Pod-Orchestrated Dual Cluster Framework for Intelligent Task Offloading in Satellite-Ground Edge Networks

Syed Saqib Jamal
Department of Electronic Engineering
Jeju National University
Republic of Korea
saqib132413@gmail.com

Umar Mahmood
Department of Electronic Engineering
Jeju National University
Republic of Korea
umarmahmood637@gmail.com

Muhammad Asif
Department of Computer Engineering
Jeju National University
Republic of Korea
asif07@office.jejunu.ac.kr

Wang-Cheol Song*
Department of Computer Engineering
Jeju National University
Republic of Korea
philo@jejunu.ac.kr

***Abstract*—This paper proposes a Kubernetes-based dual-cluster framework for intelligent task offloading in satellite-ground edge networks. The system integrates 12 ground stations and 66 satellite nodes, leveraging containerized pods and real-time monitoring to optimize task placement based on CPU usage, network load, and line-of-sight availability. A custom Python controller dynamically enforces offloading and migration policies, while Prometheus and Grafana facilitate continuous performance tracking. Simulation results show that the framework reduces end-to-end latency by up to 35%, decreases CPU utilization on overloaded ground nodes by 43%, and improves task success rates to over 90% within the optimal offloading range. These outcomes validate the framework's scalability and resilience in managing dynamic workloads across heterogeneous terrestrial and non-terrestrial infrastructures.**

***Index Terms*—Kubernetes Orchestration, Satellite-Ground Networks, Task Offloading, Resource Management, Pod Scheduling**

I. Introduction

The rapid growth of Low Earth Orbit (LEO) satellite constellations integrated with terrestrial networks has expanded global connectivity and edge computing capabilities [1]. With rising IoT deployments in urban, rural, and remote areas, there is a need for responsive, scalable, and intelligent computational frameworks [2]. Terrestrial infrastructures alone cannot meet real-time, latency-sensitive demands, especially under dynamic space–ground conditions [3] [4]. Key challenges include variable CPU utilization, limited processing power, and intermittent satellite connectivity due to line-of-sight (LoS) restrictions. Overloaded nodes cause congestion and poor resource use [5], while current orchestration solutions lack adaptive decision-making [6], [7]. To address these limitations, our research aims to design and implement an advanced monitoring and offloading mechanism that facilitates intelligent, load-aware task scheduling across terrestrial and non-terrestrial clusters. We utilize Kubernetes-based container orchestration to dynamically manage tasks encapsulated in pods [8], assigning them to ground station (GS) or satellite (SAT) nodes based on realtime performance metrics, such as CPU usage, network load, and link conditions [9], [10]. By proactively balancing workloads, our approach seeks to minimize end to end latency and consistently deliver high quality service (QoS).

*Corresponding author: Wang-Cheol Song (philo@jejunu.ac.kr)

This paper presents several key contributions toward advancing containerized computing in integrated terrestrial-satellite networks. First, it introduces a novel dual cluster architecture that leverages Kubernetes orchestration across both ground-based and satellite nodes, enabling seamless task management in a heterogeneous network environment. Secondly, it proposes a load-aware task offloading algorithm capable of dynamically balancing workloads by considering realtime CPU utilization and satellite line of sight constraints, thereby optimizing task placement and system responsiveness. Third, the research develops an efficient pod lifecycle management strategy that ensures optimal resource usage through lightweight pod scheduling and automatic cleanup after task completion [5], [11]. Collectively, these contributions provide a scalable and latency-aware solution for reliable service delivery in future space-terrestrial computing systems [12]. The paper is organized as follows: Section 2 reviews related work on Kubernetes orchestration and task offloading in terrestrial–satellite networks. Section 3 presents the research model for CPU utilization and load balancing under CPU and link constraints. Section 4 describes the dual-cluster architecture, scheduling, and lifecycle management. Section 5 evaluates performance in task distribution, latency, and resource use. Section 6 concludes with future research directions.

