IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 December 2022, accepted 22 December 2022, date of publication 26 December 2022,
date of current version 29 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3232131

RESEARCH ARTICLE

Node-Based Horizontal Pod Autoscaler in KubeEdge-Based Edge Computing Infrastructure

LE HOANG PHUC[1], MAJID KUNDROO[1], DAE-HEON PARK[2], SEHAN KIM[2], AND TAEHONG KIM[1], (Senior Member, IEEE)

[1]School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, South Korea
[2]Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea

Corresponding author: Taehong Kim (taehongkim@cbnu.ac.kr)

This work was partially supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00387, Development of ICT based Intelligent Smart Welfare Housing System for the Prevention and Control of Livestock Disease, 80%) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2022R1I1A3072355, 20%).

ABSTRACT KubeEdge (KE) is a container orchestration platform for deploying and managing containerized IoT applications in an edge computing environment based on Kubernetes. It is intended to be hosted at the edge and provides seamless cloud-edge coordination as well as an offline mode that allows the edge to function independently of the cloud. However, there are unreliable communication links between edge nodes in edge computing environments, implying that load balancing in an edge computing environment is not guaranteed while using KE. Furthermore, KE lacks Horizontal Pod Autoscaling (HPA), implying that KE cannot dynamically deploy new resources to efficiently handle increasing requests. Both of the aforementioned issues have a significant impact on the performance of the KE-based edge computing system, particularly when traffic volumes vary over time and geographical location. In this study, a node-based horizontal pod autoscaler (NHPA) is proposed to provide dynamical adjustment for the number of pods of individual nodes independently from each other in an edge computing environment where the traffic volume fluctuates over time and location, and the communication links between edge nodes are not stable. The proposed NHPA can dynamically adjust the number of pods depending on the incoming traffic at each node, which will improve the overall performance of the KubeEdge-based edge computing environment. In the KubeEdge-based edge computing environment, the experimental findings reveal that NHPA outperforms KE in terms of throughput and response time by a factor of about 3 and 25, respectively.

INDEX TERMS Kubernetes, KubeEdge, horizontal pod autoscaler, dynamic resource provisioning, edge computing, IoT.

I. INTRODUCTION

Over the past few years, the Internet of Things (IoT) has emerged as an important technology for addressing numerous social challenges such as agriculture, healthcare, and transportation [1], [2]. IoT enables the communication between billions of smart devices [3] for collecting and sharing sensitive data. However, deploying and managing IoT applications are challenging owing to application requirements such as latency, sensitivity, resource restriction of IoT devices, and network bandwidth [4], [5].

Cloud computing has been widely considered in various industrial fields, and its applicability has attracted the attention of researchers [6], [7], [8]. However, the architecture of cloud computing has fundamental limitations for deploying and managing IoT applications. The primary limitation is that cloud computing resources are primarily located far away from IoT devices. In other words, high round-trip latency is added to the IoT application response time. Moreover, because of the massive communication traffic between devices and the cloud, which increases bandwidth

The associate editor coordinating the review of this manuscript and approving it for publication was Wei-Wen Hu.

consumption; hence, communication bottlenecks cannot be avoided in cloud computing.

To overcome the limitations of cloud computing, edge computing was introduced to move the computational resources closer to the data sources (devices) and minimize the propagation distance of packets from end devices to the computational resources. By allocating computational resources, IoT-based edge computing applications can maintain low latency and low bandwidth usage [9], [10] because incoming requests do not need to be transferred to the cloud.

In addition, containerization, represented by Docker, is a lightweight virtualization technology that packages applications and dependent libraries so that they can be run in an independent environment. It has been widely used for deploying and managing applications in both cloud and edge computing environments due to its portable, flexible, and easy deployment features [11]. However, deploying and managing containerized applications requires a container orchestration platform, such as Kubernetes and KubeEdge (KE), to maximize application performance under various circumstances. Kubernetes [12], [13] is a well-known container orchestration platform that offers several features, such as service management and resource provisioning of edge nodes, as well as assuring service availability. A pod is the smallest deployable unit in Kubernetes that contains one or more containers. In Kubernetes, IoT applications are deployed and run on worker nodes in the form of pods [14]. Furthermore, horizontal pod autoscaling (HPA) provided by Kubernetes can optimize the application performance and resource cost by dynamically adjusting the computational resources according to the actual demand.

KE provides better resource orchestration in an edge computing environment because it enables containerized applications to operate properly even when the communications between the cloud node and edge nodes are disrupted [15]. Furthermore, the applications can be deployed to the desired nodes based on the intentions of administrative personnel. Load balancing is an important aspect to be considered for providing seamless services in any computing environment. In the KE-based edge computing environment, the EdgeMesh agent distributes incoming requests to all pods in the cluster for load balancing at each node. But in an edge computing environment, there are no guaranteed communication links between edge nodes, which is necessary for load balancing using EdgeMesh. Load balancing between edge nodes fails due to a lack of stable communication links between geographically dispersed edge nodes. Furthermore, because KE lacks an HPA functionality, edge nodes cannot allocate more resources dynamically to handle heavy traffic demands. As a result of unstable communications links and the lack of HPA, KE is constrained to providing high scalability in an edge computing environment.

To solve the problems mentioned above, this study proposes a node-based horizontal pod autoscaler (NHPA) that allocates new computational resources independently for each edge node to eliminate the effect of the absence of communication links. Here the main idea is to dynamically auto-scale the number of pods based on resource metrics at each node independently to handle the incoming traffic requirements at the node level. The primary concept of NHPA is to apply HPA to each node to ensure that each node can handle incoming requests locally regardless of unstable communication links to the other nodes.

