adjustments.
- **Initial**: In the "Initial" mode, VPA only provides initial resource recommendations for newly created pods. It does not actively adjust resources for running pods. This mode is suitable for scenarios where we want VPA to suggest resource requests when pods are created but not change them afterward.
- **Auto**: The "Auto" mode is the most dynamic one. In this mode, VPA continuously monitors pods and, if necessary, adjusts their resource requests based on their actual resource usage. It can even evict pods to force them to be recreated with updated resource requests if they significantly diverge from the recommendations.

Recommender receives real-time utilization information directly from the Metrics Server. It also accesses historical utilization data stored in the History Storage. By combining both real-time and historical data, Recommender can make appropriate recommendations for resource allocations based on past and current resource usage patterns. This hybrid approach ensures that the recommendations are both flexible to immediate needs and considerate of long-term resource trends.

Updater takes an important role in fulfilling this task within the VPA. When a pod operates in "Auto" mode with VPA, it can decide to update it with the latest recommendations. Updater frequently monitors all VPA API objects and FL pods in the KubeEdge cluster. It compares the recommended resources obtained from Recommender with the currently configured resources request of the FL pods. If the recommended resource values significantly differ from the existing pod configurations, indicating a potential need for resource adjustment, Updater decides to make an update by evicting the old pod. Once a pod is evicted, VPA Controller detects the termination and takes action to recreate the pod. The newly created pod will have the recommended resource requests provided by VPA. These resource requests are aligned with the pod's actual resource needs, managing resource utilization and ensuring the pod has the necessary resources to run efficiently. Specifically, we used the **Deployment** rolling update mechanism so the newly created pod will continue the FL task of the evicted pod without training from the beginning.

Algorithm 1 VPA Algorithm

Input: $min_{CPU} = 1000$, $min_{RAM} = 2048$, $SafetyMargin(SM) = 0.15$, $ratio_{CPU} = \frac{iniLimitCPU}{iniRequestCPU}$, $ratio_{RAM} = \frac{iniLimitRAM}{iniRequestRAM}$.
Output: Recommend resources: $rec_{CPU,RAM}$.

procedure
  Retrieve the list of VPA API objects from all nodes
  $vpaList = [vpaEdge1, vpaEdge2, vpaEdge3, ...]$
  while len($vpaList$) >= 1 **do**
    for $FLpod \in vpaList$ **do**
      // Get the historical and the real-time pod utilization under time-series data:
      $historyDict$ = {
      - Timestamp VPA_API_object CPU RAM
      - 24/03/11 10:00 vpaEdge1 1.0 2.0
      - 24/03/11 10:01 vpaEdge1 1.1 2.3
      - ... (more real-time data points)
      // Compute the new resources for each pod:
      $rec_{CPU,RAM} \leftarrow$ **Recommender**($historyDict$)
      // Update the new resource to FL pod:
      $req_{CPU,RAM}, lim_{CPU,RAM} \leftarrow$ **Updater**($rec_{CPU,RAM}$)
    end for
    time.sleep(60s)
  end while
end procedure
procedure RECOMMENDER($historyDict$)
  Initialize parameter $target = 0.9$
  size = len($historyDict$)
  $historyList \leftarrow historyDict${key=CPU,key=RAM}
  $sortHistoryList \leftarrow$ sort($historyList$, ascending = True)
  $targetEstimate_{CPU,RAM} \leftarrow sortHistoryList$[size*$target$]
  $targetEstimate_{CPU,RAM} += targetEstimate_{CPU,RAM} * SM$
  // Set $minResource$ in the absence of historical data:
  $minResources_{CPU,RAM} \leftarrow min_{CPU}, min_{RAM}$
  $rec_{CPU,RAM} \leftarrow$ max($minResources$,$targetEstimate$)
  return $rec_{CPU,RAM}$
end procedure
procedure UPDATER(rec_{CPU}, rec_{RAM})
  Initialize parameter $threshold = 0.1$
  $req_{CPU,RAM} \leftarrow getCurReqValue(FLpod)$
  $diff_{CPU,RAM} \leftarrow \frac{|req_{CPU,RAM} - rec_{CPU,RAM}|}{req_{CPU,RAM}}$
  if $diff_{CPU} > threshold$ or $diff_{RAM} > threshold$ **then**
    $req_{CPU,RAM} \leftarrow rec_{CPU,RAM}$
    $lim_{CPU,RAM} \leftarrow rec_{CPU} * ratio_{CPU}, rec_{RAM} * ratio_{RAM}$
  end if
  return $req_{CPU,RAM}, lim_{CPU,RAM}$
end procedure

4.2. VPA algorithm

Algorithm 1 explains how VPA works to give the recommended resources for FL pods based on historical and real-time resource utilization data. All the parameter descriptions are listed in Table 2.

Considering the input parameters, we set the minimum recommended CPU and RAM ($minCPU$, $minRAM$) required for an FL pod at 1 CPU core and 2 GB memory, respectively. These two parameters are also used as the initial VPA recommendation for the Recommender module. Safety Margin (SM) is typically used to introduce a margin of safety or flexibility in the recommended resource values. Additionally, we also mentioned the ratios between the initial resource limits and the initial resource requests ($ratio_{CPU}$, $ratio_{RAM}$). These ratios determine the relationship between the initial resource limits set for a pod and the initial resource requests made by that pod. The main output of the VPA algorithm is to provide appropriate resource recommendations for CPU and RAM ($rec_{CPU,RAM}$) to FL pods.

At the beginning of the process, VPA retrieves historical and real-time resource usage data from Prometheus and Metric Server. Within the loop, each FL pod's resource utilization history is obtained, and the **Recommender()** function computes recommended CPU and RAM resources. The **Updater()** function then updates the existing FL pod's resource requests based on the differences between the current requests and the recommended resources, with an adjustable threshold to control updates. After processing each pod in the list and updating its resource allocations, VPA completes the for loop. VPA set a 60-second pause before starting the next iteration of the while loop. This is the default time interval for VPA operation such as allows periodic