II. Literature Review

Kubernetes is a central platform for managing containerized workloads, with extensive research on pod scheduling in multi-node clusters. Gao et al. classify scheduling algorithms into multi-objective and AI-driven strategies, highlighting their impact on cluster performance [11]. Zhang et al. apply deep reinforcement learning to edge-based Kubernetes scheduling, improving response times and load balancing [13]. Advanced techniques include gang scheduling for HPC workloads, as in AWS Batch Multi Node Processing Jobs on Amazon EKS [14], and the multi-cluster Containerized Workflow Engine (CWE) for scalable resource allocation [15]. Pod lifecycle management spanning Pending, Running, Succeeded, and Failed phases is vital for reliability [16]. For stateful applications, Ahmad et al. propose resilience-focused lifecycle strategies to enhance fault tolerance and resource efficiency [17]. Energy and deadline-aware frameworks balance effi-

ciency with task urgency [18], while fine-grained scheduling optimizes latency and throughput for HPC workloads [19].

III. Proposed System Model

Our proposed system, illustrated in Fig. 1, consists of a dual-cluster architecture leveraging Kubernetes-based container orchestration to efficiently manage computational tasks across integrated terrestrial–satellite networks. The simulation environment models 12 ground stations (GS) organized into the GSCluster and 66 satellite nodes forming the SATCluster. These values were chosen to reflect typical configurations in current Low Earth Orbit (LEO) constellations (e.g., Iridium) while remaining within Kubernetes scalability testing constraints. The framework is adaptable to varying numbers of GS and SAT nodes by adjusting cluster sizes and scheduling thresholds.

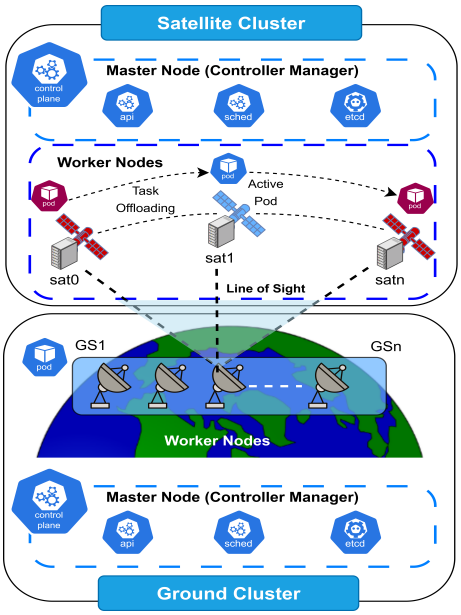

Fig. 1. System Model for Efficient Resource Utilization and load balancing across Kubernetes-based GS and SAT Clusters

At runtime, the SATCluster dynamically selects and maintains the 33 most relevant satellites, denoted as $S_{\text{active}} \subseteq S$, based on line-of-sight (LoS) availability and link quality metrics. Let $G = \{GS_1, GS_2, \ldots, GS_{12}\}$ represent the set of ground station nodes, and $S = \{SAT_1, SAT_2, \ldots, SAT_{66}\}$ represent the set of satellite nodes. At any given time, the active satellite cluster is dynamically selected such that $|S_{\text{active}}| = 33$. A computational task T, encapsulated in a Kubernetes pod, is initially scheduled to a ground station node $\text{GS}_i \in G$ if both CPU utilization and network load satisfy in (1).

$$\text{CPU}_{\text{GS}_i} \leq \text{CPU}_{\text{threshold}}, \quad \text{NL}_{\text{GS}_i} \leq \text{NL}_{\text{threshold}} \tag{1}$$

If these conditions are not met, indicating overload, T is dynamically offloaded to a satellite node $\text{SAT}_j \in S_{\text{active}}$ that satisfies in (2).

$$\begin{aligned} \text{CPU}_{SAT_j} &\leq \text{CPU}_{\text{threshold}}, \\ \text{NL}_{SAT_j} &\leq \text{NL}_{\text{threshold}}, \\ \text{LOS}_{SAT_j} &= 1. \end{aligned} \tag{2}$$

Here, (2) represented as $\text{LOS}_{\text{SAT}_j} = 1$ for satellite SAT_j and ground station GS_i at time t. If $\text{LOS}_{\text{SAT}_j} = 0$, no data transfer is possible between them, and tasks must be scheduled or migrated to other nodes that maintain LoS. This ensures that offloading decisions reflect realistic satellite visibility constraints. The load-aware offloading algorithm minimizes total resource usage mentioned in (3).