The remainder of this paper is organized as follows. Section II presents related work, and Section III describes the fundamental background of KubeEdge. Section IV describes the proposed NHPA and how it solves the problem of KubeEdge in an edge computing environment. The performance evaluations of NHPA compared with KubeEdge in various traffic scenarios are reported in Section V. Finally, we conclude the paper in Section VI.

II. RELATED WORKS

Edge computing has been considered a new computing paradigm that moves computing and storage capabilities to the network's edge, i.e., nearer to the end devices. Since its advent, edge computing has received significant attention from researchers. More specifically, edge computing has been applied to improve the performance of several technologies, such as voice/vision recognition [16] and 5G [17]. Moreover, several studies have focused on resolving obstacles in deploying applications in an edge computing environment. These studies have demonstrated that resource provisioning and dynamic resource allocation play a vital role in the successful deployment of the application.

Nguyen et al. [18] proposed ElasticFog, which re-allocates computational resources between edge nodes based on the network traffic accessing them. To be precise, ElasticFog distributes more resources to nodes that have more incoming traffic. Similarly, NetMARKS [19] enables resource provisioning with network traffic awareness to improve the quality of IoT services. It improves the efficiency of the Kubernetes scheduling mechanism by extending the Kube scheduler and applying the dynamic network metrics collected with the Istio Service Mesh. Santos et al. [20], [21] proposed a network-aware scheduling mechanism (NAS), which aimed to improve the efficiency of the scheduling mechanism of Kubernetes by extending the Kube-scheduler to reduce the application response time and avoid bandwidth usage violation. Some studies suggest using a leader to coordinate tasks among pods to achieve good performance. Because of its inherent design, the leader often carries heavy workloads. Having the leaders of multiple applications concentrated in a particular node in a Kubernetes cluster may cause the system to bottleneck. So some studies like [22] provide leader selection algorithms that solve the bottleneck problem by uniformly distributing the leaders among nodes in the cluster. Although the experimental results of the study mentioned above show that the developed mechanisms are highly effective, they cannot be thoroughly applied in such an edge computing environment because they do not consider unstable or

unestablished communication links resulting from the geographical distribution of the edge nodes.

Phuc et al. [4] proposed a traffic-aware horizontal pod autoscaler (THPA) that provides proper resource autoscaling according to the actual network traffic accessing edge nodes in an edge computing environment. In other words, THPA collects network traffic information and calculates the number of pods that need to be adjusted on each edge node to ensure that IoT service performance can be improved compared with Kubernetes default autoscaling (KHPA). Similarly, Libra [23] provides a hybrid resource autoscaling mechanism by leveraging both the vertical and horizontal autoscaling mechanisms of Kubernetes. Libra aims to improve autoscaling by calculating and adjusting the appropriate resource limit for every pod in the cluster.

In addition to conventional approaches, several studies have applied machine learning (ML) to improve the efficiency of existing autoscaling mechanisms. One such mechanism that uses ML is HPA+ [24]. HPA+ was developed to provide a proactive autoscaling mechanism to improve the quality of IoT services by employing a multi-forecast ML model to make scaling decisions. Tenfei Hu et al. [25] applied a forecast model to precisely predict the number of pods in the cluster; thus, the application performance can be optimized based on the load fluctuation. Dong et al. [26] proposes JCETD (Joint Cloud-Edge Task Deployment), a reinforcement learning and pruning algorithm based resource management and task deployment strategy.

In addition, the KE platform has also been considered in certain studies to improve the application performance in an edge computing environment. For instance, in [16], KE was applied to improve the efficiency of spectrum intelligent applications by solving terminal storage and computing resource problems. The researchers proposed that the algorithm can be trained on the cloud side based on KE spectrum sensing architecture while interference occurs at edge nodes. In [27], the authors proposed a novel edgemesh framework developed on the KE platform to fit dynamic latency-sensitive multiple service circumstances.

The studies mentioned above have made numerous contributions to solving the problem of the existing resource allocation mechanisms, such as KHPA and Kube scheduler. However, they continue to ignore the unstable or established communication links between nodes, which is one of the essential characteristics of an edge computing environment. Therefore, in this study, we propose the NHPA mechanism that independently adjusts the number of pods on nodes to maximize the application performance regarding response time and throughput in a KE-based edge computing environment.

III. PRELIMINARIES

This section introduces KubeEdge principles, the primary components of KubeEdge, and how KubeEdge works. We also address the Horizontal Pod autoscaling algorithm and its working, as well as the issues that Horizontal Pod autoscaling faces, particularly in Edge Computing contexts.

A. KubeEdge

KE is an open-source orchestration platform built on Kubernetes [15] for extending native containerized applications to edge computing. One of the primary goals of KE is to provide a variety of functions in the network infrastructure that connects the cloud and the edges, such as service deployment and metadata synchronization.

Like Kubernetes, KE forms a cluster with a master node in the cloud and edge nodes at the edges. The main components of KE are Edged, EdgeHub, CloudHub, EdgeController, EventBus, DeviceTwin, MetaManager, ServiceBus, and DeviceController. EdgeD is an agent on edge nodes that manages the pod life cycle. It helps users to deploy containerized workloads or applications at the edge node. EdgeHub is a web socket client responsible for updating cloud-side resources to the edge and reporting status changes from the edge-side host and device to the cloud. In contrast, CloudHub is a web socket server responsible for updating cloud-side changes, caching, and sending update messages to EdgeHub. In KE, the data can be targeted to a specific edge node by the EdgeController, an extended Kubernetes controller. In contrast to Kubernetes, KE supports components' pub/sub capabilities by implementing the EventBus, an MQTT client/server model. DeviceTwin is responsible for storing device statuses and updating their statuses on the cloud side. Devices metadata can be stored in MetaManager in a lightweight database (SQLite) that enables applications easily retrieve necessary information. Finally, DeviceController manages devices to ensure that the device metadata/status data can be synchronized between the edge and cloud by extending the Kubernetes controller. In contrast, ServiceBus offers a reachable HTTP connection from cloud components to the servers running at the edge. In KE, the network connectivity is provided by EdgeMesh [28], which ensures the continuity of applications even when the connection between the edge and cloud is disrupted. The EdgeMesh component comprises two main sub-components: the EdgeMesh Server and EdgeMesh Agent. While EdgeMesh Server is responsible for establishing a connection with the EdgeMesh Agent to provide relay and hole punching capability, EdgeMesh Agent ensures that incoming traffic can reach the backend pods running on the edges.