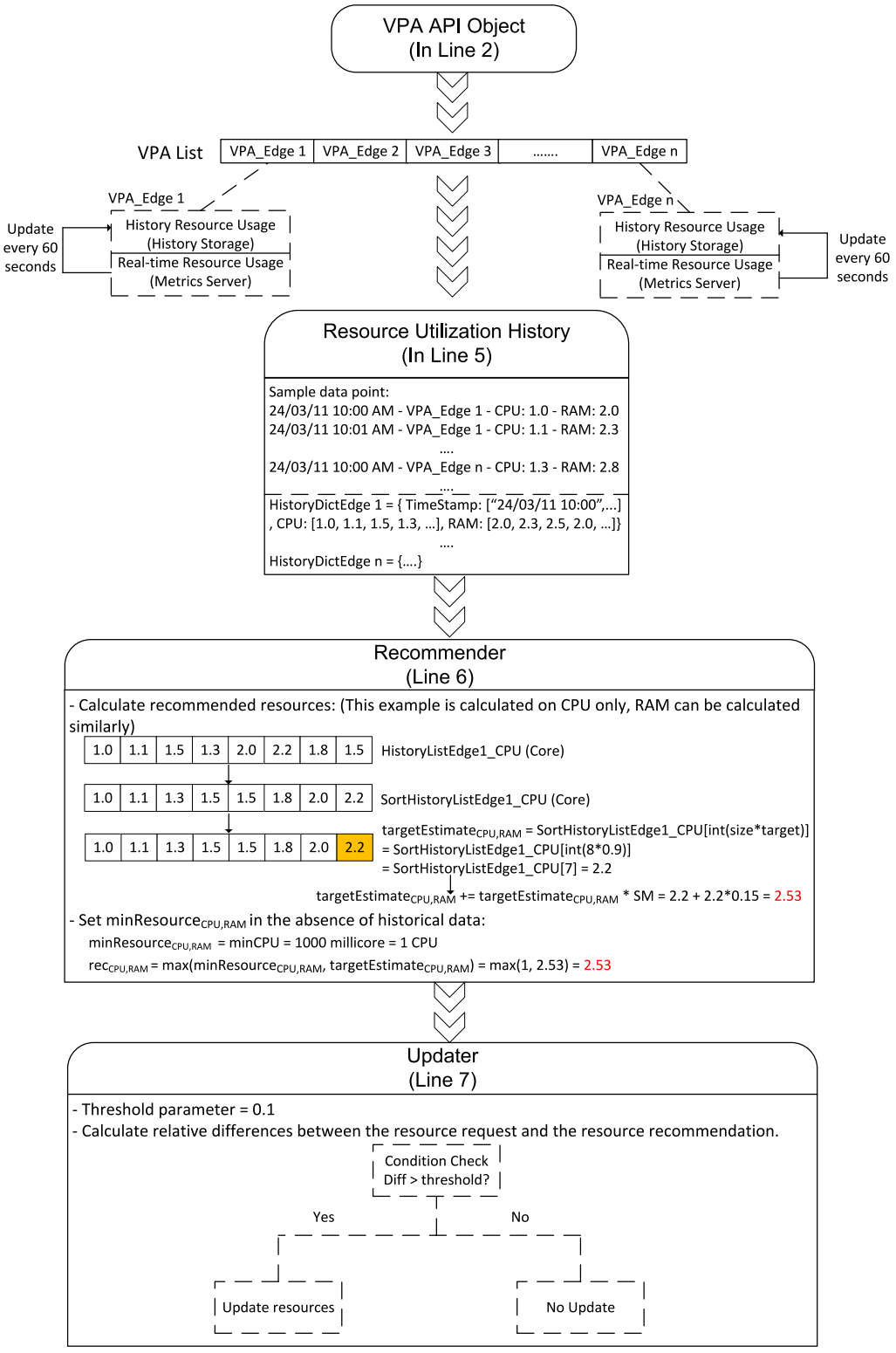


Fig. 2. VPA Algorithm Flow Chart.

resource adjustments and monitoring, it can be changed based on each application.

The **Recommender()** function, we first initialize a target percentile ($target$) set to 0.9, this function extracts CPU and RAM information from the historical data ($historyDict$) and sorts it in ascending order ($sortHistoryList$). The target estimate ($targetEstimate_{CPU,RAM}$) is then calculated by selecting the historical resource value at the specified target percentile, multiplied by SM. It acts as a multiplier to adjust the target estimate of recommended resources. The idea is to ensure that the recommended resources are not too close to the actual historical

maximum resources. In situations where no historical data is available, the function sets the minimum resources ($minResources_{CPU,RAM}$) to the minimum CPU and RAM values. The final recommendation returned by the function is the maximum value between the computed target estimate and the minimum resources.

The **Updater** function initializes a threshold parameter set to 0.1, representing a percentage difference threshold. It then retrieves the current resource requests of the FL pod ($req_{CPU,RAM}$). The function calculates the relative differences between the current and recommended values for both CPU and RAM ($diff_{CPU,RAM}$). If either the CPU or RAM difference exceeds the specified threshold, the function updates the FL pod's resource requests ($req_{CPU,RAM}$) to the recommended values ($rec_{CPU,RAM}$). Otherwise, it continues without making any changes. This approach ensures that the resource requests are adjusted only when the difference between the current and recommended values exceeds the defined threshold, preventing unnecessary adjustments in cases where the difference is within an acceptable range.

The algorithm's time complexity primarily depends on the number of FL pods and the size of their resource history lists, resulting in $O(m \times (k \log k))$ complexity, where m is the number of FL pods and k is the size of the history list for each pod.

Besides, Fig. 2 explains the workflow of the VPA algorithm, which automatically adjusts resource requests and limits for KubeEdge pods based on historical and real-time resource usage data. The process starts with VPA components running on multiple nodes, retrieving API objects and historical/real-time resource usage data for the pods they manage. A Recommender component then analyzes this data to generate recommended resource requests (e.g., CPU, RAM) for each Pod, taking into account factors like target percentiles and safety margins. An Updater component compares the recommendations to the current resource requests/limits and updates the Pod resources if the difference exceeds a configured threshold. This continuous monitoring and dynamic adjustment process helps manage resource usage across the KubeEdge cluster.

As illustrated in Fig. 1, the proposed VPA algorithm is designed to run primarily on the FL cloud node, which typically has sufficient computational resources to handle the resource monitoring and adjustment tasks. The FL cloud node monitors the resource status of the FL edge nodes via Metrics Server and decides on the VPA for each pod resource via VPA API object integrated into FL edge nodes. Since most of the VPA computation tasks are assigned to the cloud node, our proposed framework can work on various types of edge devices, including those with limited computational capabilities and resource constraints.

VPA, when deployed in a KubeEdge-based environment, can dynamically adjust resource allocation for individual pods to improve performance, making it more suitable for edge nodes. Moreover, the integration of VPA with KubeEdge's edge computing framework enables smooth coordination between edge and cloud resources. In particular, the scalability [39] and lightweight size [32] inherent in KubeEdge's platforms enhance VPA's adaptability, allowing it to accommodate varying workload demands. This approach ensures that the resource allocation process does not overload the edge devices, making the framework suitable for deployment in heterogeneous edge computing environments.

5. Performance evaluations

The objective of this experiment is to comprehensively evaluate the potential of VPA in efficient resource allocation for FL across diverse system heterogeneity, including Low, Medium, and High resource capacities. In addition to key performance metrics such as local training time, processing time, and convergence time, the evaluation includes examining the correspondence between the pod resource recommendations provided by VPA and the actual resources used in real-time scenarios under data heterogeneity and heterogeneous environments.

It is important to note that the traditional edge computing environment utilizes static resource allocation [2,16,21], and this paper is the first to propose applying a dynamic adjustment mechanism on an edge computing-based FL framework. The assessment compares VPA with diverse conditions of nonVPA, where nonVPA is configured to allocate a designated portion of node resources to each pod. For example, nonVPA-50% indicates that the pod will take 50% of CPU and RAM resources of the node, while the pod in nonVPA-100% will take all of the node resources.

In this experimental setup, we used the CIFAR-10 [40] dataset which is a collection of labeled images commonly used for training and testing machine learning algorithms in computer vision tasks. It consists of 60,000 32 × 32 color images in 10 different classes, with 6,000 images per class. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is split into 50,000 training images and 10,000 test images. CIFAR-10 is widely used as a benchmark dataset for image classification tasks due to its small size and complexity. It is particularly popular for evaluating the performance of convolutional neural networks (CNNs) and other deep learning architectures.

