Received 12 October 2023, accepted 26 October 2023, date of publication 6 November 2023, date of current version 14 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3330649

RESEARCH ARTICLE

SatEdge: Platform of Edge Cloud at Satellite and Scheduling Mechanism for Microservice Modules

YANZHE HUANG, XING ZHANG, (Senior Member, IEEE), AND ZECHANG XU
School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Xing Zhang (zhangx@ieee.org)

This work was supported by the National Key Research and Development Program of China under Grant 2021YFB2900504 and Grant 2020YFB1807900.

ABSTRACT Edge cloud at satellite (ECS) is a newly developed edge computing (EC) technology that uses EC services offered by satellites to support high reliability and seamless global coverage. Satellites assume the role of computing and storage nodes for edge clouds, while terrestrial control centers function as cloud centers. In this paper, we propose a novel system and software architecture for the ECS to improve the cloud management of satellite networks and increase the flexibility of satellite service provision at the edge. Then, we propose a platform for the ECS based on KubeEdge called SatEdge. SatEdge has many function modules to meet the needs of the satellite-terrestrial network (STN) such as high reliability, high flexibility, and low latency. On this platform, we designed a microservice scheduling algorithm called optimal microservice scheduling with adaptivity and mobility (OMS-AM). OMS-AM can schedule a globally optimal workflow for microservice modules on the satellites to minimize task processing latency, failed task rate, and energy consumption. Compared with our last work, OMS-AM reduces the task processing latency by 14% at most. Additionally, OMS-AM improves the mobility of the current scheduling method put forth in our previous study, which may help lower the task failure rate. Energy usage and the total normalized costs are additional indicators of the efficiency of the microservice architecture.

INDEX TERMS Edge cloud at satellite, satellite-terrestrial network, edge computing, microservice scheduling.

I. INTRODUCTION

According to the Global Mobile Market Report released by Ericsson in 2022 [1], 5G now covers 25% of the global population and is projected to reach 75% by 2027. This poses a challenge to current terrestrial networks because the current ground communication system cannot achieve seamless coverage. Therefore, the optimization of future communication networks is urgently required. As illustrated in [2], the future generation of wireless communication systems is expected to support the enormous growth in traffic and guarantee seamless coverage. Satellite networks, as suggested by [3], offer a robust solution to enhance and extend terrestrial mobile communication systems.

Additionally, with the development of high-performance computing technology in space, satellite computing capabilities have improved, providing a computational foundation for the development of satellite edge computing (EC) technology [4]. EC provides more network functions and content to the network edge [5]. A software-defined network based on the edge cloud computing was proposed in [6]. And the authors of [7] proposed a data aggregation mechanism based on EC. However, future satellite-terrestrial network (STN) will require differentiated service scenarios such as wide-area high-speed bandwidth, wide-area reliability, and wide-area Internet of Things (IoT), which pose higher demands on rate, error rate, and latency indicators. This leads to problems such as limited resources, low efficiency of running isolation, and delayed response time when adapting to future STN. Through satellite edge cloud, we can deploy cloud computing resources and services in satellite systems, providing computing and storage services to support various application requests from Earth [8].

Although there have been some studies on satellite network platforms integrated with EC, these systems still lack experimental verification in production and development

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamed M. A. Moustafa.

environments. According to our survey, the most relevant research lacks well-defined software architecture to provide guidance for the development process. In [3], the authors proposed a novel satellite terrestrial network using double edge computing. A double-edge intelligent integrated satellite and terrestrial network was proposed in [2]. Both of these works focus on the design of EC in STN, and the EC servers are deployed in satellite networks. In [9], the authors proposed a classic software-defined-network (SDN) aware hybrid satellite-terrestrial network architecture. The architecture and design of the mentioned satellite platforms can be referred to in future systems. However, these platforms are built based on OPNET. This tool cannot realize the decentralized architecture of a real EC. In [10], the authors proposed a satellite edge computing simulator, SatEdgeSim, which is an extension of PureEdgeSim. A dynamic network virtualization technique in the STN scene was proposed in [11], but these two studies lacked a decentralized software architecture. Despite numerous studies on satellite architecture and platforms, few have employed container orchestration tools on these platforms. The authors of [12] discussed containers and clusters for edge cloud architecture, and application and service orchestration can help manage and orchestrate applications through containers. The suitability of container and cluster technologies can be used to build an STN system. Thus, in our work, when we build the satellite edge cloud platform, we not only refer to the cloud and EC architecture of social enterprises but also industrial practice [13]. Therefore, we used container-based edge cloud architecture. Moreover, the authors in [14] researching on the satellite-high altitude platform-terrestrial, and this work maximizes the sum rate of secondary network.

Technologies such as containers and virtualization have changed the manner in which various workloads and applications are deployed, run, and managed [15]. Most existing studies assume that applications are deployed in cloud data centers, but cloud computing is moving from centralized, large-scale data centers to a more distributed edge cloud architecture [12]. To meet requirements such as data privacy and low latency, enterprises need to extend workloads from the cloud to the edge to perform data aggregation, machine learning inference tasks, etc. This can also alleviate the pressure on communication networks caused by the explosive growth in data traffic. In [16], the authors proposed an eCaaS (namely edge Container as a Service) framework to address challenges from an edge computing service provider perspective. In addition, container orchestration using cluster tools is a prevalent approach that involves master and agent roles. The master manages the workloads of the agents and monitors their real-time statuses. Kubernetes (K8s) [17], Rancher, and Docker are popular platforms for building clusters. However, new platforms such as K3s [18] and KubeEdge [19] have emerged to meet the lightweight requirements of edge nodes in edge computing scenarios. New use cases require performant, available, and scalable orchestration at the edge [20]. The authors in [21] discuss EC and its current situation. We also compared several open-source tools (Table 1). In the K8s architecture, the minimum configuration requirements for edge nodes are 2 core CPU and 2GB of memory [17]. In the K3s architecture, the minimum configuration requirements for edge nodes are 1 core CPU and 512MB of memory [18]. In the KubeEdge architecture, the minimum configuration requirement for edge nodes is 1 core CPU and 80MB [19]. So in Table 1, we can draw conclusions about the lightweighting degree of edge nodes. In [16], the authors also compared various aspects of the three tools. Based on Table 1, we can conclude that KubeEdge is more suitable for deployment in edge computing scenarios owing to its lightweight and decentralized nature, and its additional functions that enable the efficient operation of edge nodes [22].

The microservice architectural pattern has emerged based on the development of containers. Microservices split an application into small self-contained services that can be independently developed, deployed, and scaled. Containers are commonly employed in hosting services within a microservice architecture [15]. The lightweight, portable, and self-contained nature of microservices makes them ideal for use in satellite networks that have limited resources and rapidly changing topologies. Because the virtualization technology represented by containers can effectively improve resource limited issues, such as resource isolation, resource scheduling, resource limitations, and quotas. In satellite communication systems, the authors in [23] introduced a threshold-based user scheduling scheme designed to enhance secure transmission. However, it is worth noting that there is currently a gap in the literature concerning scheduling schemes tailored specifically for microservices in this context. Microservices not only integrate the excellent features of the container, but also changes the traditional software operation mode from a stand-alone application to independent and autonomous service modules. The microservice architecture can promote satellite edge computing, such as task migration and computation offloading on satellites, ground control center detection, and satellite management. However, owing to the limitations of satellite payload, volume, energy consumption, and computing and storage resources, a more flexible and controllable resource management approach is required. However, few studies have been conducted on microservice architecture in satellite networks. A 5G satellite edge computing framework consisting of embedded hardware platforms and microservices in satellites was proposed in [24], and the experiments were carried out in MATLAB. Moreover, the deployment of microservices on satellite platforms requires effective task workflow planning and container orchestration tools to manage the lifecycle of container-based microservices. Although some researchers have developed models for microservices to minimize network delays and costs, they have not defined specific workloads or run their algorithms in real production environments, such as [25] and [26]. Additionally, in [27], the authors propose microservice scheduling methods in satellite networks, but the number of satellites involved was limited, and its experimental results were incomplete.