Load balancing allows you to spread the workload evenly among available resources. Its goal is to provide continuous service if any component fails by provisioning and de-provisioning application instances to utilize resources properly. Furthermore, load balancing tries to reduce task response time and optimize resource usage, which improves system performance at a reduced cost. Load balancing also offers scalability and flexibility for applications whose size may rise in the future and demand additional resources, as well as prioritize jobs that require immediate execution above other processes.

Load balancing is achieved in KubeEdge-based edge computing systems through the EdgeMesh agent, which acts as a network load balancer by forwarding incoming requests using the random, round-robin, and session persistence methods. As a result, device requests can be efficiently handled by distributing them to local and remote backend pods on the cluster's edge nodes where the requests are processed. However, in edge computing environments, this approach has several disadvantages, including:

1) In Edge computing environments, the pods of an application can be geographically dispersed, which can incur significant delays when the requests are forwarded from one node to another. Also, in geographically distributed systems, the traffic load varies by location and time [29], resulting in a demand imbalance across nodes.
2) Forwarding requests to nodes without knowing the status of their resources can result in non-optimal results. Some nodes are too busy to manage a high traffic volume, while others remain idle [20], resulting in degraded overall performance.
3) In an environment such as edge computing, where communication links between nodes might be unstable or even absent, any attempt to distribute the incoming requests between nodes can reduce the overall performance of the system.

B. HORIZONTAL POD AUTOSCALER

The Horizontal Pod Autoscaler (HPA) is a representative technology for providing high scalability and availability in Kubernetes that regularly alters the number of computational resources (pods) in the cluster. It ensures that the cluster's state always meets the desired state according to the specified metric values [30]. Furthermore, the HPA is configurable with metrics such as CPU, memory, and custom metrics to support various usage circumstances.

The workflow of HPA is illustrated in Algorithm 1 [4]. In each period denoted by *HPA_Sync_Period*, HPA collects the number of current pods in the cluster (*curPods*), the current metric value (*curMetricVal*), and the desired metric value (*dMetricVal*). Subsequently, it calculates the desired number of pods in the cluster (*dPods*) based on the *ratio* (in line 5). If the desired number of pods in the cluster differs from the current number of pods in the cluster, the number of application pods in the cluster is adjusted by the HPA (line 8 in Algorithm 1).

The upscaling process is invoked if the desired number of pods is greater than the current number of pods in the cluster, whereas the downscaling process is invoked in the opposite case. For instance, given that the CPU usage metric is enabled, assume that all pods' current average CPU usage is 100 m (where m stands for millicore) and the desired CPU metric value is 50 m. HPA doubles the number of pods in the cluster. In contrast, when the current CPU metric value is 50 m and the desired CPU metric value is 100 m, HPA halves the number of pods in the cluster. Let n denote the number of nodes in the cluster. Lines 2-6 will take one unit of time(constant) and the function *setDesiredPods*(*app*, *dpods*) in line 8 sets the number of pods by allocating new pods for each edge node (upscaling) or terminating pods for each node (downscaling), it needs to send commands to n nodes. Therefore, we can say that the time complexity of Algorithm 1 is $O(n)$. Although HPA is an essential feature for dynamic resource orchestration, it is not a default option of KubeEdge. Thus, it cannot handle incoming traffic dynamically if the amount of traffic is high.

Algorithm 1 HPA Algorithm [4]

pods : *list of application pods in cluster*.
curPods : *current number of application pods*.
dPods : *desired number of application pods*.
curMetVal : *current metric value*.
dMetVal : *desired metric value*.
HPA_Sync_Period : *HPA sync period*.

```
1:  while true do
2:      curPods = getCurPods()
3:      curMetVal = getCurMetricValue(app)
4:      dMetVal = getDesiredMetricValue(app)
5:      ratio = curMetricVal ÷ dMetricVal
6:      dPods = ceil[ratio × curPods]
7:      if dPods ! = curPods then
8:          setDesiredPods(app, dPods)
9:      end if
10:     time.sleep(HPA_Sync_Period)
11: end while
```

IV. NODE-BASED HORIZONTAL POD AUTOSCALER

In this section, we will go through the KubeEdge-based edge computing architecture and highlight some of the current issues that KHPA is dealing with. Then, we discuss NHPA, its working, and the detailed algorithm of NHPA.

A. KubeEdge-BASED EDGE COMPUTING ARCHITECTURE

Figure 1 (a) illustrates the architecture of a KE-based edge computing system. The application is deployed on edge nodes in the form of pods and can be accessed by IoT devices. Notably, the EdgeMesh agent is enabled and configured with a round-robin-based traffic forwarding mechanism on each node to ensure that the traffic loads are evenly distributed among application pods in the cluster. In other words, local and remote pods handle incoming traffic at each node. Thus, none of the nodes becomes overloaded solely because traffic distribution of the EdgeMesh agent.

However, in an edge computing environment, edge nodes are geographically dispersed. Thus, their communication links might not be stable or even existent. Because EdgeMesh's load balancing works on connected worker nodes, it cannot provide round-robin-based traffic forwarding in an edge computing environment where the links between nodes are unstable or absent, as shown in Figure 1 (a).