Table 3
Resource and data distributions setup for VPA/nonVPA on system heterogeneity.

VPA			Edge1	...	Edge5	Edge6	...	Edge26	Edge27	...	Edge30
Data samples			7000	7000	7000	7000	7000	7000	7000	7000	7000
Node resources		CPU (Core)	1	1	1	2	2	2	4	4	4
		RAM (GB)	4	4	4	4	4	4	8	8	8
Pod resources		CPU (Core)	–	–	–	–	–	–	–	–	–
		RAM (GB)	–	–	–	–	–	–	–	–	–
nonVPA			Edge1	...	Edge5	Edge6	...	Edge26	Edge27	...	Edge30
Data samples			7000	7000	7000	7000	7000	7000	7000	7000	7000
Node resources		CPU (Core)	1	1	1	2	2	2	4	4	4
		RAM (GB)	4	4	4	4	4	4	8	8	8
Pod resources	25% node CPU	CPU (Core)	0.25	0.25	0.25	0.5	0.5	0.5	1	1	1
		RAM (GB)	3	3	3	3	3	3	3	3	3
Pod resources	50% node CPU	CPU (Core)	0.5	0.5	0.5	1	1	1	2	2	2
		RAM (GB)	3	3	3	3	3	3	3	3	3
Pod resources	75% node CPU	CPU (Core)	0.75	0.75	0.75	1.5	1.5	1.5	3	3	3
		RAM (GB)	3	3	3	3	3	3	3	3	3
Pod resources	100% node CPU	CPU (Core)	1	1	1	2	2	2	4	4	4
		RAM (GB)	3	3	3	3	3	3	3	3	3

"–": The initial pod resources are left unassigned, allowing FL pods to use up to 100% of node resources when new recommendations are applied by VPA.

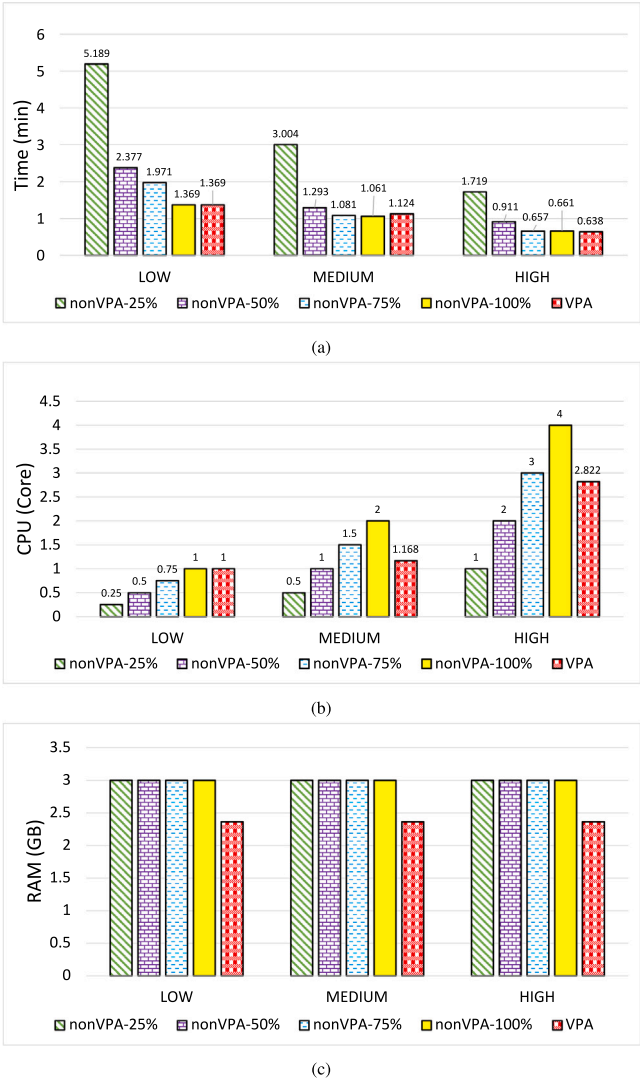


Fig. 3. (a) Local training time, (b) CPU usage, and (c) RAM usage on system heterogeneity.

The target accuracy for convergence time in our experiments is set to 0.72, representing a significant milestone in achieving acceptable model accuracy within a specified time frame. This accuracy threshold is carefully chosen to balance the trade-off between model performance and training time, ensuring that the FL model achieves satisfactory results while reducing the computational burden.

5.1. VPA on system heterogeneity

This subsection evaluates the effect of VPA on the system heterogeneity according to Table 3 summarizing the resource and data distribution for VPA and nonVPA scenarios. Both scenarios employ 30 edge nodes and utilize an identical set of training samples (7,000). In terms of node resources, the first five edge nodes (Edge 1 to Edge 5) are categorized as having low resources, equipped with 1 CPU core and 4 GB of RAM. The subsequent 21 nodes (Edge 6 to Edge 26) fall into the medium resources category, each featuring 2 CPU cores and 4 GB of RAM. Finally, the last four nodes (Edge 27 to Edge 30) are classified as high-resource nodes with 4 CPUs and 8 GB of RAM. For the VPA scenario, pod resources are allocated based on VPA recommendations, presented as "-". Conversely, in the nonVPA scenario, pod resources are set proportionally to the node resources, running from 100% to 25% in terms of CPU (Core).

Fig. 3(a) illustrates the local training times for both VPA and nonVPA scenarios across system heterogeneity. For low-resource edge nodes, as we allocate more CPU resources to the pods, ranging from 25% to 100% of the node resources, the local training time decreases from 5.189 min to 1.369 min. At 100% CPU allocation for both nonVPA (1 CPU) and the VPA approach, they achieve similar low local training times, approximately 1.369 min per round. This similarity occurs because VPA dynamically adjusts pod resources up to 100% CPU based on historical and real-time utilization, which will be shown in Fig. 3(b). In the case of medium-resource edge nodes, unlike the low-resource edge nodes, the local training time does not consistently decrease as we increase pod CPU resources from 75% to 100%. This indicates that allocating more than 75% CPU resources may not significantly improve performance. Similarly, for high-resource edge nodes, the local training time also does not consistently decrease with increasing pod CPU resources from 75% to 100%. These results indicate that VPA provides an efficient resource adjustment precisely as required, preventing overprovisioning. In other words, VPA fine-tunes resources based on historical utilization and ensures optimal performance without unnecessary resource allocation, as will be discussed in Fig. 3(b) and Fig. 3(c).