$$\min_{i \in G \cup S_{\text{active}}} \left(\sum (\text{CPU}_i + \text{NL}_i) \right) \tag{3}$$

Subject to the constraints in (1) and (2), this optimization ensures efficient load balancing and prevents individual nodes from becoming bottlenecks. In practice, satellite nodes have lower processing power, higher per-unit energy consumption, and narrower communication bandwidth compared to ground nodes [3], [10]. These differences impact offloading efficiency and will be incorporated into future cost-aware optimization objectives. Throughout task execution, Kubernetes handles lightweight scheduling, resource monitoring (via Prometheus), and automatic pod cleanup upon completion. Continuous monitoring supports the goal of minimizing end-to-end latency and maintaining high quality of service (QoS) across heterogeneous terrestrial–satellite infrastructures.

IV. System Architecture

Our proposed architecture is based on dual-cluster design integrating terrestrial ground stations and satellite edge nodes via Kubernetes orchestration. The GSCluster and SATCluster operate independently, each with a master node running the API server, scheduler, etcd, and controller manager. Worker nodes at ground stations and satellites execute containerized workloads in pods. Ground nodes, with stable power and network, are prioritized for task scheduling; satellite nodes, with intermittent connectivity, process tasks only when ground resources are constrained. A custom Python controller monitors metrics and enforces offloading/migration policies to sustain performance and service quality.

A. Notation and System Components

Let $G = \{GS_1, \ldots, GS_{12}\}$ denote ground stations and $S = \{SAT_1, \ldots, SAT_{66}\}$ denote satellites. Due to communication and resource limits, only $S_{\text{active}}(t) \subseteq S$ with $|S_{\text{active}}(t)| = 33$ are active at time t, selected by line-of-sight (LoS) and link quality. Tasks $\mathcal{T} = \{T_1, \ldots, T_M\}$ are encapsulated in pods P_k. The mapping $\sigma_k(t) \in G \cup S_{\text{active}}(t)$ identifies the hosting node. For node i, $\text{CPU}_i(t)$ is CPU usage, $\text{NL}_i(t)$ is network load. LoS between SAT_j and GS_i is binary: $\text{LOS}_{j,i}(t) \in \{0, 1\}$. Pods are placed only when resource and LoS constraints (1), (2) are satisfied, ensuring balanced workloads across the heterogeneous network.

B. Initial Pod Scheduling

Upon arrival, task T_k is scheduled to a GS_i if $\text{CPU}_{GS_i}(t_0) \leq \text{CPU}_{\text{threshold}}$ and $\text{NL}_{GS_i}(t_0) \leq \text{NL}_{\text{threshold}}$. If multiple GS nodes qualify, the least loaded is chosen. Execution begins locally until completion or overload.

C. Overload Detection and Offloading

If at t_1 a GS node exceeds thresholds, the controller offloads P_k to an eligible $SAT_j \in S_{\text{active}}(t_1)$ satisfying CPU, network, and LoS constraints. The target satellite minimizes $\text{CPU}_{SAT_j}(t_1) + \text{NL}_{SAT_j}(t_1)$ to ensure efficiency and reachability.

D. Satellite Pod Migration

If LoS is lost ($\text{LOS}_{j,i}(t_2) = 0$), P_k is migrated to another $SAT_\ell \in S_{\text{active}}(t_2)$ meeting CPU, network, and LoS limits. The selected SAT_{ℓ^*} minimizes combined load $\ell^* = \arg\min_{\ell \in S_{\text{active}}(t_2)} [\text{CPU}_{SAT_\ell}(t_2) + \text{NL}_{SAT_\ell}(t_2)]$. This maintains connectivity and balances satellite usage.