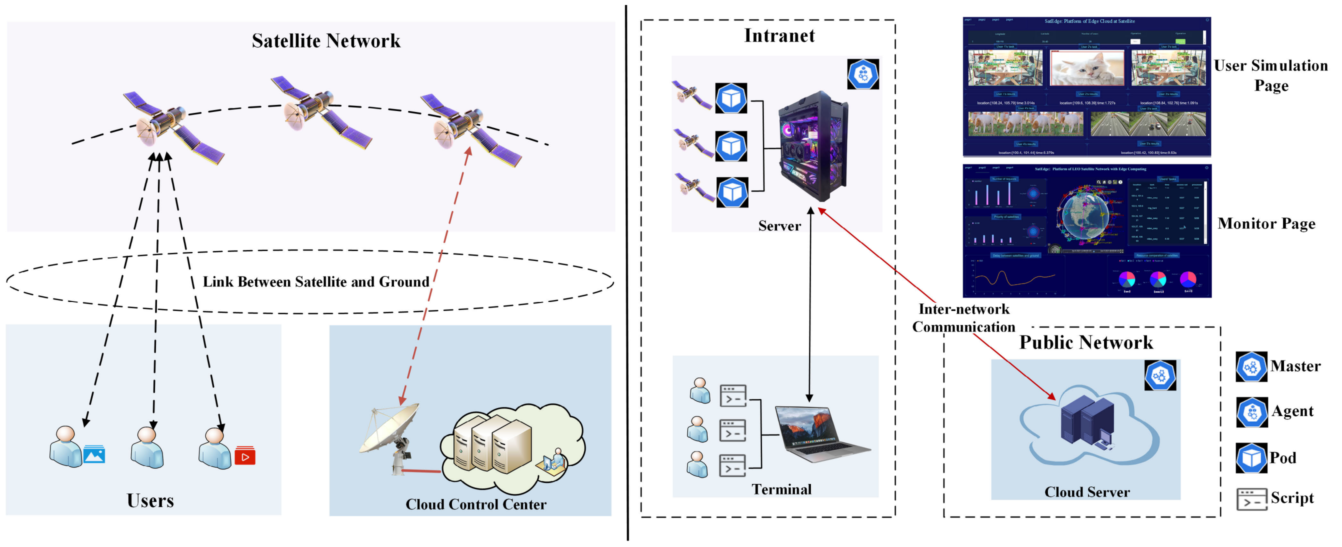


FIGURE 1. Network scenario and topology of SatEdge.

TABLE 1. Comparison between K8s, K3s and KubeEdge.

Feature	Sub-Feature	K8s	K3s	KubeEdge
Characteristics	Degree of Node Lightweighting	Low	Medium	High
	Degree of Resource Consumption of Edge Nodes	High	Medium	Low
	Degree of Installation Complexity	High	Low	Medium
	Degree of Decentralization of Edge Nodes	Low	Medium	High
Functions	Cloud-Edge Collaboration	Not supported	Not supported	Supported
	Edge Autonomy	Not supported	Not supported	Supported
	Cloud-Edge Deployment Across Subnets	Not supported	Not supported	Supported
	Massive Node Access	Not supported	Partially Supported	Supported
	Containerized Application Orchestration	Supported	Supported	Supported

The major contributions of this paper are summarized as follows:

- Comparative Analysis: We conduct a comprehensive comparison of various open-source tools, analyzing their characteristics and functions. This work serves as a valuable guide for researchers in satellite networks and aids cloud computing developers in selecting suitable platforms.
- SatEdge: We introduce SatEdge, a lightweight edge cloud platform based on KubeEdge specifically designed for satellite environments. SatEdge enhances the flexibility of satellite service provisioning and contributes to the overall management of the satellite system. We provide a comprehensive overview of SatEdge, including network scenarios, system deployment, system architecture, and software architecture. This fills a gap in existing research by providing a well-defined software architecture that guides the development process.
- OMS-AM Algorithm: Building upon our previous work, we propose the optimal microservice scheduling algorithm with adaptivity and mobility (OMS-AM). This algorithm aims to minimize latency, energy consumption, and task failure rate. It takes into account topology changes and propagation delay in the satellite network, focusing on addressing mobility challenges. OMS-AM considers both the task's path among satellites and ensures the successful transmission of results back to users.

Furthermore, based on the aforementioned three points, we introduce a series of experiments to verify the performance of SatEdge and OMS-AM.

II. SATELLITE-BASED LIGHTWEIGHT EDGE CLOUD ARCHITECTURE

In this section, we introduce SatEdge from the perspectives of scenario, topology, and architecture. The scenario and physical topology of our platform are shown in Fig. 1. And SatEdge's system architecture is shown in Fig. 2, which provides the design principle for software development, as shown in Fig. 3.

The scene of the satellite-terrestrial hybrid network is shown on the left side of Fig. 1. In space, satellites orbit in a certain pattern, and the constellation is dynamic. Owing to the EC diagram, every satellite has certain resources to handle user requests. The user may initiate an Internet service request, such as an image or video. Furthermore, the entire satellite system should be managed by a terrestrial control center, which can communicate with a certain number of satellites through station gateways. Managers can monitor satellite networks for things such as microservice deployment, resource allocation, and satellite status monitoring.

The topology of SatEdge is shown on the right side of Fig. 1. As previously mentioned, SatEdge is built based on KubeEdge, which extends container orchestration and device management to hosts at the edge. Therefore we built our platform with a cluster-based approach while taking advantage of the KubeEdge architecture's strength in edge computing. Specifically, we set up a cloud server in the public network to represent the terrestrial center and another server running satellites in the intranet. To simulate users, we used a laptop running functional scripts to generate users and tasks. In addition, we also developed two User Interface (UI) pages: a monitor page to supervise the whole system and a user simulation page to generate users and tasks.

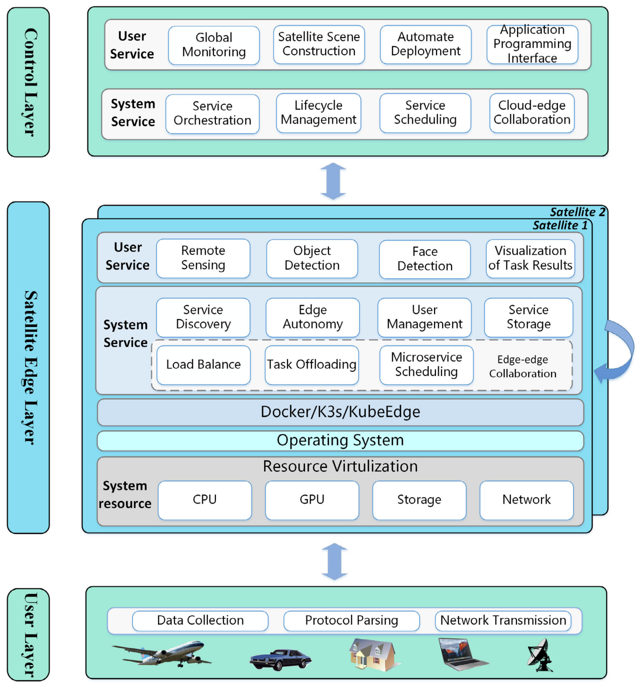


FIGURE 2. System architecture of SatEdge.

The system architecture mainly includes the control, satellite edge, and user layers. The control layer control and interact with satellite networks, manages them, and monitors their status. The satellite edge layer is primarily deployed on each satellite, and the resources of the satellite can be managed and scheduled. The user layer is used to transmit user requests to the satellite edge layer and provide feedback on the processing results to the user. In particular, we partitioned the user service and system service at the control layer and edge layer. The request method of the user service is initiated by the user who waits for a response. System services refer to services provided for the internal parts of the system, and their service targets are the various parts of the system. User services are targeted at users or terminals to access services. The system has two types of user: general users and administrators. The former's requirements are to access specific business applications, such as object detection, whereas the latter's requirements are to manage the entire system, such as monitoring data.