Fig. 3(b) presents a visualization of CPU pod resources for both VPA and nonVPA scenarios across system heterogeneity. VPA dynamically adjusted CPU resource allocation for all three edge node types (low, medium, and high) during the FL process based on the demands of the training workload. For example, in the case of low-resource edge nodes, VPA recommended and set the pod CPU resources to 1 CPU, utilizing the full 100% of the edge node's resources. This configuration ensured that low-resource nodes efficiently exploited their CPU capacity, enabling effective completion of training tasks. At medium-resource edge nodes, VPA recommended and adjusted CPU resources to 1.168 CPU. Similarly, for high-resource edge nodes, VPA recommended and adjusted CPU resources to 2.822 CPUs. We can observe that VPA in both medium and high-resource edge nodes achieved comparable performance to nonVPA-75% and nonVPA-100%, respectively. It is worth noting that VPA achieved this level of performance while utilizing fewer CPU resources for FL tasks compared to nonVPA setups (2 CPUs for nonVPA medium resources compared to 1.168 CPUs for VPA, and 4 CPUs for nonVPA high resources compared to 2.822 CPUs for VPA). This highlights that VPA efficiently adjusted the suitable CPU resources for each type of edge node during the FL training, aiming to improve the performance of the training tasks such as local training time, and manage resources efficiently.

Fig. 3(c) represents memory resources for both VPA and nonVPA scenarios across system heterogeneity. Based on the provided results, it appears that the memory resources recommended by VPA among the low, medium, and high-edge node resources are not significantly different. This is due to the setting of homogeneous distribution among edge nodes (7,000 training samples). Compared with the nonVPA approach, VPA only needs 2.3 GB instead of 3 GB to run the FL tasks without OOM issues. VPA is effectively recommending memory resources (GB) based on the specific edge node types (low, medium, high) and their available data training samples. The VPA's recommendations align with the capacity of the edge nodes, ensuring effectively utilization of memory resources during the training of FL tasks. Overall, the results indicate that VPA plays a crucial role in managing memory resources for FL tasks on edge nodes, leading to improved performance and efficient resource utilization.

Fig. 4 demonstrates the efficiency of VPA in enhancing processing times during each round. Typically, there is a reduction in processing time as we increase resource allocation from 25% to 100% of node resources within nonVPA approaches. This trend underscores that dedicating more resources to pods results in faster processing times. When comparing VPA and nonVPA approaches, VPA consistently achieves lower processing times in most cases. This suggests that VPA's dynamic

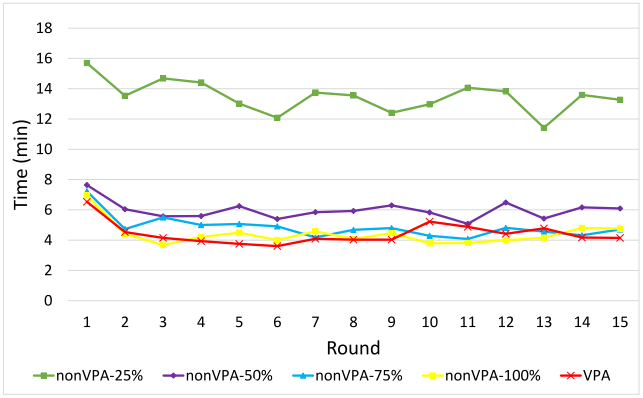


Fig. 4. Processing time on each round.

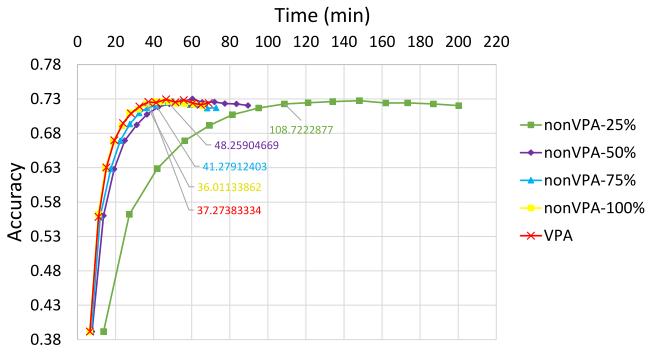


Fig. 5. Time to reach 0.72 accuracy between VPA and nonVPA.

Table 4
Experimental setup for VPA on data heterogeneity.

Edge node	Samples	CPU (Cores)	Memory (GB)
Edge 1	2000	2	4
Edge 2	10,000	2	4
Edge 3	25,000	2	4

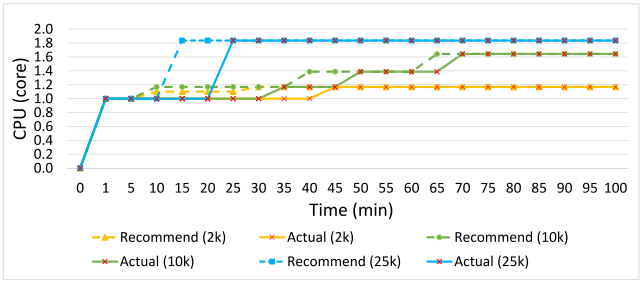
resource adjustments based on historical and real-time utilization can enhance resource allocation and reduce overall processing time.

Fig. 5 provides a clear illustration of VPA's effectiveness in improving convergence times during each round. Typically, convergence times decrease as the available resources for pods increase. Within nonVPA configurations, allocating more node resources results in quicker convergence, emphasizing the impact of resource availability on convergence speed. Interestingly, the convergence time for VPA is similar to the case where 100% of node resources are used without VPA. Specifically, deploying VPA can significantly improve convergence time (37.27 min) compared to nonVPA-25% (108.72 min) with the same level of data distribution. For FL on 50% and 75% node utilization, using nonVPA configurations already leads to relatively fast convergence (48,25 and 41.27 min). However, VPA can still offer advantages in optimizing resource usage and ensuring consistent convergence, achieving it in 37.27 min. In summary, the nonVPA approach just only gives fixed resources during the training, which may not always be the best fit. For example, if we give too many resources to a client with very little data, it is like giving them more tools than they actually need. And if we give too few resources to a client with lots of training data, it will slow down the training process. On the other hand, VPA reduces processing time in each round through real-time monitoring and dynamic adjustment of pod resources every minute, based on that, the convergence time will be also decreased. In edge computing, resource efficiency utilization is a critical factor for diverse resource edge node environments. That is why VPA is a better solution to speed up the training process in each round. It constantly checks how much work needs to be done and adjusts the resources accordingly every minute. This helps make the training time much faster, and we reach our target accuracy sooner.

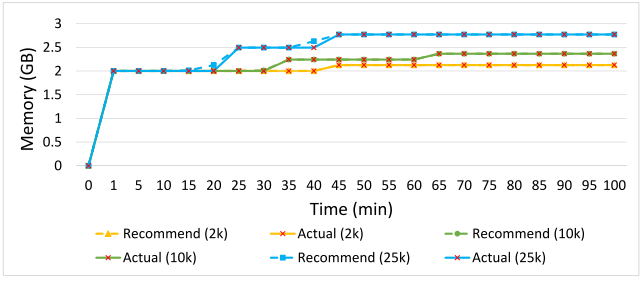
5.2. VPA on data heterogeneity

This subsection focuses on evaluating VPA's resource recommendations during FL training in data heterogeneity distribution scenarios. As summarized in Table 4, three different edge nodes are allocated different sizes of training samples: 2000, 10000, and 25000, respectively, each with 2 CPU cores and 4 GB RAM.

Fig. 6(a) presents the recommendations and actual pod resource usage made by VPA for CPU resources during FL tasks on three different edge nodes. For the initial recommendation stage, VPA does not provide any recommendations (0 CPU cores) for all three nodes. At



(a)



(b)

Fig. 6. (a) VPA recommendation and (b) actual pod resources (CPU/Memory) applied in real-time for data heterogeneity.