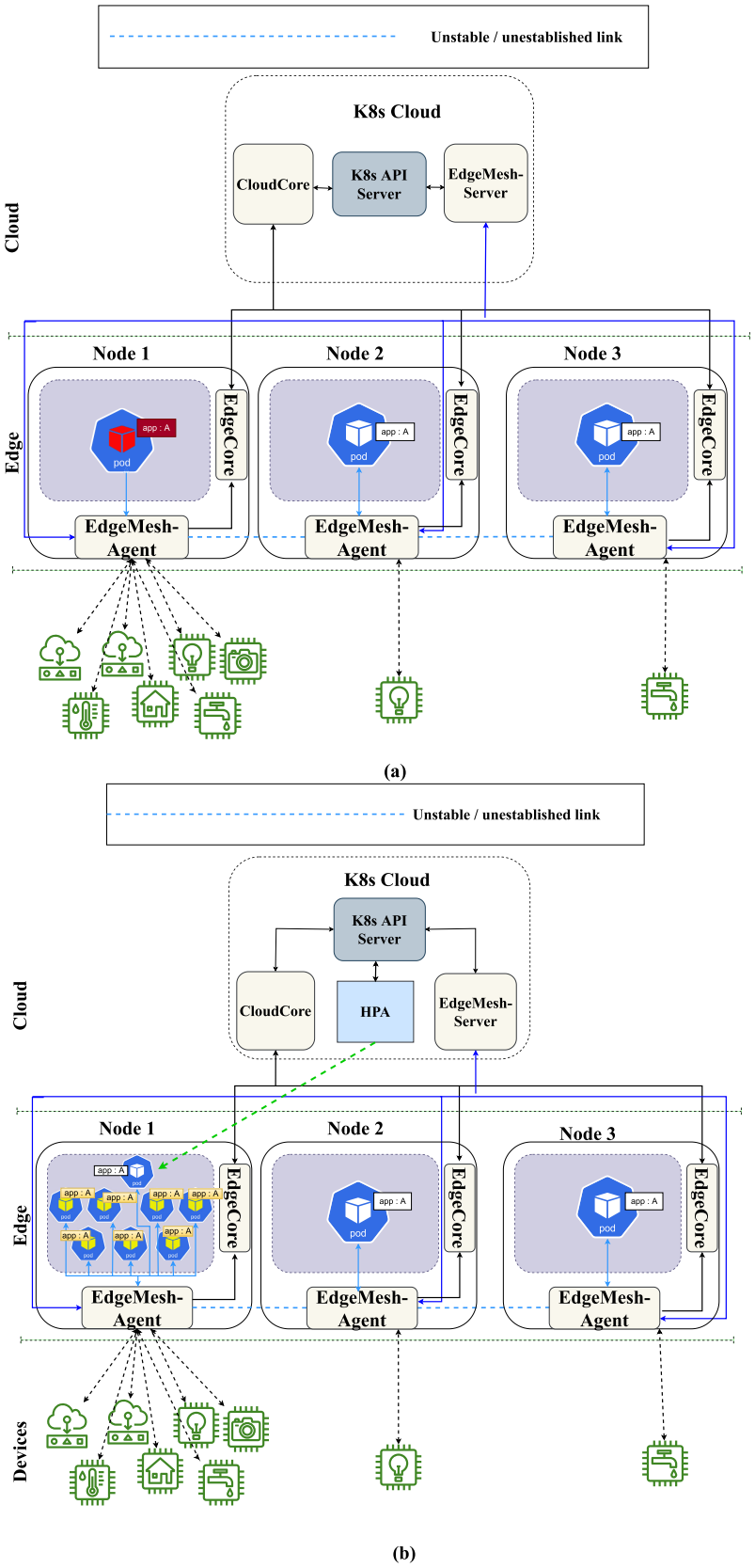

FIGURE 1. (a) KubeEdge-based edge computing architecture, and (b) Node-based HPA in KubeEdge-based edge computing architecture.

For example, although the volume of incoming traffic at Node 1 is high as it deals simultaneously with seven devices, the EdgeMesh agent on Node 1 cannot spread incoming traffic because of unstable or absent links. Thus, traffic at Node 1 can only be handled by local pods, which might lead to an overload at Node 1 when handling heavy traffic concentrated from seven devices.

In addition, the number of pods on edge nodes is fixed because KE does not necessarily include an HPA feature. However, to adapt to the actual demand from IoT devices that fluctuates over time and locations in an edge computing environment, the number of pods on edge nodes can be adjusted based on their workload in real-time and locations. In particular, as shown in Figure 1 (a), more application pods should be allocated to Node 1 to adapt to the demand of seven IoT devices to reduce resource overload and improve application performance.

By carefully considering the abovementioned problems in an edge computing environment, we propose an NHPA that independently adjusts the number of pods on edge nodes to maximize the application performance. As shown in Figure 1 (b), because the communication links between edge nodes are not established, the NHPA adjusts the number of pods at each edge node independently based on resource usage. Thus, the number of pods on Node 1 becomes nine, while other nodes retain the same number of pods as earlier. Therefore, the number of incoming requests at Node 1 can be handled effectively and locally, even without cooperation with other nodes.

B. NODE-BASED HORIZONTAL POD AUTOSCALER

This subsection describes the details of the proposed NHPA. Generally, KE does not provide HPA, although it is inherited from Kubernetes. Thus, it cannot provide scalability for workload fluctuations. For example, in Figure 1 (a), we assume that a KE system comprises three edge nodes with one application pod, which are not mutually connected because of the geographical distribution. Although the number of devices accessing Node 1 increases to seven, the number of pods on Node 1 remains one because of the lack of autoscaling features. Consequently, the application performance at Node 1 will deteriorate as Node 1 becomes overloaded with traffic from seven devices. Furthermore, it cannot offload the traffic to the remote edge nodes because of the absence of connecting links.