The components of the control layer are introduced as follows.

- Global monitoring: This is designed for administrators. We have also provided UI pages, as shown in the Fig. 1.
- Satellite Scene Construction: We used the STN scene to validate the performance of SatEdge.
- Automated deployment: Users can deploy tasks by uploading resource configuration files.
- Service Orchestration: Automating the management and scheduling of multiple containerized services through orchestration tools.
- Life cycle management: Perform various operations and management of containerized applications, such as image building, deployment, scaling, and monitoring.
- Service Scheduling: It deploys services to available nodes based on the resource requirements of the application.
- Cloud-edge Collaboration: The collaborative work between cloud and edge computing devices improves task processing efficiency.

Some components of the satellite edge layer are introduced as follows.

- Service discovery: Automatic identification and management of containerized applications to ensure service availability and reliability.
- Edge autonomy: Edge nodes have a certain degree of autonomy and can perform task scheduling, fault recovery and other operations without the need for cloud intervention.
- User management: It manages users accessing the system and record their information.
- Edge-edge Collaboration: By enabling efficient collaboration between edge nodes, the system efficiency and response speed can be improved.

To further demonstrate SatEdge's functions, we organized the platform's functional metrics in Table 2.

The SatEdge architecture is shown in Fig. 3. SatEdge's architecture is primarily divided into clouds and edges. The cloud represents the terrestrial control center, and the edge is responsible for receiving instructions from the cloud and running the containers. Moreover, the blue components are K8s elements, such as API and DataBase. The yellow components are KubeEdge elements, such as CloudCore and EdgeCore. The reds are the components that we develop. Subsequently, some components of SatEdge were introduced.

- Terrestrial Cloud Control System: It helps administrators manage the SatEdge system. In particular, we developed a demo interface based on VUE to display services and visualize our platform's situation.

TABLE 2. Functions of SatEdge.

User Function		System Function			
For General Users	**For Administrators**	**Edge-edge Collaboration**	**Satellite Scene Construction**	**Service Lifecycle Management**	**Others**
Object detection	Global monitoring	Task offloading	Satellites' topology management	Pull images and run containers	Edge autonomy
Face detection	Application programming interface	Microservice scheduling	Satellites' link management	Service development	User management
Remote sensing	Automate deployment	Load balance	Satellites' resource management	Service Discovery	

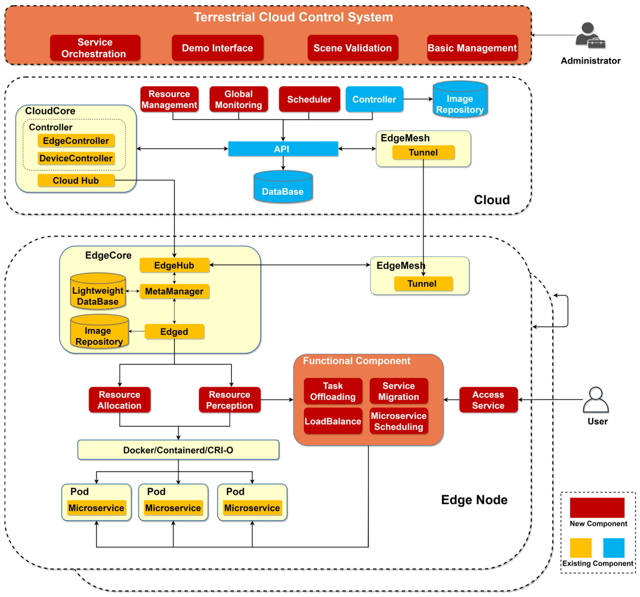


FIGURE 3. Software architecture of SatEdge.

- Service Orchestration: Based on customized templates, the services provided by satellites to the outside world are orchestrated on-demand.
- Scene Validation: We used the STN scene to validate SatEdge's performance. The way in which we build the STN scene can be referred to in our last work [27]. We obtained the satellite motion trajectory through Satellite Tool Kit (STK) simulation software, and obtained the access satellite and link delay through subsequent processing. We also imported this information into the SatEdge platform through the Traffic Control tool provided by the Linux system.
- Basic Management: It manages satellite node resources such as computation and storage, as well as link latency.
- Resource Management: It collects and determines every satellite node's CPU and Memory resources.
- Functional Components: This component provide service that help satellites process tasks more efficiently.
- Resource Allocation: It allocates satellite nodes' CPU or Memory resources.
- Resource Perception: This perceives the computing resources of satellites and inter-satellite link status.
- Access Service: It records information about the user and the task, such as the user's location and the type of task.
- Functional Components: After the users send tasks to this system, our platform uses offloading or load-balancing strategies to process the tasks. These strategies are described in [27].

III. MICROSERVICE MODEL AND PROBLEM FORMULATION

In this section, the microservice model and the problems that we want to solve are introduced. First, we introduce the mobility in satellite networks, and then further elaborate on the problems brought about by the dynamic changes in topology in a satellite network that integrates microservice architecture. Finally, the scenario modeling and the problems to be solved are presented through mathematical formulas.

A. MOBILITY IN STN

Satellite nodes have undergone rapid changes. When analyzing a satellite scene, the satellite's changes can be divided into several time slots at a certain sampling frequency, and the satellite constellation can be viewed as static in each time slot. As shown in Fig. 4, for ground users, the satellite topology undergoes different changes in consecutive time slots of T1, T2, and T3. From T1 to T2, the access satellite remains the same, but the inter-satellite delay changes owing to the satellite's movement. From T2 to T3, there is a significant change in topology, and the access satellite also changes.

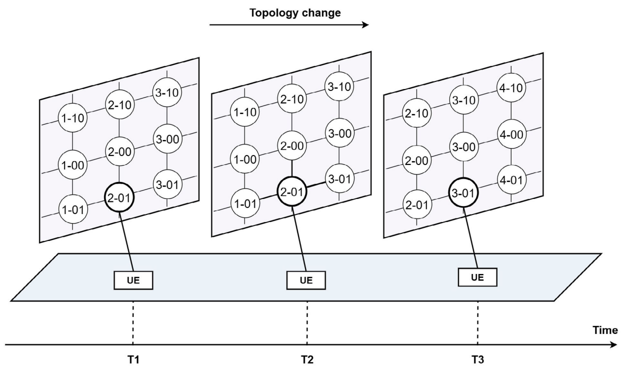


FIGURE 4. Mobility in STN.

B. STN SCENARIOS WITH MICROSERVICE ARCHITECTURE

Many business units, such as services, operate on satellites waiting to be requested. Traditionally, every service runs as a whole on a satellite, but a whole service can be divided into a series of lightweight and self-contained components by the microservice architecture. In this manner, a user request is executed by different satellites, which can formulate a workflow.

To illustrate the impact of mobility on tasks for users in a satellite network with a microservice architecture more clearly, we have drawn Fig. 5. The scene is divided into two parts: the task handling process and the task result return process. The first process is represented by a orange arrow, and the second process is represented by a blue arrow.

- The task handling process refers to the process in which the user sends the task to the satellite network through satellite access and the satellites collaborate to process it.
- The task result return process refers to the process of returning the result to the user through inter-satellite forwarding.

Fig. 5 illustrates the changes in the direction of the task owing to the mobility of the satellite during task processing. Where the solid line in the figure represents the scheduling of the workflow by the system when the topology structure is unchanged. The dashed lines indicate that the scheduling process is affected by satellite mobility. According to this strategy, it needs to be rescheduled, and the direction of the adjusted task flow is shown on the right side of Fig. 5 (a). Fig. 5 (b) illustrates how to successfully return the task result to the user if there are changes in the satellite topology structure when the task result is returned.