E. Overall Optimization Objective

The system's global objective is to minimize the total load across all nodes both ground and active satellite nodes over time. Let $I(t)$ denote the index set of all nodes in $G \cup S_{\text{active}}(t)$. The optimization problem is formulated in (4)

$$\min_{\{\sigma_k(t)\}} \sum_{i \in I(t)} (\text{CPU}_i(t) + \text{NL}_i(t)) \tag{4}$$

subject to CPU/network ($\text{CPU}_{\text{SAT}_\ell}(t_2) \leq \text{CPU}_{\text{threshold}}$) and ($\text{NL}_{\text{SAT}_\ell}(t_2) \leq \text{NL}_{\text{threshold}}$) thresholds and node activity constraints:

$$\text{CPU}_i(t) \leq \text{CPU}_{\text{threshold}}, \quad \text{NL}_i(t) \leq \text{NL}_{\text{threshold}}, \quad \forall i \in I(t), \forall t.$$

Pods on ground nodes continue running until either (a) they finish and expire, or (b) the node becomes overloaded, triggering offloading. On satellites, pods run until completion, TTL expiry, or LoS loss, triggering migration. This strategy ensures efficient resource distribution and prevents any single node from becoming a bottleneck.

V. Results and Evaluation

To validate the performance and scalability of our proposed Kubernetes-based dual-cluster framework, we conducted extensive simulations replicating real-time task execution in an integrated satellite-ground edge environment.

A. Experimental Setup and Monitoring

The system integrates tightly with Kubernetes for managing pod lifecycles and monitoring resource metrics. Pods are initially created on a ground station node GS_i if its CPU and network load are below predefined thresholds. Realtime metrics, including CPU usage, network load, and line of sight (LOS), are collected using Prometheus, which scrapes data from node exporters and kube state metrics. A custom Python controller evaluates these metrics and triggers offloading or migration decisions based on the conditions defined in the system equations. Pods are automatically deleted upon task completion via TTL expiration. The Kubernetes master node comprising the controller manager, API server, etcd, and scheduler handles pod creation requests submitted via custom controller, and schedules them on appropriate worker nodes that meet the resource constraints. Ground worker nodes are implemented using KIND (Kubernetes-in-Docker), while satellite nodes run k3s agents. These nodes host Docker containers (pods) that process tasks and continuously report metrics to Prometheus. Prometheus scrapes the system wide metrics, and Grafana dashboards visualize CPU and network utilization per node. Alerts are triggered when resource usage approaches the defined thresholds, enabling proactive load balancing and reliability assurance.

B. Experimental Results and Analysis

We evaluated the proposed framework under varying workloads to measure CPU and network utilization, offloading frequency, load balancing, pod distribution, and cluster behavior. As shown in Fig. 2, average latency decreased significantly after enabling offloading: GSCluster latency dropped from about 120 ms to 80 ms, and SATCluster latency from roughly 200 ms to 130 ms. These results confirm the framework's ability to dynamically offload tasks and balance workloads using real-time CPU, network, and line-of-sight metrics. Fig. 3 shows that before offloading, GSCluster was overloaded (80%) while SATCluster was underutilized (35%). After offloading, GSCluster usage dropped to 45% and SATCluster rose to 65%, indicating effective load balancing. Fig. 4 reveals an optimal offloading range of 40–60%, achieving peak task success (90%), reduced latency (120 ms), and balanced CPU usage. Lower offloading causes ground congestion, while higher offloading reduces success rates due to satellite limits.