Therefore, to adapt to fluctuations in traffic for each location (node), we propose NHPA that can independently adjust the number of pods for each node based on its workload. This approach is more flexible and efficient than HPA as each node can independently adjust the number of pods as per its requirement, hence it also doesn't require any coordination between edge nodes. Details of the underlying mechanism of NHPA are presented in Algorithm 2. The primary benefit of NHPA is that it works for each node individually in contrast to the HPA in Algorithm 1. Thus, the information such as the current number of pods in the cluster (*curPods*), current metric value (*curMetVal*), the desired metric value (*dMetVal*) and the desired number of pods in the cluster (*dPods*) are measured and calculated from the perspective of an individual node in Algorithm 2. Subsequently, if the desired number of pods differs from the current number of pods, the desired number of pods in the cluster will be updated, and scaling will occur. For instance, in Figure 1 (b), NHPA can adjust the number of pods at Node 1 to nine while retaining one at Nodes 2 and 3 to handle the increasing traffic. Consequently, the nine pods at Node 1 can efficiently handle incoming traffic from seven devices without the need to distribute it to other nodes. Let n denote the number of nodes in the cluster. The loop started in line 2 of Algorithm 2 will be executed n times and the rest of the lines 3-10 will take one unit of time(constant). Therefore, we can say that the time complexity of Algorithm 2 is $O(n)$. Here we need to consider that the setDesiredPods() function in line 8 of Algorithm 2 is executed one time for a specific node, while the setDesiredPods() function in Algorithm 1 requires a repetitive process for n nodes. Hence, the proposed approach can ensure an effective pod autoscaling feature in an edge computing environment with unstable or absent links between edge nodes.

Algorithm 2 NHPA Algorithm

Nodes : *list of edge nodes in cluster*.
pods : *list of application pods in cluster*.
curPods : *current number of application pods of each node*.
dPods : *desired number of application pods of each node*.
curMetVal : *current metric value of each node*.
dMetVal : *desired metric value of each node*.
HPA_Sync_Period : *HPA sync period*.

```
1:  while true do
2:      for node ∈ Nodes do
3:          curPods = getCurPods(node)
4:          curMetVal = getCurMetricValue(app, node)
5:          dMetVal = getDesiredMetricValue(app, node)
6:          ratio = curMetricVal ÷ dMetricVal
7:          dPods = ceil[ratio × curPods]
8:          if dPods ! = curPods then
9:              setDesiredPods(app, dPods, node)
10:         end if
11:     end for
12:     time.sleep(HPA_Sync_Period)
13: end while
```

V. PERFORMANCE EVALUATIONS

In this section, we first define the experimental setup and the performance assessment criteria. Then we enlisted some scenarios to be tested for performance comparison in normal and overloaded behaviours. Further, we examine the outcomes of pod distribution with and without NHPA. We also test the performance of NHPA utilizing throughput and latency at one

TABLE 1. Pods distribution according to network traffic accessing cluster.

Scenario	Traffic ratio[Node1:Node2:Node3]	Pods Distribution [Node1-Node2-Node3] in NHPA	Pods Distribution [Node1-Node2-Node3] in KE
1	1:1:1	2-2-2	1-1-1
2	3:3:3	5-5-5	1-1-1
3	5:2:2	8-4-4	1-1-1
4	7:1:1	9-2-2	1-1-1
5	5:5:5	8-8-8	1-1-1
6	7:4:4	9-6-6	1-1-1
7	7:7:1	9-9-2	1-1-1

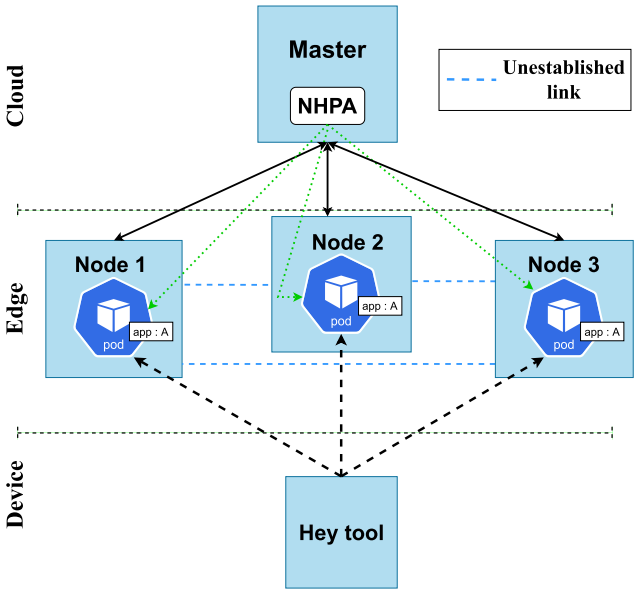

FIGURE 2. Experimental setup.

edge node. Finally, we analyze the performance of NHPA at the cluster level.

A. EXPERIMENTAL SETUP

In this section, we evaluate the performance of NHPA in terms of throughput and latency in various scenarios. A KE cluster was set up as shown in Figure 2, which includes one master node, three edge nodes, and one for generating traffic.

The communication links between edge nodes are unestablished to simulate actual edge computing environment conditions. All machines were equipped with six CPU cores and 8 GB of RAM, and the KE version used in the cluster was 1.9.1. A hey tool is installed on the traffic generator machine to send requests to the application. The number of pods in the KE cluster was set to three by default, and they were evenly distributed across the three edge nodes. For NHPA, the minimum and the maximum number of pods and average CPU usage threshold were set as 3, 12, and 80%, respectively.