C. PROBLEM FORMULATION

In this section, we present a microservice scheduling problem using mathematical formulations. The user tasks that we are considering are related to Internet services, such as object detection and facial recognition. The tasks for which users initiate requests primarily involve videos or images of a certain size. The processing results for these types of tasks are usually in the form of text output; therefore, the size of the task processing results can be ignored.

When tasks are executed, the proposed algorithm schedules a workflow path among the available satellites and microservices to minimize total latency, total energy consumption, and task failure rate. The total latency comprises the following parts: transmission latency from the ground to satellites T^{trans}_{gnd}, propagation latency from the ground to satellites T^{prop}_{gnd}, propagation latency between satellites T^{prop}, transmission latency between satellites T^{trans}, and computation latency T^{comp}. The i-th satellite is V_i, and V_1 represents the access satellite. Suppose there is a whole business application that can be divided into microservices $M_1, M_2, \ldots, M_j$ components according to functions, and these microservices are distributed in the satellite networks. A task is always received by access satellite V_1 first, and after the task is processed by M_1, the algorithm decides whether it is processed by M_2 in V_1 or M_2 in other satellites. Thus, this scheme can form a scheduled workflow, namely path P: $p_{1,1}$, $p_{2,2}$, …, $p_{i,j}$. This implies that there are p satellite nodes in the network. $p_{i,j}$ means a task is processed by the j-th microservice component of the i-th satellite. The communication link between satellites v_i and v_{i+1} is denoted as $L_{i,i+1}$.

γ_i denotes the processing density in cycles per bit of satellite i. f_i^j denotes the computing power assigned to M_j. m_{j-1} is the incoming data size of M_j, and m_0 is the initial task size. The output data size is represented as m_j, and m_j is sent to its successor satellite node for further processing. The transmission rate between satellites is R^{ISL}. The transmission latency between satellite $V_{i'}$ and another satelliteV_i can be expressed as $\frac{m_j}{R^{ISL}_{i',i}}$. $T^{prop}_{i',i}$ is the propagation delay between satellite $V_{i'}$ and another satelliteV_i.

Task processing delay is an objective that we want to minimize. Based on the aforementioned mathematical symbols, we define the total computing and transport delay in the task processing process as T^{total}. γ_i denotes the processing density in cycles/bit. f_i^j denotes the computing power assigned to Mj.

$$\begin{aligned} T^{total} &= T^{trans}_{gnd} + T^{prop}_{gnd} + T^{comp} + T^{trans} + T^{prop} \\ &= T^{trans}_{gnd} + T^{prop}_{gnd} + \sum_{i,j}^{p} T_{p_{i,j}} + \sum_{i,j}^{p-1} T_{L_{p_i,p_{i+1}}} \\ &= \frac{m_0}{R_{gnd}} + \tau_{gnd} + \sum_{i,j}^{p} \frac{\gamma_i m_j}{f_i^j} + \sum_{i,j}^{p}\left(\frac{m_{j-1}}{R^{ISL}_{i,i+1}} + \frac{d_{i,i+1}}{c}\right) \end{aligned} \tag{1}$$

Another objective of our algorithm is to minimize energy consumption, which is defined as E^{total}. Energy consumption mainly involves three processes: transmission of tasks from the ground to access satellites, computing tasks on satellite E^{comp} and transmitting tasks and their parametersE^{trans} between satellites. ϵ_i represents the energy cost of the satellite V_i per CPU cycle. P_{ISL} is the transmission power between satellites.

If a task's size is W_n, the computation cost energy that the satellite V_i processes for this task can be defined as (2).

$$E^{comp} = \epsilon_i W_n \gamma_i \tag{2}$$

The energy cost that the ground transmits tasks to access satellites could be defined as (3).

$$E^{tans}_{gnd} = \frac{W_n P_{gnd}}{R_{gnd}} \tag{3}$$

The energy cost of satellite V_i transmitting to satellite V_{i+1} could be defined as (4).

$$E^{tans} = \frac{W_n P_{ISL}}{R^{ISL}_{i,i+1}} \tag{4}$$

Then, the total energy cost ofthe computation and transmission can be denoted as (5).

$$\begin{aligned} E^{total} &= E^{trans}_{gnd} + E^{comp} + E^{trans} \\ &= E^{trans}_{gnd} + \sum_{i,j}^{p} E_{p_{i,j}} + \sum_{i,j}^{p-1} E_{L_{p_i,p_{i+1}}} \\ &= \frac{m_0 P_{gnd}}{R_{gnd}} + \sum_{i,j}^{p} \epsilon_i m_j \gamma_i + \sum_{i,j}^{p-1} \frac{m_j P_{ISL}}{R^{ISL}_{i,i+1}} \end{aligned} \tag{5}$$

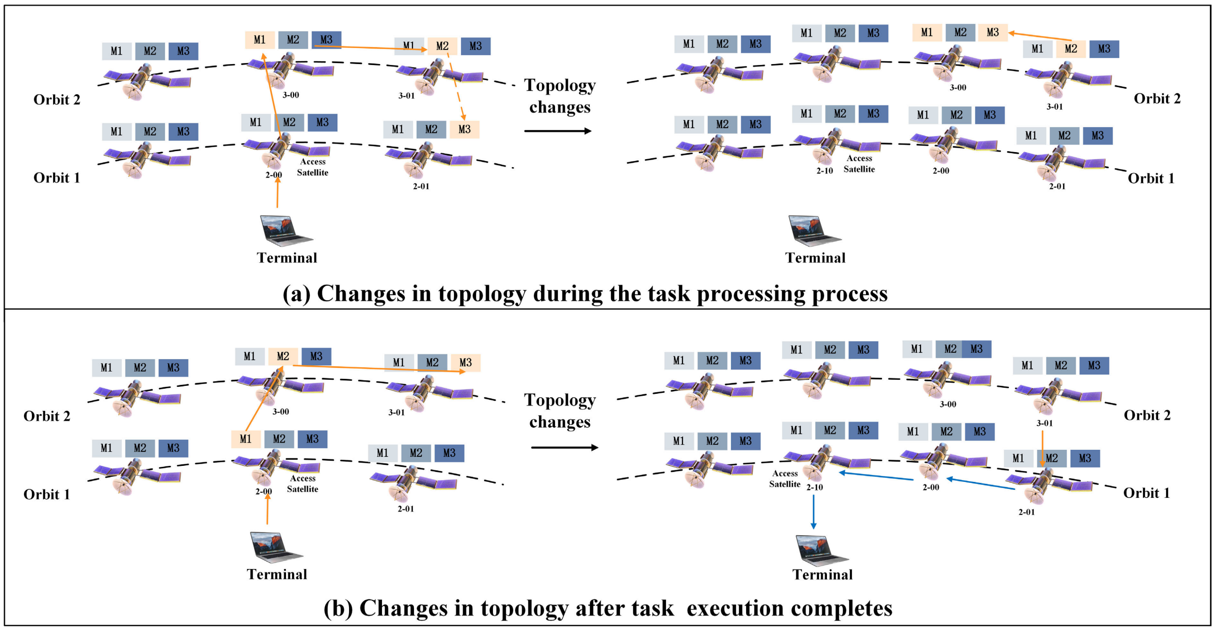

Orbit 2
Orbit 1
M1
M2
M3
3-00
3-01
2-00
2-01
2-10
Access Satellite
Terminal
Topology changes
(a) Changes in topology during the task processing process
(b) Changes in topology after task execution completes

FIGURE 5. Mobility in STN scenarios with microservice architecture.

As illustrated at the beginning of this section, the size of the task results can be ignored; therefore the return of task results does not consume energy. The objective of the proposed algorithm is to minimize both the T^{total} and the E^{total}.