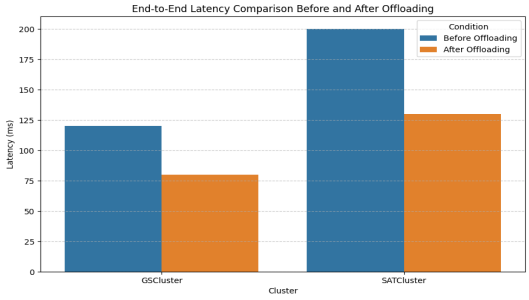

Fig. 2. End-to-End Latency Performance before and after Offloading

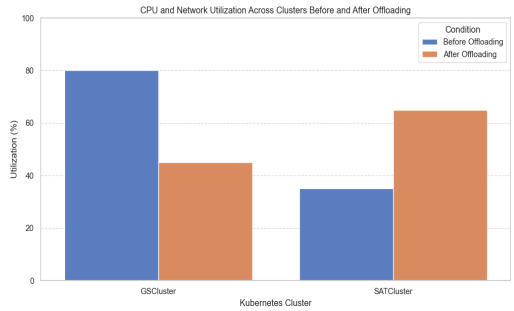

Fig. 3. CPU and Network Utilization Across Clusters Before and After Offloading

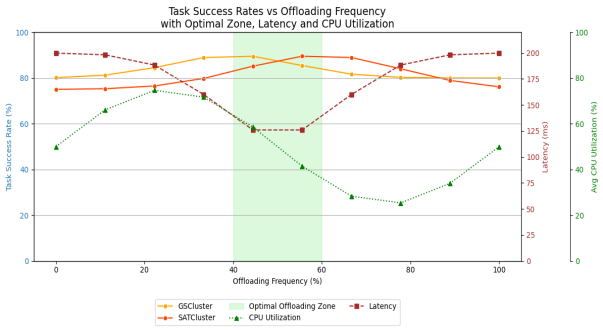

Fig. 4. Task Success Rate over Offloading Frequency

Fig. 5 compares no offloading with offloading without load balancing. Offloading reduces latency from 320 ms to 220 ms, CPU usage from 88% to 75%, and task failures from 18% to 10%, but introduces 13 task migrations. This indicates

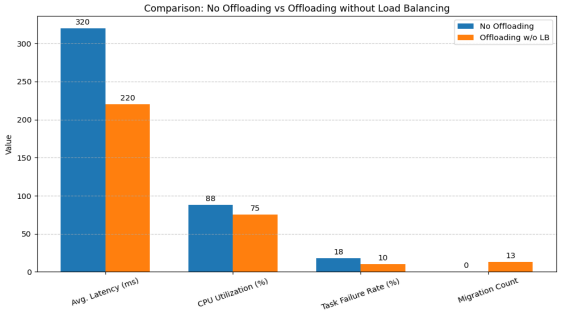

Fig. 5. No Offloading vs Offloading without load balancing

that while offloading eases congestion, load balancing is still needed for optimal placement and stability. Fig. 6 shows offloading events over a 1-hour simulation, with peaks when ground nodes near resource limits. These fluctuations indicate the controller actively balances workloads, preventing overload and maintaining consistent performance across clusters.

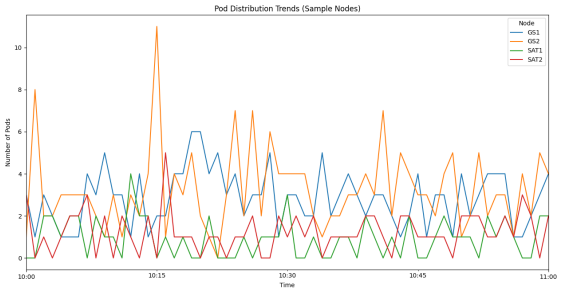

Fig. 6. Offloading Events over Time

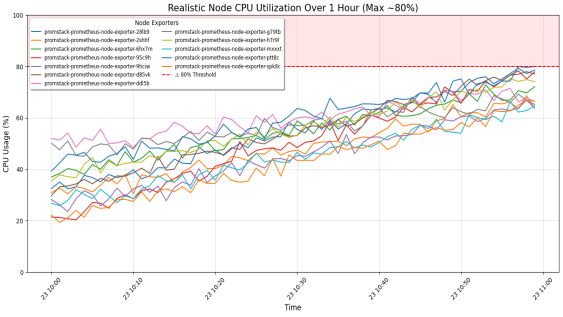

Fig. 7. Realistic Node CPU Utilization Over 1 Hour (Max 80%)

Fig. 7 shows CPU usage over one hour, capped at 80%. Most nodes operate efficiently between 40% and 75%, with few nearing the limit, indicating safe operation and overload prevention. Variations across nodes confirm intelligent load distribution, ensuring stability and responsiveness under fluctuating demand. Overall, the graph validates that the proposed framework effectively prevents CPU saturation, dynamically balances tasks, and sustains stable cluster performance under fluctuating demand.

VI. Conclusion

In this paper, we presented a pod-based dual-cluster orchestration framework that combines Kubernetes with real-time monitoring to enable intelligent task offloading between ground and satellite nodes. Our approach addresses the challenges of dynamic resource availability and intermittent connectivity by leveraging a custom controller and threshold-driven policies for efficient task scheduling and migration. Simulation results validate the system's ability to reduce latency, optimize CPU utilization, and improve task success rates through balanced workload distribution. The findings highlight the framework's potential to enhance the scalability, responsiveness, and reliability of next-generation space-terrestrial edge computing systems.