1 min, VPA recommends allocating 1 CPU core for all nodes, indicating it starts with the default resource allocation. Actual CPU usage for all three edge nodes closely matches the recommendations during this initial stage. For the adaptation to the workload stage at 10-25 min, as time goes, VPA begins to adjust its CPU recommendations based on the evolving workload. VPA recommends around 1.1 CPU cores to the 2k node, reflecting a slight increase to meet the growing workload. For the 10k node, VPA recommends around 1.17 CPU cores, indicating a moderate increase in resources to accommodate the workload. We can see that VPA recommends 1.84 CPU cores for the 25k node due to the heavier workload. In the last 30 to 100 min, after the initial adjustment and adaptation to the workload period, a steady-state stage is reached where both recommended and actual resource usage stabilizes. At this stage, the workload demands successfully match the CPU recommendations by VPA. This demonstrates VPA adapts its CPU recommendations based on the number of training samples, ensuring that resource allocation is proportional to workload.

Fig. 6(b) provides a similar illustration for VPA's memory resource recommendations and actual pod resource usage during FL tasks. At

Table 5
Resource and data distributions setup for VPA/nonVPA on heterogeneous environments.

VPA	Data samples	Node resources		Pod resources	
		CPU	RAM	CPU	RAM
Edge 1	5000	2	4	0.1	4
Edge 2	2000	2	4	0.2	1
Edge 3	1000	2	4	1	3
Edge 4	10 000	2	4	3	6
Edge 5,6	25 000	2	4	2	4
Edge 7	15 000	2	4	0.5	1
Edge 8	20 000	2	4	1.5	2
Edge 9	500	2	4	1	3
Edge 10	1000	1	4	1	3
Edge 11	2000	1	4	0.5	3
Edge 12	3000	1	4	6	6
Edge 13	4000	1	4	1	4
Edge 14	**5000**	**1**	**4**	–	–
Edge 15	6000	4	8	1	2
Edge 16	7000	4	8	2	4
Edge 17	8000	4	8	–	–
Edge 18	**9000**	**4**	**8**	**0.5**	**1**
Edge 19	10 000	2	4	0.2	2
Edge 20	11 000	2	4	0.4	4
Edge 21	12 000	2	4	4	8
Edge 22	13 000	2	4	8	12
Edge 23	14 000	2	4	1	1
Edge 24	15 000	2	4	2	4
Edge 25	16 000	2	4	0.1	1
Edge 26	17 000	2	4	2	4
Edge 27	18 000	2	4	–	–
Edge 28	**19 000**	**2**	**4**	**1.5**	**2**
Edge 29, 30	19 000	2	4	1.5	2

nonVPA	Data samples	Node resources		Pod resources		Pod resources		Pod resources		Pod resources	
		CPU	RAM	25% node CPU		50% node CPU		75% node CPU		100% node CPU	
				CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM
Edge 1	5000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 2	2000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 3	1000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 4	10 000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 5,6	25 000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 7	15 000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 8	20 000	2	4	0.5	3	1	3	1.5	3	2	3
Edge 9	500	2	4	0.5	3	1	3	1.5	3	2	3
Edge 10	1000	1	4	0.25	3	0.5	3	0.75	3	1	3
Edge 11	2000	1	4	0.25	3	0.5	3	0.75	3	1	3
Edge 12	3000	1	4	0.25	3	0.5	3	0.75	3	1	3
Edge 13	4000	1	4	0.25	3	0.5	3	0.75	3	1	3
Edge 14	**5000**	**1**	**4**	**0.25**	**3**	**0.5**	**3**	**0.75**	**3**	**1**	**3**
Edge 15	6000	4	8	1	3	2	3	3	3	4	3
Edge 16	7000	4	8	1	3	2	3	3	3	4	3
Edge 17	8000	4	8	1	3	2	3	3	3	4	3
Edge 18	**9000**	**4**	**8**	**1**	**3**	**2**	**3**	**3**	**3**	**4**	**3**
Edge 19,. . . ,27	10k,. . . ,18k	2	4	0.5	3	1	3	1.5	3	2	3
Edge 28	**19 000**	**2**	**4**	**0.5**	**3**	**1**	**3**	**1.5**	**3**	**2**	**3**
Edge 29,30	19 000	2	4	0.5	3	1	3	1.5	3	2	3

"–": The initial pod resources are left unassigned, allowing FL pods to use up to 100% of node resources when new recommendations are applied by VPA.

1 min, VPA recommends allocating 2 GB of memory for all nodes, suggesting an initial memory resource allocation. For the next 30 min (15-45 min), VPA begins to make slight adjustments in memory resource recommendations based on the increasing workload. The steady-state stage starts when all the new resource recommendations from VPA and the actual pod resource usage closely match each other. For instance, actual pod resource usage stays still at 2.13 GB memory at 45 min for the 2k node, 2.36 GB memory at 65 min for the 10k node, and 2.77 GB memory at 45 min for the 25k node. This result demonstrates how VPA effectively gives its memory resource recommendations to match changing workloads across data heterogeneity scenarios. In summary, the data distribution affected the resource requirements during FL training. Larger datasets generally require more CPU and memory resources compared to smaller datasets. Besides, VPA provides CPU and memory resource recommendations for the pods during the training of FL on different types of data distributions with varying sample sizes

(2k, 10k, and 25k). The VPA recommendations were dynamic and adjusted over time to optimize resource utilization based on the specific workload.

5.3. VPA on heterogeneous environments

This subsection evaluates the effectiveness of VPA in a heterogeneous environment composed of system heterogeneity and data heterogeneity across 30 edge nodes. Table 5 provides an overview of the detailed setup for both nonVPA and VPA scenarios.

In the VPA setup, we first observe the distribution of data samples across the edge nodes. These data distribution represents the varying workloads on each node and their diversity underscores the real-world, dynamic nature of this heterogeneous environment. In terms of node resources, medium resource nodes (Edge 1 to Edge 9 and Edge 19 to Edge 30) are provided with 2 CPU cores and 4 GB of RAM. Edge 10 to Edge 14 fall into the low resources category, each featuring 2 CPU cores and 4 GB of RAM. The last four nodes (Edge 15 to Edge 18) are classified as high-resource nodes with 4 CPU cores and 8 GB of RAM. Especially, the pod resources column is particularly noteworthy. It represents the initial resource allocations configured by random users. However, these pod resource allocations will be updated continuously by VPA in response to the actual workloads encountered during the FL training process. On the other hand, in the nonVPA setup, we again observe the distribution of data samples across the 30 edge nodes, and the different node resources assigned to each node are categorized as low, medium, and high edge node resources. These four columns in terms of pod resources illustrate how CPU and RAM are distributed to pods on each edge node at various resource percentages (25%, 50%, 75%, and 100%) of their respective node's resources. Notably, these pod resource allocations are fixed during the FL training. Finally, 3 GB of RAM is used for all four node resource percentages.

The main objective is to evaluate the impact of VPA on FL performance, specifically in terms of local training time across multiple rounds. In this selection, we designated Edge 14 as the representative for low-resource edge nodes, Edge 28 for medium-resource edge nodes, and Edge 17 for high-resource edge nodes.