IV. ALGORITHM DESIGN

In this section, we introduce the proposed algorithm, namely, OMS-AM, and other algorithms for comparison in the experiment. The designed algorithm can minimize both end-to-end delay and energy consumption. The other four compared algorithms are optimal microservice scheduling with adaptivity (OMS-A), non-microservice scheduling with adaptivity and mobility (nonM-AM), and non-microservice Scheduling (nonM). OMS-AM and OMS-A are both based on a microservice architecture. But OMS-A does not consider satellites' mobility when the result is returned to users. The nonM-M and nonM algorithms are not based on the microservice architecture, that is to say, these strategies consider a service as a whole. When the result must be sent to users, nonM-M considers the satellites' mobility. While nonM will not take this into consideration, a task result comes back the way it come.

The first three algorithms are designed based on microservice architecture. We build a mesh model referring to [28], [29] to describe these three algorithms as shown in Fig. 6. We adapt this model to the STN and improve the performance of satellite system.

A. OPTIMAL MICROSERVICE SCHEDULING WITH ADAPTIVITY AND MOBILITY (OMS-AM) ALGORITHM

OMS-AM was improved based on the Optimal Microservice Scheduling with Adaptive Link Changes (OMS-ALC) proposed in our previous work [27]. Some characteristics

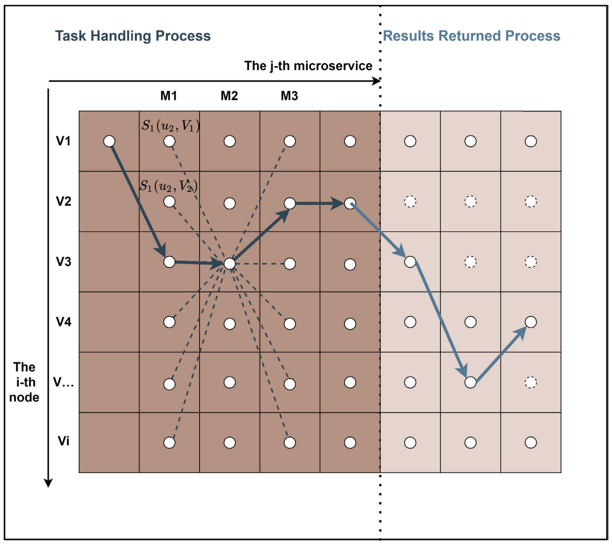


FIGURE 6. Mesh model of OMS-AM.

of OMS-AM are in line with those of the OMS-ALC. For example, both algorithms take advantage of the flexibility of microservices to formulate workflows. Second, they focused on a globally optimal solution among the available microservices and satellites. However, compared to OMS-ALC, OMS-AM has three advantages.

- The number of satellites involved has increased. OMS-ALC only considers five satellites, whereas OMS-AM considers all satellites that may be involved.
- The mobility of satellite networks requires careful consideration. We consider the influence of mobility not only in the task handling process but also in the results return process.

- These results are more persuasive. The performance of OMS-AM is verified using richer metrics. In [27], the results of experiments verified the OMS-ALC using delay tests. However, in this paper, we show the results in terms of delay, energy consumption, and task success rate.

OMS-AM consists of two processes: the task handling process and the results return process. In the task handling process, tasks are sent to the network, and the algorithm schedules task requests to formulate a workflow among satellites according to collectible indices in the real production environment. In this process, the OMS-AM considers the mobility of satellites and works with adaptive link change characteristics. In the results return process, after a task is processed, the satellite topology may change because of task execution time delays. Thus, the results must be forwarded to the users.

1) TASK HANDLING PROCESS

To represent the OMS-AM well, we built a 2D mesh model, as shown in the Fig.6. The horizontal axis represents microservice components $M_1, M_2, \ldots.$, and M_0 represents the start of the workflow. The vertical axis represents available satellite nodes $V_1, V_2, \ldots.$, and V_1 represents the access satellite. In addition, a cell in the mesh is $S_j^{total}(m_j, V_i)$, which represents the total cost from the task request sent to m_j processed completely by M_j. Moreover, the cost is composed of five parts: CPU cores, CPU utilization, number of parallel tasks, inter-satellite propagation delay, and energy consumption, which are denoted as $core_i$, $util_i$, $para_i$, $delay_i$, and $energy_i$, respectively.

$$S_j(m_j, V_i) = core_i + util_i + para_i + delay_i + energy_i \quad (6)$$

These five parameters determine the cost value, and to normalization them, we need to standardize them. Because we want the minimal cost sum S_j, we perform the normalization as follows: If there exists a group of parameters in terms of one performance, $X = [x_1, x_2, \ldots, x_i]$, and the larger the x_i is, the better. We normalize x_i as 7. For example, $util_i$, $para_i$, $delay_i$, and $energy_i$.

$$x\_norm_i = \frac{x_i - \min(X)}{\max(X) - \min(X)} \quad (7)$$

If the smaller the x_i is, the better, we normalize x_i as 8. Such as $core_i$.

$$x\_norm_i = \frac{\max(X) - x_i}{\max(X) - \min(X)} \quad (8)$$

Then, we have the same recursion method referring to the (13) defined in [27] to the final solution. In this work, we use $S_j^{min}(m_j, V_i)$ to replace $S_j^{max}(m_j, V_i)$, and $S_j^{total\_min}$ to replace $S\_total_j^{max}$.

When $j = 1$, the initial S_1^{total} is defined as follow:

$$S_1^{total\_min} = \min\{S_1(m_1, V_i)\} \quad (9)$$

According to the above computational process, we can obtain the minimal cost of the workflow. Once a task comes in, OMS-AM projects a reasonable workflow to minimize the cost sum. After each recursion, the best satellite V_i will be selected and added to the available satellite set V. And the m_j will be assigned according to V. Similarly, when the workflow executed by some satellite changes owing to the topology of satellites or the delay of inter-satellite links, OMS-AM abandons the S_total_j, and calculates a new workflow according to the (13) defined in [27]. Similarly, a mesh model can still be established, as shown in Fig. 6.

2) RESULTS RETURN PROCESS

When the task is completed, the results must be returned to the user. Because of the computing time, transmission delay, and propagation delay, it is likely that the topology of the satellite will change after the micro-service module has finished processing a certain task. If the satellite is returned along the original path, the access satellite directly connected to the user will not be the satellite in the previous time slot. Therefore, it is necessary to first determine whether the satellite topology has changed.

If the topology changes during the return, the return portion of the satellite receipt algorithm that obtains the task processing results obtains the schedule path with the minimum transmission delay using the shortest path method to ensure that the task results are successfully returned to the user. The return of a mission can be thought of as a graph structure. From the final satellite, V_i is the starting point, and the end point is the access satellite that the user connects to in real time. As shown in Fig. 6, the return section only considers the topology of the satellites and, not the micro-services and their connections. By default, each satellite has a forwarding capability. We consider the return process to be a directed acyclic graph, $G = (V, E)$. V represents a set of nodes, that is, each satellite node. To build G, we also set a series of parameters. The source node is determined by the task handling process, and we define it as V_s. The destination node is access satellite V_d. Num_V^{orbit} is the number of node V's orbit. Num_V^{id} is the number of node V in one orbit. In the topology $G = (V, E)$, $V(G) = \{V_1, V_2, \ldots, V_n, \ldots, V_N\}$, we can compute the N according to (11).

$$N = \left| Num_{V_s}^{orbit} - Num_{V_d}^{orbit} + 1 \right| \times \left| Num_{V_s}^{id} - Num_{V_d}^{id} + 1 \right| \quad (10)$$

E represents the set of edges, and each edge $(u, v) \in E$ has a non-negative weight $w(u, v)$, which is expressed in terms of inter-satellite delay. The set S is used to store nodes that currently have the lowest delay. From StartV_i, V_i joins the priority queue and sets its lowest delay $d(V_i)$ to zero. From Start V_i, traverses all adjacent satellite nodes $v \in V$, calculates the distance from V_i to v, that is, $d(u) + w(u, v)$, if this distance is smaller than the shortest distance of the current record $d(v)$, then update the value of $d(v)$ and add V to the S. The core algorithm is expressed as (11).

$$d(v) = min\{d(v), d(u) + w(u, v)\} \quad (11)$$

The flow of the results return is illustrated in Fig. 6. We can see that it starts at node, that is the final satellite computing task. It then determines finds the minimal end-to-end delay in the workflow. This process ensures that the results can be successfully sent to the users.