B. PODS DISTRIBUTION ACCORDING TO NETWORK TRAFFIC

Table 1 compares the pod distribution between the cases in which KE is used without the NHPA feature and when NHPA is applied to KE in various scenarios. From here on, we use the format x:y:z to represent the proportion of concurrent requests accessing nodes and x-y-z to represent the distribution of the pods in the cluster. For example, 1:1:1 indicates the case with one request accessing each node, while 1-1-1 denotes each node having one application pod. Furthermore, Table 1 shows that the number of pods in KE remains constant regardless of traffic types, whereas NHPA adjusts them to adapt to the traffic distribution. For example, in 1:1:1 traffic, although one application pod cannot effectively handle one concurrent request, KE maintains one pod on each node. In contrast, NHPA adjusts the number of pods in each node to 2-2-2 to effectively handle the traffic demand based on the average CPU usage. In particular, NHPA scales up the number of pods at the node scale to ensure that each node has a sufficient number of pods to handle the incoming requests locally. For example, when the distribution of traffic is balanced (e.g., 3:3:3), NHPA increases the number of pods on each node to 5. In contrast, the pod distributions in the cluster are changed to 8-4-4 and 9-2-2, respectively, for imbalanced traffic in cases such as 5:2:2 and 7:1:1. Consequently, application performance when using NHPA is considerably higher than that of KE, which will be scrutinized in the following sections.

C. APPLICATION PERFORMANCE AT ONE EDGE NODE

To demonstrate the efficiency of NHPA in an edge computing environment, we estimated the application throughput and response time of NHPA and KE at Node 1, as shown in Figure 3. Because the communication links between nodes are not established, EdgeMesh Agent at Node 1 only forwards incoming traffic to the local pods of Node 1. Thus, the

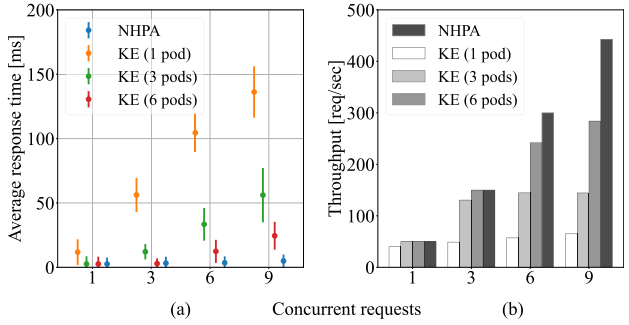

FIGURE 3. Application performance at Node 1 in terms of response time (a), and throughput (b).

incoming traffic is handled locally by the pods on Node 1. Furthermore, KE enables the configuration of the number of pods to be changed according to traffic demand. Thus, we used the 1, 3, and 6 pods configuration in this evaluation. In NHPA, we initially set the number of pods to one. In any case, it can dynamically autoscale the number of pods based on the traffic volume. To compare the performance according to the amount of traffic, we increased the number of concurrent requests accessing the application service on the node from 1 to 9.

As shown by the observed average response time in Figure 3 (a), a higher number of pods at Node 1 enables the system to handle more incoming requests without delay, thereby lowering the application response time. However, KE lacks dynamic scaling of the number of pods, which limits its ability to handle incoming requests effectively. For example, as it requires two pods to handle one concurrent request, as already discussed in Table 1, KE (one pod) results in a high response time because it cannot increase the number of pods. KE (three pods) and KE (six pods) achieved low response times for one concurrent request based on sufficient resources; however, the response times tended to increase as the number of concurrent requests increased. In contrast, NHPA exhibits a steady increment in response time from 2.6 ms to 4.9 ms because it allocates a sufficient number of pods, i.e., 2, 5, 9, and 11 for 1, 3, 5, and 9 concurrent requests, respectively.

Figure 3 (b) compares the throughput of NHPA and KE according to the increase in traffic accessing Node 1. It shows that the throughput increases as more pods are allocated in the case of KE; however, the maximum achievable throughput is limited by the number of pods. For example, the maximum throughput with one pod, three pods, and six pods is approximately 65, 144, and 284 req/s, respectively, and it cannot increase further, even in the case of higher network traffic. Furthermore, the number of available resources limits the maximum achievable throughput. In contrast, NHPA dynamically adjusts the number of pods according to network traffic. For instance, it allocates 2, 5, 9, and 11 pods for one, three, six, and nine concurrent requests, respectively. Therefore, we can conclude that NHPA improves throughput by allocating pod resources proportionally to the network traffic volume that accesses each node.

D. APPLICATION PERFORMANCE AT CLUSTER SCALE

To analyze the effect of pod distribution on the overall system performance in an edge computing environment, this subsection evaluates the application response time and cumulative throughput of KE and NHPA according to scenarios 2, 3, and 4 of Table 1. We evaluated the network traffic accessing each edge node such as 3:3:3, 5:2:2, and 7:1:1, and NHPA dynamically allocates the number of pods to 5-5-5, 8-4-4, and 7-1-1 to accommodate each traffic pattern as reported in Table 1. KE, on the other hand, cannot dynamically adjust and has a fixed number of pods. By default, KE allocates one pod to each node, noted as KE (1-1-1), as listed in Table 1, but we extend the evaluation cases such as KE (3-3-3) and KE (6-6-6) that each edge node is allocated three and six pods respectively for the fair comparison with NHPA.

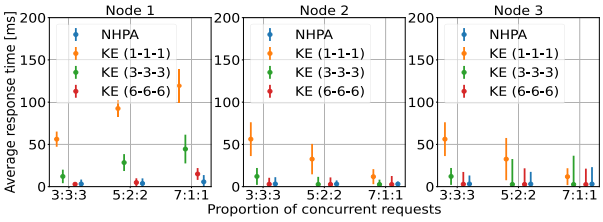

FIGURE 4. Application response time at three nodes.