Fig. 7(a) presents the average local training times (in minutes) across 15 rounds in FL for three different types of edge nodes: low, medium, and high resources. The results are compared across five different setups: nonVPA-25%, nonVPA-50%, nonVPA-75%, nonVPA-100%, and VPA setup. While comparing low and high-resource edge nodes, we observe a significant difference in local training time across all setups. This is expected, as higher-resource nodes have more computational power available for training, resulting in faster training times. However, in some cases, certain setups perform similarly in terms of local training times. For example, in a low-resource node, setups like nonVPA-100% and VPA take about 1.1 min per round. In a medium-resource node, setups like nonVPA-75%, nonVPA-100%, and VPA take about 2.5 min. Finally, in a high-resource node, nonVPA-75% and VPA both take about 1.8 min. This indicates that if an edge node already has enough CPU resources for the given training dataset size, adding more CPU resources to the pod will not basically lead to a reduction in local training time.

In Figs. 7(b) and 7(c), we observe how VPA recommends and adjusts pod resources (CPU/Memory) under a heterogeneous environment across different types of edge nodes in terms of hardware resources, training samples, and initial user-defined pod configurations.

Regarding the low-resource edge node, in the context of CPU resources, VPA consistently suggests utilizing 1 CPU core right from the start and maintains this allocation throughout the training. This consistency arises because 1 CPU core represents the maximum CPU resource available on this particular edge node (edge 14). The actual CPU usage on Edge 14 aligns perfectly with these recommendations throughout the entire process. In terms of memory resources, VPA's memory recommendations begin at 0 GB and rise to 2 GB within the first minute. Subsequently, they increase slightly to 2.13 GB around at 40 min, and this level is maintained for the rest of the training. However, the actual memory usage remains fixed at 2 GB. This is because the margin between 2 GB and 2.13 GB, indicated by the *threshold*, is below 0.1, making it ineligible for implementing the new memory value for the pod, where the *threshold* is defined as Algorithm 1-line 19.

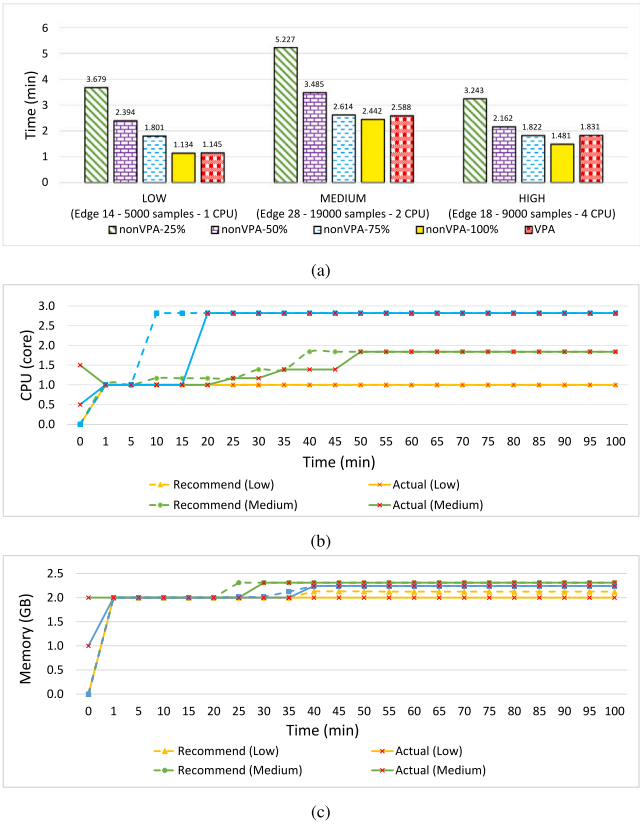


Fig. 7. (a) Local training time, (b) VPA recommendation, and (c) actual pod resources (CPU/Memory) under heterogeneous environment.

Concerning the medium-resource edge node, initially, the CPU resource usage is set at 1.5 CPU cores and 2 GB memory, assuming that users have manual control over pod resources for FL tasks. However, within the first minute, the actual CPU resource drops to 1 CPU core due to the *threshold* exceeding 0.1 between 1 CPU core and 1.5 CPU cores. As time goes, especially after 10 min, VPA begins to adapt its CPU and memory recommendations in response to the increasing workload, changing from 1.17 CPU cores to 1.84 CPU cores and from 2 GB memory to 2.31 GB memory. The actual CPU usage closely aligns with these recommendations during this process.

Similarly, for the high-resources edge node, the initial configuration of 0.5 CPU cores and 1 GB RAM is increased to 2.82 CPU cores and 2.24 GB RAM according to VPA's recommendation at 10 min.

In real-world scenarios, the initial memory allocation set by users for specific tasks might sometimes prove insufficient to ensure smooth application execution. This situation is often referred to as an OOM problem. In the context of FL tasks, when a pod encounters OOM issues, the FL tasks within that pod become pending and cannot proceed with training. However, by implementing VPA, we can eliminate concerns about OOM problems and the need to precisely determine resource allocations for FL tasks on each unique node within heterogeneous environments. VPA handles this regard by dynamically adjusting pod resources based on historical/real-time usage patterns and the currently available resources on the node, ensuring efficient and uninterrupted FL task execution.

5.4. Summary of lessons

This section provides a summary of the key lessons learned regarding evaluating the VPA performance across a range of factors, such as VPA's impact on system heterogeneity and its usefulness to data heterogeneity distribution as well as heterogeneous environments.

- **VPA's Dynamic Resource Allocation:** The core idea behind deploying VPA is to reduce the training time in each round through real-time monitoring and dynamic pod resource adjustments every minute, thereby optimizing convergence time and local training time for FL applications. Based on these characteristics, VPA efficiently improves the training process in heterogeneous environments where each edge node has different hardware resources, training samples, and user-defined pod configurations
- **Improve Performance without Overprovisioning:** The nonVPA approach just only gives the fixed resources during the training, which may not always be the best fit. For example, if we give too many resources to a client with very little data, it is like giving them more tools than they actually need. And if we give too few resources to a client with lots of training data, it will slow down the training process. VPA outperforms nonVPA in dynamically adjusting resources using historical and real-time utilization data, ensuring effective performance while preventing overprovisioning.
- **Preventing Critical Issues:** In real-world scenarios, the initial memory allocation set by users for specific tasks might sometimes prove insufficient to ensure smooth application execution. In particular, employing VPA eliminates concerns about OOM issues, which is a critical problem when running FL applications, by dynamically adjusting pod resources based on historical/real-time usage patterns and the currently available resources on the node, ensuring efficient and uninterrupted FL task execution.

6. Discussion

This section discusses the limitations of our EFL framework, focusing on the impact of diverse factors in resource allocation and the management overhead associated with VPA. Additionally, we will delve into practical applications of VPA in real-world scenarios.

To further expand the understanding of resource allocation dynamics within edge computing environments, it's essential to acknowledge the potential influence of various factors beyond our current scope (CPU, memory). These factors include networking, file system operations, other applications, or parallel running applications, which can also impact resource allocation and system performance. Exploring the integration of these factors into our framework could be an avenue for future research. By incorporating additional metrics and considerations, we can develop more comprehensive resource management strategies that address the diverse requirements and dynamics of edge computing environments.