3) THE WHOLE PROCESS

And the pseudocode of whole OMS-AM is shown as follow. If there are m microservices and n available satellites, it can be concluded that the time complexity is $O(mn)$.

Algorithm 1 (OMS-AM) Algorithm

```
Input: Available satellites set $V(i = 1, 2, , . . . , I)$, Task
  $m_j (j = 1, 2, . . . , J)$, set S
Output: Target satellites set $V_{coming}$ of tasks handling
  process, Target satellites set $V_{back}$ of results returned
  process.
  In the task handling process
  for $j = 1$ to $J$ do
    Normalize $util_i$ $para_i$ $delay_i$ $energy_i$ referring to (4)
    Normalize $core_i$ referring to (5)
    Execute the same steps referring to [27]
  end for
  In the results return process
  get the $V_s$ from $V_{coming}$
  if topology of satellite network stays the same then
    return the results according to $V_{coming}$
  else
    construct the graph G according to (10), initialize $w$
    according to inter-satellite delay.
    while len(S)<len(G) do
      update $d(V)$ according to (11)
      add $V$ to the priority queue S
      record $V$'s previous node in a dictionary D
      if status of link changes then
        reconstruct the graph G
      end if
      compute $V_{back}$ according to S and D
    end while
  end if
```

The task handling process and the results return process can both be modeled as a graph G. A comparison is presented in Table 3. Process 1 represents the task handling process, and Process 2 is the results returned process. In Process 1, the source node is fixed because the process starts from the access satellite. In Process 2, the algorithm starts with the final satellite that has computed the results. In terms of destination, Process 1 is unfixed. The workflow is scheduled using real-time parameters, and the network topology is changing. In terms of structure, the graph Process 1 is acyclic because a satellite can forward a task to another, and it can also compute the task locally. In Process 2, a satellite is unlikely to forward the results to itself. In process 1, the weights are determined by five parameters, as shown in (6). In process 2, no tasks need to be computed, and only the results need to be forwarded, so the weights are decided by the inter-satellite delay.

TABLE 3. Comparison of two processes.

	Process 1	Process 2
Source node	fixed	fixed
Destination	unfixed	fixed
Structure of G	acyclic	cyclic
Weights of G	five parameters	one parameters

B. OTHER COMPARED ALGORITHMS

The Other three compared algorithms are OMS-A, nonM-AM, and nonM. OMS-A. The OMS-A is the algorithm proposed in [27]. The difference between OMS-AM and OMS-A lies in the results return process. OMS-A does not use the shortest path method to compute the return path, and forwards the results according to the original path. In other words, if the topology changes after the tasks are computed, the results cannot be sent to the users.

The nonM-AM algorithm does not use a microservice architecture. Therefore, every service runs on the satellites as a whole. In this case, the definition of M_j does not exist. Services among the available satellites have no connections. We use the same five indexes to judge which satellite is the best to process a task according to (6). The results returned process by this algorithm are similar to those of OMS-AM.

The nonM algorithm has also been designed for non-microservice applications. It is similar to nonM-AM, but it does not consider the resultsof the returned process.

V. EXPERIMENT

In this section, we show the UI pages of our platform and the performance of some algorithms.

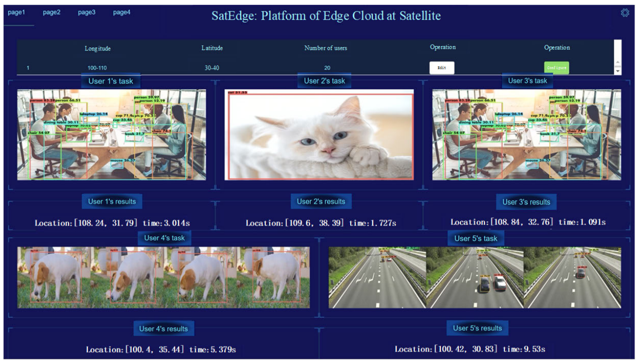
A. FUNCTION VERIFICATION OF SATEDGE

We developed some UI pages to verify the functions of SatEdge as shown in Fig. 7 and 8. In Fig. 7, when simulating users who request service on the satellite system, we can set a series of users' information, such as location and number. If the number of users is greater than five, the page will choose five users and their tasks to display. This page also visualizes the results of object detection tasks. We can observe the processing time for each task. Our object detection service can process images and videos. The first three tasks shown in the figure are images and the last two tasks are videos.

The monitoring page is illustrated in Fig. 8. In the center of Fig. 8, there isa dynamic 3D demo of satellites and the earth. On this page, we can monitor all the information of the satellite system. For the satellite network, this page can display resource comparisons such as CPU, memory, and utilization. We can also observe the priority of the satellite nodes. In addition, we can monitor some details of the users' tasks. First, the page contains the number of requested services. Second, it tracks every task's execution information, as shown in the users' tasks section.

B. PERFORMANCES OF ALGORITHMS

We conducted a series of experiments to verify the performance of the OMS-AM. The configuration of the cloud



	Longitude	Latitude	Number of users	Operation	Operation
1	100-110	30-40	20	Edit	Configure

Location: [108.24, 31.79] time: 3.014s

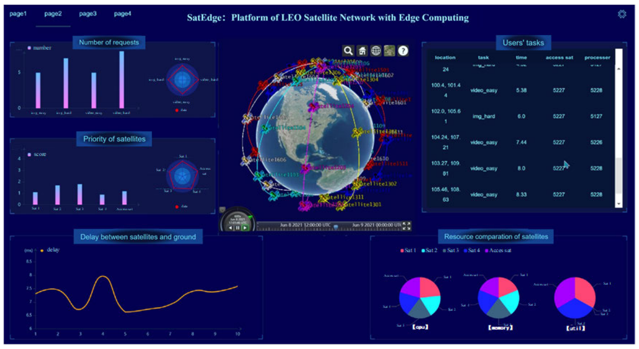
Location: [109.6, 38.39] time: 1.727s

Location: [108.84, 32.76] time: 1.091s

Location: [100.4, 35.44] time: 5.379s

Location: [100.42, 30.83] time: 9.53s

FIGURE 7. The user simulation page of SatEdge's UI.


page1
page2
page3
page4
SatEdge: Platform of LEO Satellite Network with Edge Computing
Number of requests
Priority of satellites
Users' tasks
Delay between satellites and ground
Resource comparation of satellites
Sat 1
Sat 2
Sat 3
Sat 4
Acces sat

location	task	time	access sat	processer
100.4, 101.44	video_easy	5.38	5227	5228
102.0, 105.61	img_hard	6.0	5227	5127
104.24, 107.21	video_easy	7.44	5227	5226
103.27, 109.81	video_easy	8.0	5227	5228
105.46, 108.63	video_easy	8.33	5227	5228

FIGURE 8. The monitoring page of SatEdge's UI.

server is an Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50 GHz, and the server running satellite nodes is an Intel(R) Xeon(R) W-2123 CPU @ 3.60 GHz. As for the open-source tools: EdgeMesh version 1.12.0, KubeEdge version 1.8.2, and K8s version 1.21.0. The constellation settings and the parameters mentioned above are listed in Table 4. Our parameter configurations were informed by precedent studies, as explicated in the introduction section. In our study, parameters γ_i and ϵ are constant after the simulation initiation, impacting the overall energy consumption magnitude. On the other hand, parameters f_i, P_{ISL}, and W_n are dynamic and exert a substantial influence on performance metrics, while it's important to note that our algorithm considers global optima, ensuring its robust performance in the research scenario.