Figure 4 shows that the response time of KE varies according to the amount of incoming traffic because it maintains a constant number of pods at each node. In contrast, NHPA maintains a low response time regardless of the type of traffic. In other words, the application response times in both KE and NHPA tend to be reduced for smaller traffic and higher resources. This result can be attributed to the fact that the incoming requests are handled by local edge nodes owing to the lack of communication links between nodes. When the traffic accessing the cluster is balanced (e.g.,3:3:3), all nodes have the same response time because they have the same number of pods. However, KE (1-1-1) exhibits a response time greater than 50 ms, whereas others exhibit a low response time of less than 10 ms. This discrepancy in response time indicates that one pod at each node cannot efficiently handle three concurrent requests immediately, resulting in a processing delay. For imbalanced traffic, such as 5:2:2 and 7:1:1, we observe that the response time at Node 1 in KE increases as the incoming traffic increases. This result demonstrates that each node in the KE has a predetermined and fixed number of pods. Thus, the system cannot dynamically adapt to increasing traffic. In contrast, NHPA shows a low response time regardless of the traffic pattern and amount of traffic accessing each node. This low response time can be achieved because each node in the NHPA adjusts the number of pods dynamically and independently, according to the traffic load from the other edge nodes.

Figure 5 shows the cumulative throughput of KE and NHPA against diverse network traffic patterns. In general, the cumulative throughput of the KE tends to increase for the allocated pod resources, but it decreases as the traffic concentrates on Node 1. For example, the aggregated throughput for 7:1:1 decreases compared with 3:3:3 for all pod distributions in the KE. However, in this case, the total amount of incoming traffic of 3:3:3, 5:2:2, and 7:1:1 is the same, and only the traffic distribution accessing each node is different. In KE, as the traffic concentrates on Node 1, the throughput of Nodes 2 and 3 decreases owing to the reduced incoming traffic, whereas that of Node 1 is bounded owing to the fixed number of pod resources. In contrast, NHPA independently

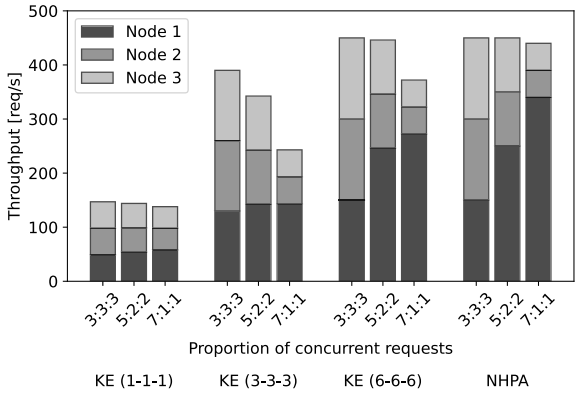

FIGURE 5. **Application cumulative throughput at three nodes.**

adjusts the number of pods of each node in all traffic cases; thus, it achieves a high aggregated throughput regardless of the traffic distribution. Especially in imbalanced traffic, such as in the case of 7:1:1, NHPA achieves approximately 219%, 81%, and 18% higher throughput than KE (1-1-1), KE (3-3-3), and KE (6-6-6), respectively, by allocating more pods to Node 1. Therefore, we can conclude that NHPA improves the overall performance for all types of traffic distribution accessing the edge computing environment by fully utilizing the available resources of each node independently and dynamically.

VI. CONCLUSION

KubeEdge-based edge computing environments are widely used in the distribution and administration of containerized IoT applications because they allow applications to operate smoothly even when connections between the cloud and the edge are disrupted. However, communication links between nodes may be weak or even nonexistent in edge computing environment where edge nodes are geographically distributed, therefore KubeEdge's load balancing that forwards requests to remote nodes may fail. Although the edge node needs to handle the incoming request locally in this case, KubeEdge lacks a mechanism for dynamically allocating computing resources. In this study, we proposed NHPA, which independently adjusts the number of pods in the cluster for each edge node to maximize application performance regarding response time and throughput. The experimental results showed that the NHPA performed better than KE in the KubeEdge-based edge computing environment. When compared to KE, the throughput and response time of NHPA were reduced by a factor of 3 and 25, respectively. Therefore, we concluded that providing resource autoscaling independently for each node can significantly improve application performance by removing the effect of unstable communication links in an edge computing environment.

ACKNOWLEDGMENT

(Le Hoang Phuc and Majid Kundroo contributed equally to this work.)