Next, while VPA offers dynamic resource allocation capabilities, there is a built-in overhead associated with its management and monitoring which is called Metrics Server. The Metrics Server continuously monitors the resource utilization of pods, which involves collecting and analyzing metrics such as CPU and memory usage. This monitoring process itself can consume resources, especially in resource-constrained edge computing environments with a large number of pods and frequent monitoring intervals [41]. To mitigate this limitation, our framework already involved adaptive monitoring intervals, where the frequency of monitoring is adjusted based on resource utilization patterns and user-defined, reducing unnecessary monitoring overhead.

In the context of edge computing-based FL applications, some practical use cases can demonstrate the integration of VPA into real-world scenarios. For instance, within autonomous vehicle systems, the incorporation of VPA within the edge computing infrastructure allows for dynamic resource allocation tailored to FL tasks. In blocked traffic conditions or crowded populated areas, VPA can dynamically allocate additional CPU and memory resources to handle the heightened demand for model training, thereby enhancing the performance and responsiveness of autonomous vehicle systems [23]. Similarly, in Closed-Circuit Television (CCTV) systems, where each camera may have varying hardware resources and can run different applications (multi-task problems) [42], VPA can address these complexities by dynamically allocating resources based on the specific needs of each camera's tasks. For example, VPA can adjust CPU and memory allocations for cameras running video analytics tasks, ensuring efficient utilization of edge resources and optimizing system performance, while also accommodating other concurrent tasks on the same edge infrastructure.

7. Conclusion

In this paper, we introduced an EFL framework that leverages the VPA with Kubernetes to enhance the performance of FL applications. By integrating VPA into FL, we achieved elastic scaling of resources, ensuring optimal model training and earlier achievement of target accuracy. Our experiments demonstrated that this approach significantly enhances FL efficiency, in edge computing environments, opening doors to more efficient and scalable AI applications while preserving data privacy, eviction and recreation, leveraging the deployment rolling update mechanism.

Our future work will focus on verifying the feasibility and effectiveness of the proposed framework in multi-tenant environments. Besides, we will investigate the challenges and opportunities associated with managing resources for multi-task FL applications together, ensuring effective isolation, and resource adjustment among diverse users in edge computing scenarios.

CRediT authorship contribution statement

Khanh Quan Pham: Methodology, Investigation, Resources, Software, Validation, Visualization, Writing – original draft. **Taehong Kim:** Conceptualization, Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research was partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2022R1I1A3072355, 50%) and Innovative Human Resource Development for Local Intellectualization program through the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government(MSIT) (IITP-2024-2020-0-01462, 50%).

References

[1] G. Drainakis, K.V. Katsaros, P. Pantazopoulos, V. Sourlas, A. Amditis, Federated vs. Centralized machine learning under privacy-elastic users: A comparative analysis, in: IEEE 19th International Symposium on Network Computing and Applications, (NCA), 2020, pp. 1–8, http://dx.doi.org/10.1109/NCA51143.2020.9306745.

[2] L.H. Phuc, L.A. Phan, T. Kim, Traffic-aware horizontal pod autoscaler in kubernetes-based edge computing infrastructure, IEEE Access 10 (2022) 18966–18977, http://dx.doi.org/10.1109/ACCESS.2022.3150867.

[3] S.H. Kim, T. Kim, Local scheduling in KubeEdge-based edge computing environment, Sensors 23 (2023) http://dx.doi.org/10.3390/s23031522.

[4] T.X. Tran, A. Hajisami, P. Pandey, D. Pompili, Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges, IEEE Commun. Mag. 55 (2017) 54–61, http://dx.doi.org/10.1109/MCOM.2017.1600863.

[5] A.M. Abdelmoniem, C.Y. Ho, P. Papageorgiou, M. Canini, A comprehensive empirical study of heterogeneity in federated learning, IEEE Internet Things J. 10 (2023) 14071–14083, http://dx.doi.org/10.1109/JIOT.2023.3250275.

[6] S. Pal, N.Z. Jhanjhi, A.S. Abdulbaqi, D. Akila, A.A. Almazroi, F.S. Alsubaei, A hybrid edge-cloud system for networking service components optimization using the internet of things, Electronics (Switzerland) 12 (2023) http://dx.doi.org/10.3390/electronics12030649.

[7] P.K. Quan, M. Kundroo, T. Kim, Experimental evaluation and analysis of federated learning in edge computing environments, IEEE Access 11 (2023) 33628–33639, http://dx.doi.org/10.1109/ACCESS.2023.3262945.

[8] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-IID data, 2018, http://dx.doi.org/10.48550/arXiv.1806.00582, http://arxiv.org/abs/1806.00582.

[9] A. Imteaj, U. Thakker, S. Wang, J. Li, M.H. Amini, Federated learning for resource-constrained IoT devices: Panoramas and state-of-the-art, 2020, URL http://arxiv.org/abs/2002.10610.

[10] M.A. Jan, M. Zakarya, M. Khan, S. Mastorakis, V.G. Menon, V. Balasubramanian, A.U. Rehman, An AI-enabled lightweight data fusion and load optimization approach for internet of things, Future Gener. Comput. Syst. 122 (2021) 40–51, http://dx.doi.org/10.1016/j.future.2021.03.020, URL https://www.sciencedirect.com/science/article/pii/S0167739X21001011.

[11] D.J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K.H. Li, T. Parcollet, P.P.B. de Gusmão, N.D. Lane, Flower: A friendly federated learning research framework, 2020, URL http://arxiv.org/abs/2007.14390.

[12] L.A. Haibeh, M.C. Yagoub, A. Jarray, A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches, IEEE Access 10 (2022) 27591–27610, http://dx.doi.org/10.1109/ACCESS.2022.3152787.

[13] X. Zhang, S. Debroy, Resource management in mobile edge computing: A comprehensive survey, ACM Comput. Surv. 55 (2023) http://dx.doi.org/10.1145/3589639.

[14] L.G.F. da Silva, D.F. Sadok, P.T. Endo, Resource optimizing federated learning for use with IoT: A systematic review, J. Parallel Distrib. Comput. 175 (2023) 92–108, http://dx.doi.org/10.1016/j.jpdc.2023.01.006, URL https://www.sciencedirect.com/science/article/pii/S0743731523000060.

[15] T.P.P. da Silva, A.R. Neto, T.V. Batista, F.C. Delicato, P.F. Pires, F.A. Lopes, Online machine learning for auto-scaling processing services in the edge computing environment, SSRN Electron. J. (2022) http://dx.doi.org/10.2139/ssrn.4160595.

[16] L.H. Phuc, M. Kundroo, D.H. Park, S. Kim, T. Kim, Node-based horizontal pod autoscaler in KubeEdge-based edge computing infrastructure, IEEE Access 10 (2022) 134417–134426, http://dx.doi.org/10.1109/ACCESS.2022.3232131.

[17] S. Trindade, L.F. Bittencourt, N.L. da Fonseca, Resource management at the network edge for federated learning, Digit. Commun. Netw. (2022) http://dx.doi.org/10.1016/j.dcan.2022.10.015, URL https://www.sciencedirect.com/science/article/pii/S2352864822002243.