The user task we used is a universal object detection application that is based on a neural network model. For the microservice application design, we used the typical neural network written in Keras with six convolution layers. In addition, we trained the neural network on an open-source dataset and achieved an accuracy of approximately 90% accuracy. After the training was complete, we divided the network construction into three parts to build three connected microservices. At the beginning, each microservice only needs to load a part-trained weight to fit its construction.

Next, we will introduce the performance of OMS-AM algorithm.

TABLE 4. Parameters of experiments.

Parameters	Value
Constellation type	LEO
Altitude of Constellation	508Km
Number of orbits	30
Satellites per orbit	60
f_i	1GHz-4GHz (uniform distribution)
P_{ISL}	30mW-120mW (uniform distribution)
W_n	1Mbit-5Mbit (uniform distribution)
ϵ	4J/Ghz
P_{gnd}	4W
γ	1000cycle/bit
R_{ISL}	10Gbps
R_{gnd}	200Mbps
τ_{gnd}	1.5-2.15ms

Based on the four test dimensions, we conclude that OMS-AM achieves the best performance compared to the other three algorithms. The OMS-AM achieved the lowest end-to-end delay, task failure rate, energy consumption, and total normalized cost. In addition, we found that strategies based on microservice architecture are better.

The average delay varies with the number of tasks, as shown in Fig. 9. With an increase in task requests, the average delay for all strategies increase. However, the growth rate of the OMS-AM is the lowest. The algorithms based on the microservice architecture, colored blue and purple, can achieve a lower end-to-end delay. This result proves that the microservice architecture is more efficient, because the algorithm using microservice architecture has more flexibility in task processing. Moreover, the OMS-AM proposed in this paper is better than the algorithm proposed in our previous work [27], which is colored in purple. When the number of tasks is 60, the average delay of the OMS-AM is approximately 14% lower than that of the OMS-A. Because the OMS-AM is based on the global optimal characteristics of the OMS-A algorithm, and also considers the problems caused by the network mobility in the task return process to prevent the terminal from receiving the task result timeout due to the satellite topology changes.

In Fig. 10, we show the impact of the algorithms' mobility on task failure rates. Both OMS-AM and nonM-AM have strategies for coping with the results return process, so they have the lowest number of failed tasks. The nonM algorithm considers neither the microservice architecture nor the results returned process, so the task delay would be longer, and it is likely to meet topology changes. This is why the line of nonM shakes more violently than the others do. In addition, the algorithms that consider the results return process, have a lower task failure rate.

The energy consumption of the four algorithms is shown in Fig. 11. These four lines were grouped into two categories. OMS-AM and OMS-A are based on a microservice architecture, and their energy costs are lower than those of nonM-AM and nonM. Based on these results, we can conclude that microservices enable satellite networks to operate

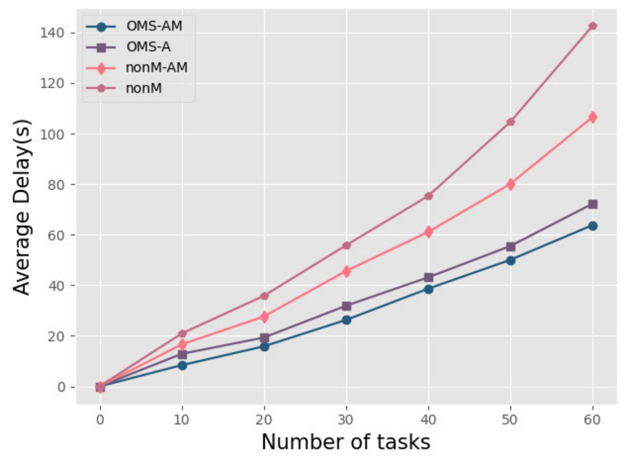


FIGURE 9. The average delay varies with number of tasks.

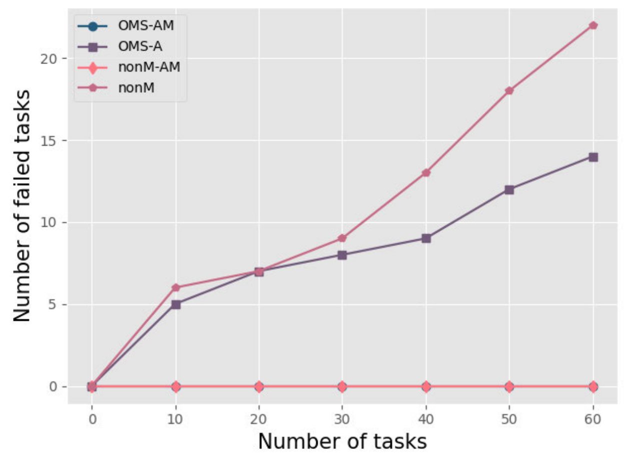


FIGURE 10. Number of failed tasks varies with number of tasks.

more efficiently. This is because the algorithms consider computational and transmission energy consumption when searching for the global optimal solution.

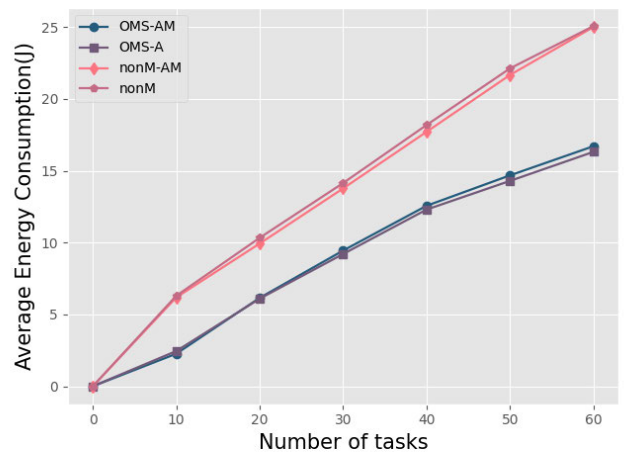


FIGURE 11. Energy consumption varies with number of tasks.

Moreover, we compare the sum of the normalized costs, as defined in (6) after normalization. In Fig. 12, the trends are similar to the comparison of the energy consumption. Strategies that use microservices incur lower costs in terms of computation, transmission, and energy.

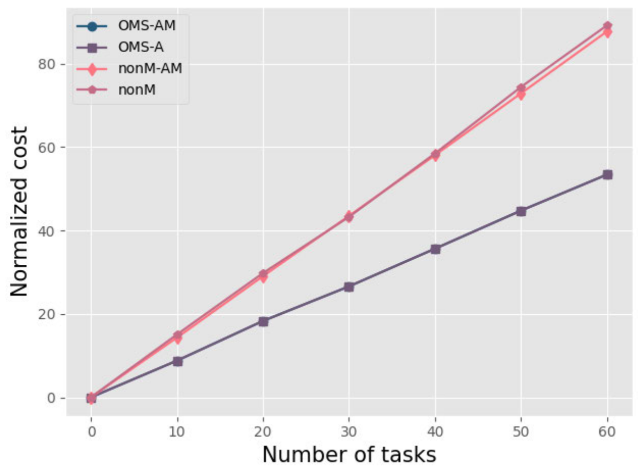


FIGURE 12. Sum of normalized cost varies with number of tasks.

From these four experimental comparisons, we can conclude that the OMS-AM is the most efficient algorithm. Because it has the lowest delay and task failure rate, its energy consumption and cost sum are also low.