REFERENCES

[1] B. Ahlgren, M. Hidell, and E. C.-H. Ngai, "Internet of Things for smart cities: Interoperability and open data," *IEEE Internet Comput.*, vol. 20, no. 6, pp. 52–56, Nov. 2016.
[2] A. A. Sadri, A. M. Rahmani, M. Saberikamarposhti, and M. Hosseinzadeh, "Fog data management: A vision, challenges, and future directions," *J. Netw. Comput. Appl.*, vol. 174, Jan. 2021, Art. no. 102882.
[3] H. Zhang, J. Yu, C. Tian, L. Tong, J. Lin, L. Ge, and H. Wang, "Efficient and secure outsourcing scheme for RSA decryption in Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6868–6881, Aug. 2020.
[4] L. H. Phuc, L.-A. Phan, and T. Kim, "Traffic-aware horizontal pod autoscaler in Kubernetes-based edge computing infrastructure," *IEEE Access*, vol. 10, pp. 18966–18977, 2022.
[5] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13723–13736, Sep. 2021.
[6] Y.-H. Hung, "Investigating how the cloud computing transforms the development of industries," *IEEE Access*, vol. 7, pp. 181505–181517, 2019.
[7] S. Aljanabi and A. Chalechale, "Improving IoT services using a hybrid fog-cloud offloading," *IEEE Access*, vol. 9, pp. 13775–13788, 2021.
[8] N. Al-Nabhan, S. Alenazi, S. Alquwaifili, S. Alzamzami, L. Altwayan, N. Alaloula, R. Alowaini, and A. B. M. A. A. Islam, "An intelligent IoT approach for analyzing and managing crowds," *IEEE Access*, vol. 9, pp. 104874–104886, 2021.
[9] M. Bukhsh, S. Abdullah, and I. S. Bajwa, "A decentralized edge computing latency-aware task management method with high availability for IoT applications," *IEEE Access*, vol. 9, pp. 138994–139008, 2021.
[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st, Ed., MCC Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 13–16.
[11] J. Zhang, X. Zhou, T. Ge, X. Wang, and T. Hwang, "Joint task scheduling and containerizing for efficient edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 2086–2100, Aug. 2021.
[12] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4228–4237, May 2020.
[13] E. Truyen, N. Kratzke, D. Van Landuyt, B. Lagaisse, and W. Joosen, "Managing feature compatibility in Kubernetes: Vendor comparison and analysis," *IEEE Access*, vol. 8, pp. 228420–228439, 2020.
[14] Z. Cai and R. Buyya, "Inverse queuing model-based feedback control for elastic container provisioning of web systems in Kubernetes," *IEEE Trans. Comput.*, vol. 71, no. 2, pp. 337–348, Feb. 2022.
[15] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 373–377.
[16] D. Wang, Y. Zhou, J. Chen, and C. Zhang, "Research on spectrum intelligent recognition technology based on an edge processing framework KubeEdge," in *Proc. 6th Int. Conf. Intell. Comput. Signal Process. (ICSP)*, Apr. 2021, pp. 847–850.
[17] S. Douch, M. R. Abid, K. Zine-Dine, D. Bouzidi, and D. Benhaddou, "Edge computing technology enablers: A systematic lecture study," *IEEE Access*, vol. 10, pp. 69264–69302, 2022.
[18] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and T. Kim, "ElasticFog: Elastic resource provisioning in container-based fog computing," *IEEE Access*, vol. 8, pp. 183879–183890, 2020.
[19] L. Wojciechowski, K. Opasiak, J. Latusek, M. Wereski, V. Morales, T. Kim, and M. Hong, "NetMARKS: Network metrics-AwaRe Kubernetes scheduler powered by service mesh," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–9.
[20] J. Santos, T. Wauters, B. Volckaert, and F. D. Turck, "Resource provisioning in fog computing: From theory to practice," *Sensors*, vol. 19, no. 10, p. 2238, May 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/10/2238
[21] N. A. Cv and R. Lavanya, "Fog computing and its role in the Internet of Things," in *Advancing Consumer-Centric Fog Computing Architectures*. Hershey, PA, USA: IGI Global, 2019, pp. 63–71.
[22] N. D. Nguyen and T. Kim, "Balanced leader distribution algorithm in Kubernetes clusters," *Sensors*, vol. 21, no. 3, pp. 1–15, 2021.
[23] D. Balla, C. Simon, and M. Maliosz, "Adaptive scaling of Kubernetes pods," in *Proc. NOMS IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2020, pp. 1–5.

[24] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for Kubernetes edge clusters," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 958–972, Mar. 2021.
[25] T. Hu and Y. Wang, "A kubernetes autoscaler based on pod replicas prediction," in *Proc. Asia–Pacific Conf. Commun. Technol. Comput. Sci. (ACCTCS)*, Jan. 2021, pp. 238–241.
[26] Y. Dong, G. Xu, M. Zhang, and X. Meng, "A high-efficient joint 'cloud-edge' aware strategy for task deployment and load balancing," *IEEE Access*, vol. 9, pp. 12791–12802, 2021.
[27] T. Yang, J. Ning, D. Lan, J. Zhang, Y. Yang, X. Wang, and A. Taherkordi, "Kubeedge wireless for integrated communication and computing services everywhere," *IEEE Wireless Commun.*, vol. 29, no. 2, pp. 140–145, Apr. 2022.
[28] *GitHub Kubeedge/Edgemesh: Simplified Network and Services for Edge Applications*. Accessed: Nov. 26, 2022. [Online]. Available: https://github.com/kubeedge/edgemesh
[29] Z. Ma, S. Shao, S. Guo, Z. Wang, F. Qi, and A. Xiong, "Container migration mechanism for load balancing in edge network under power Internet of Things," *IEEE Access*, vol. 8, pp. 118405–118416, 2020.
[30] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, "Horizontal pod autoscaling in Kubernetes for elastic container orchestration," *Sensors*, vol. 20, no. 16, p. 4621, Aug. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/16/4621

LE HOANG PHUC received the B.S. degree in electronics and communication engineering from Can Tho University, in 2018. He is currently pursuing the M.S. degree with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include the IoT applications, container orchestration, open-source software, and cloud/edge computing.

MAJID KUNDROO received the M.S. degree in computer science from the Islamic University of Science and Technology, Jammu and Kashmir, India, in 2019. He is currently pursuing the Ph.D. degree with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include edge computing, edge AI, the Internet of Things, and federated learning.

DAE-HEON PARK received the B.S., M.S., and Ph.D. degrees in communication and information engineering from Sunchon National University, in 2006, 2008, and 2015, respectively. Since 2011, he has been a Senior Research with the Electronics and Telecommunications Research Institute (ETRI), South Korea. His research interests include the IoT, AI, cloud, bigdata, and ICT convergence with agriculture.

SEHAN KIM received the B.S. and M.S. degrees in computer engineering from Korea Aerospace University, South Korea, in 1998 and 2000, respectively. He was a Research Staff at the Samsung Advanced Institute of Technology, in 2000. Since 2001, he has been the Principal Researcher and the Director of the Electronics and Telecommunications Research Institute (ETRI), South Korea. His research interests include digital twin, platform, data science with the IoT, AI, cloud, bigdata, and intelligent ICT convergence with agriculture, food, and fisheries.

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