[18] D. Yang, W. Zhang, Q. Ye, C. Zhang, N. Zhang, C. Huang, H. Zhang, X. Shen, DetFed: Dynamic resource scheduling for deterministic federated learning over time-sensitive networks, IEEE Trans. Mob. Comput. (2023) http://dx.doi.org/10.1109/TMC.2023.3303017.

[19] A. Salh, R. Ngah, L. Audah, K.S. Kim, Q. Abdullah, Y.M. Al-Moliki, K.A. Aljaloud, H.N. Talib, Energy-efficient federated learning with resource allocation for green IoT edge intelligence in B5G, IEEE Access 11 (2023) 16353–16367, http://dx.doi.org/10.1109/ACCESS.2023.3244099.

[20] H. Xiao, J. Zhao, Q. Pei, J. Feng, L. Liu, W. Shi, Vehicle selection and resource optimization for federated learning in vehicular edge computing, IEEE Trans. Intell. Transp. Syst. 23 (2022) 11073–11087, http://dx.doi.org/10.1109/TITS.2021.3099597.

[21] F. Nikolaidis, M. Symeonides, D. Trihinas, Towards efficient resource allocation for federated learning in virtualized managed environments, Future Internet 15 (2023) http://dx.doi.org/10.3390/fi15080261.

[22] J. Dogani, F. Khunjush, Proactive auto-scaling technique for web applications in container-based edge computing using federated learning model, J. Parallel Distrib. Comput. 187 (2024) 104837, http://dx.doi.org/10.1016/j.jpdc.2024.104837, URL https://www.sciencedirect.com/science/article/pii/S0743731524000017.

[23] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H.V. Poor, Federated learning for internet of things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (2021) 1622–1658, http://dx.doi.org/10.1109/COMST.2021.3075439.

[24] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou, A survey of convolutional neural networks: Analysis, applications, and prospects, IEEE Trans. Neural Netw. Learn. Syst. 33 (12) (2022) 6999–7019, http://dx.doi.org/10.1109/TNNLS.2021.3084827.

[25] G.V. Houdt, C. Mosquera, G. Nápoles, A review on the long short-term memory model, Artif. Intell. Rev. 53 (2020) 5929–5955, http://dx.doi.org/10.1007/s10462-020-09838-1.

[26] Y. Zhou, Y. Qing, J. Lv, Communication-efficient federated learning with compensated overlap-FedAvg, 2020, URL http://arxiv.org/abs/2012.06706.

[27] B. Magableh, M. Almiani, A self healing microservices architecture: A case study in docker swarm cluster, Adv. Intell. Syst. Comput. 926 (2020) 846–858, http://dx.doi.org/10.1007/978-3-030-15032-7_71.

[28] Q.M. Nguyen, L.A. Phan, T. Kim, Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy, Sensors 22 (2022) http://dx.doi.org/10.3390/s22082869.

[29] P. Saha, A. Beltre, M. Govindaraju, Exploring the fairness and resource distribution in an apache mesos environment, 2019, http://dx.doi.org/10.1109/CLOUD.2018.00061, http://arxiv.org/abs/1905.08388.

[30] C. Carrin, Kubernetes scheduling: Taxonomy, ongoing issues and challenges, ACM Comput. Surv. 55 (7) (2022) 1–37.

[31] C. Mahler, Kubernetes on the edge: getting started with KubeEdge and kubernetes for edge computing, 2022, [Online]. Available: https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing/. (Accessed 24 March 2024).

[32] J.D. Sanil Kumar D. (Huawei), KubeEdge, a kubernetes native edge computing framework, 2019, [Online]. Available: https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro/. Accessed 24 March 2024.

[33] T. Goethals, F.D. Turck, B. Volckaert, Extending kubernetes clusters to low-resource edge devices using virtual kubelets, IEEE Trans. Cloud Comput. 10 (2022) 2623–2636, http://dx.doi.org/10.1109/TCC.2020.3033807.

[34] Y. Xiong, Y. Sun, L. Xing, Y. Huang, Extend cloud to edge with KubeEdge, in: Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018, Institute of Electrical and Electronics Engineers Inc., 2018, pp. 373–377, http://dx.doi.org/10.1109/SEC.2018.00048.

[35] M. Wang, D. Zhang, B. Wu, A cluster autoscaler based on multiple node types in kubernetes, in: 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), vol. 1, 2020, pp. 575–579, http://dx.doi.org/10.1109/ITNEC48623.2020.9084706.

[36] T.T. Nguyen, Y.J. Yeom, T. Kim, D.H. Park, S. Kim, Horizontal pod autoscaling in kubernetes for elastic container orchestration, Sensors (Switzerland) 20 (2020) 1–18, http://dx.doi.org/10.3390/s20164621.

[37] M. Niazi, S. Abbas, A.H. Soliman, T. Alyas, S. Asif, T. Faiz, Vertical pod autoscaling in kubernetes for elastic container collaborative framework, Comput. Mater. Contin. 74 (2023) 591–606, http://dx.doi.org/10.32604/cmc.2023.032474.

[38] Prometheus, 2022, [Online]. Available: https://prometheus.io/. (Accessed 24 March 2024).

[39] W. Xu, Test report on KubeEdge's support for 100,000 edge nodes, 2022, [Online]. Available: https://kubeedge.io/zh/blog/scalability-test-report/. (Accessed 24 March 2024).

[40] B. Recht, R. Roelofs, L. Schmidt, V. Shankar, Do CIFAR-10 classifiers generalize to CIFAR-10?, 2018, URL http://arxiv.org/abs/1806.00451.

[41] Z. Wang, M. Goudarzi, J. Aryal, R. Buyya, Container orchestration in edge and fog computing environments for real-time IoT applications, 2022, URL http://arxiv.org/abs/2203.05161.

[42] K. Doshi, Y. Yilmaz, Multi-task learning for video surveillance with limited data, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, (CVPRW), 2022, pp. 3888–3898, http://dx.doi.org/10.1109/CVPRW56347.2022.00434.

Pham Khanh Quan received the B.S. degree in Electronics and Telecommunications Engineering from Ton Duc Thang University, Ho Chi Minh City, Vietnam in 2020, where he was working as a student research assistant of Wireless Communications Research Group, under the supervision of Dr. Phuong T. Tran. He received his M.S degree in Information and Communication Engineering at Chungbuk National University (CBNU) in 2023. He is currently pursuing his Ph.D. degree in Information and Communication Engineering at Chungbuk National University, South Korea. His major interests include edge computing, edge AI, federated learning, IoT applications, and container orchestration.

Taehong Kim received his B.S. degree in computer science from Ajou University, Republic of Korea, in 2005, and his M.S. degree in information and communication engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2007. He received his Ph.D. degree in computer science from KAIST in 2012. He worked as a Research Staff Member at Samsung Advanced Institute of Technology (SAIT) and Samsung DMC R&D Center from 2012 to 2014. He also worked as a Senior Researcher at the Electronics and Telecommunications Research Institute (ETRI), Republic of Korea, from 2014 to 2016. Since 2016, he has been an Associate Professor with the School of Information and Communication Engineering, Chungbuk National University, Republic of Korea. He has been an Associate Editor of IEEE Access since 2020. His research interests include edge computing, container orchestration, Internet of Things, and federated learning.