VI. CONCLUSION

This study makes three main contributions. First, we compare these open-source tools in terms of characteristics and functions, which would allow more researchers to know about container and cluster technology. Next, we developed a satellite edge cloud platform based on KubeEdge. SatEdge increases the flexibility of satellite service provision and helps manage the entire satellite system. We introduce SatEdge from network scenarios and system deployment as well as from system architecture and software architecture. According to our survey, most relevant research lacks concrete software architecture to guide development. Third, we extended the mobility of the microservice scheduling algorithm based on our previous work. Moreover, OMS-AM not only considers a task's coming path among satellites but also ensures that the result can be sent back to users. Then, we introduce a series of experiments to verify the performance of the OMS-AM on our platform. The results show that the OMS-AM outperforms the other solutions in terms of delay, energy consumption, and success rate. Our experiments validated the effectiveness of the microservice architecture in satellite networks. Meanwhile, more improvements can be done in the future. For example, we can modify the KubeEdge native module to make the platform more adaptable to STN. Besides, we can design algorithms by considering more issues related to the mobility of the satellite network, such as connection reliability.

REFERENCES

[1] *Ericsson Mobility Report*. Accessed: Nov. 2022. [Online]. Available: https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf

[2] J. Zhang, X. Zhang, P. Wang, L. Liu, and Y. Wang, ''Double-edge intelligent integrated satellite terrestrial networks,'' *China Commun.*, vol. 17, no. 9, pp. 128–146, Sep. 2020.

[3] Y. Wang, J. Zhang, X. Zhang, P. Wang, and L. Liu, ''A computation offloading strategy in satellite terrestrial networks with double edge computing,'' in *Proc. IEEE Int. Conf. Commun. Syst. (ICCS)*, Dec. 2018, pp. 450–455.

[4] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, ''A survey on mobile edge networks: Convergence of computing, caching and communications,'' *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[5] R. Xie, Q. Tang, Q. Wang, X. Liu, F. R. Yu, and T. Huang, ''Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues,'' *IEEE Netw.*, vol. 34, no. 3, pp. 224–231, May 2020.

[6] A. Ahmed, S. Abdullah, S. Iftikhar, I. Ahmad, S. Ajmal, and Q. Hussain, ''A novel blockchain based secured and QoS aware IoT vehicular network in edge cloud computing,'' *IEEE Access*, vol. 10, pp. 77707–77722, 2022.

[7] A. Ahmed, S. Abdullah, M. Bukhsh, I. Ahmad, and Z. Mushtaq, ''An energy-efficient data aggregation mechanism for IoT secured by blockchain,'' *IEEE Access*, vol. 10, pp. 11404–11419, 2022.

[8] S. Chen, S. Sun, and S. Kang, ''System integration of terrestrial mobile communication and satellite communication—The trends, challenges and key technologies in B5G and 6G,'' *China Commun.*, vol. 17, no. 12, pp. 156–171, Dec. 2020.

[9] Y. Jia, J. Zhang, P. Wang, L. Liu, X. Zhang, and W. Wang, ''Collaborative transmission in hybrid satellite-terrestrial networks: Design and implementation,'' in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[10] J. Wei, S. Cao, S. Pan, J. Han, L. Yan, and L. Zhang, ''SatEdgeSim: A toolkit for modeling and simulation of performance evaluation in satellite edge computing environments,'' in *Proc. 12th Int. Conf. Commun. Softw. Netw. (ICCSN)*, Jun. 2020, pp. 307–313.

[11] Z. Zhang, W. Zhang, and F.-H. Tseng, ''Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques,'' *IEEE Netw.*, vol. 33, no. 1, pp. 70–76, Jan. 2019.

[12] C. Pahl and B. Lee, ''Containers and clusters for edge cloud architectures—A technology review,'' in *Proc. 3rd Int. Conf. Future Internet Things Cloud*, Aug. 2015, pp. 379–386.

[13] S. Wang, Q. Li, M. Xu, X. Ma, A. Zhou, and Q. Sun, ''Tiansuan constellation: An open research platform,'' in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Sep. 2021, pp. 94–101.

[14] R. Liu, K. Guo, K. An, Y. Huang, F. Zhou, and S. Zhu, ''Resource allocation for cognitive satellite-HAP-terrestrial networks with non-orthogonal multiple access,'' *IEEE Trans. Veh. Technol.*, vol. 72, no. 7, pp. 9659–9663, Jul. 2023.

[15] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, ''Open issues in scheduling microservices in the cloud,'' *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, Sep. 2016.

[16] L. Cao, A. Merican, D. Z. Tootaghaj, F. Ahmed, P. Sharma, and V. Saxena, ''eCaaS: A management framework of edge container as a service for business workload,'' in *Proc. 4th Int. Workshop Edge Syst., Analytics Netw.*, Apr. 2021, pp. 73–78.

[17] *K8s: An Open-Source System for Automating Deployment, Scaling, and Management of Containerized Application*. Accessed: Oct. 2021. [Online]. Available: https://kubernetes.io/

[18] *K3s: Lightweight Kubernetes*. Accessed: Oct. 2021. [Online]. Available: https://k3s.io/

[19] *KubeEdge: Kubernetes Native Edge Computing Framework*. Accessed: Oct. 2021. [Online]. Available: https://kubeedge.io/

[20] A. Jeffery, H. Howard, and R. Mortier, ''Rearchitecting Kubernetes for the edge,'' in *Proc. 4th Int. Workshop Edge Syst., Analytics Netw.*, Apr. 2021, pp. 7–12.

[21] J. Liang, F. Liu, S. Li, and Z. Cai, ''A comparative research on open source edge computing systems,'' in *Proc. 5th Int. Conf. Artif. Intell. Secur. (ICAIS)*, New York, NY, USA, Jul. 2019, pp. 157–170.

[22] 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.

[23] K. Guo, K. An, B. Zhang, Y. Huang, X. Tang, G. Zheng, and T. A. Tsiftsis, ''Physical layer security for multiuser satellite communication systems with threshold-based scheduling scheme,'' *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5129–5141, May 2020.

[24] L. Yan, S. Cao, Y. Gong, H. Han, J. Wei, Y. Zhao, and S. Yang, ''SatEC: A 5G satellite edge computing framework based on microservice architecture,'' *Sensors*, vol. 19, no. 4, p. 831, Feb. 2019.

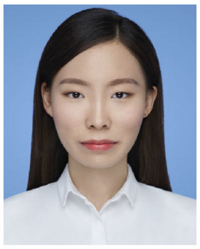
[25] L. Bao, C. Wu, X. Bu, N. Ren, and M. Shen, ''Performance modeling and workflow scheduling of microservice-based applications in clouds,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 2114–2129, Sep. 2019.

[26] A. Samanta and J. Tang, ''Dyme: Dynamic microservice scheduling in edge computing enabled IoT,'' *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6164–6174, Jul. 2020.

[27] Y. Huang and X. Zhang, ''Microservice scheduling for satellite-terrestrial hybrid network with edge computing,'' in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC Workshops)*, Aug. 2022, pp. 24–29.

[28] Q. Wu, Y. Gu, M. Zhu, and N. S. V. Rao, ''Optimizing network performance of computing pipelines in distributed environments,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 1–8.

[29] Q. Wu and Y. Gu, ''Supporting distributed application workflows in heterogeneous computing environments,'' in *Proc. 14th IEEE Int. Conf. Parallel Distrib. Syst.*, Dec. 2008, pp. 3–10.



YANZHE HUANG received the bachelor's degree from the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing. She is currently pursuing the master's degree with the Key Laboratory of Universal Wireless Communications, School of Information and Communication Engineering. Her current research interests include edge computing and satellite-terrestrial networks.

XING ZHANG (Senior Member, IEEE) is currently a Full Professor with the School of Information and Communications Engineering, Beijing University of Posts and Telecommunications, China. His research interests mainly include 6G wireless communications and networks, edge computing, satellite-terrestrial networks, cognitive radio and cooperative communications, big data, and the Internet of Things. He is a Senior Member of IEEE ComSoc and a member of CCF.

ZECHANG XU was born in Jiangsu, in 2000. He received the bachelor's degree from the University of Science and Technology Beijing. He is currently pursuing the master's degree with the Beijing University of Posts and Telecommunications. His research interests include satellite communication, cloud native, and mobile edge computing.