Acknowledgment

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT)(No.RS-2024-00398379, Development of High Available and High Performance 6G Cross Cloud Infrastructure Technology).

References

[1] Zhenyu Shan, Linhui Wei, Junjie Li, Yumei Wang, and Konglin Zhu. Edge computing-enabled real-time resource allocation system based on satellite cluster. In *2024 10th International Conference on Computer and Communications (ICCC)*, pages 2128–2132, 2024.

[2] Alessio Giorgetti, Davide Scano, Javad Chamanara, Mustafa Albado, Edgard Marx, Sean Ahearne, Andrea Sgambelluri, Francesco Paolucci, and Filippo Cugini. Kubernetes orchestration in sdn-based edge network infrastructure. pages 1–3, 2022.

[3] Yifei Hu and Wenbin Gong. An on-orbit task-offloading strategy based on satellite edge computing. *Sensors*, 23(9), 2023.

[4] Syed Saqib Jamal and Wang-Cheol Song. Gnn-based routing for link reliability optimization in sd-leo satellite networks. In *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–4, 2024.

[5] Estela Carmona-Cejudo, Francesco Iadanza, and Muhammad Shuaib Siddiqui. Optimal offloading of kubernetes pods in three-tier networks. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 280–285, 2022.

[6] Quang-Minh Nguyen, Linh-An Phan, and Taehong Kim. Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy. *Sensors*, 22(8), 2022.

[7] Naveen Kodakandla. Optimizing kubernetes for edge computing: Challenges and innovative solutions. *IRE Journals*, 4(10):210–221, 2021.

[8] Leonardo Poggiani, Carlo Puliafito, Antonio Virdis, and Enzo Mingozzi. Live migration of multi-container kubernetes pods in multi-cluster serverless edge systems. page 9–16, New York, NY, USA, 2024. Association for Computing Machinery.

[9] Alireza Bakhshi Zadi Mahmoodi and Ella Peltonen. Kubernetes task-offloading framework for edge-cloud continuum in vehicular computing. IoT '24, page 146–149, New York, NY, USA, 2025. Association for Computing Machinery.

[10] Tongguang Zhang, Chunhong Liu, Qiaomei Tian, and Bo Cheng. Cloud-edge collaboration-based multi-cluster system for space-ground integrated network. *International Journal of Satellite Communications and Networking*, 43(1):40–60, 2025.

[11] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta ur Rehman Khan. A survey of kubernetes scheduling algorithms. *J. Cloud Comput.*, 12(1), June 2023.

[12] Yanzhe Huang, Xing Zhang, and Zechang Xu. Satedge: Platform of edge cloud at satellite and scheduling mechanism for microservice modules. *IEEE Access*, 11:126283–126294, 2023.

[13] Xin Wang, Kai Zhao, and Bin Qin. Optimization of task-scheduling strategy in edge kubernetes clusters based on deep reinforcement learning. *Mathematics*, 11(20), 2023.

[14] AWS HPC Blog Team. Gang scheduling pods on amazon eks using aws batch multi-node processing jobs, 2024. Accessed: 2025-06-03.

[15] Danyang Liu, Yuanqing Xia, Chenggang Shan, Guan Wang, and Yongkang Wang. Scheduling containerized workflow in multi-cluster kubernetes. In *Big Data*, pages 149–163, Singapore, 2023. Springer Nature Singapore.

[16] Kubernetes Authors. Pod lifecycle, 2025. Accessed: 2025-06-03.

[17] Abd Elghani Meliani, Mohamed Mekki, and Adlen Ksentini. Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments. *Computer Communications*, 236:108111, 2025.

[18] Zheqi Zhang, Yaling Xun, Haifeng Yang, and Jianghui Cai. Application type awareness pod-level and system-level container scheduling. *Future Generation Computer Systems*, 173:107898, 2025.

[19] Peini Liu and Jordi Guitart. Fine-grained scheduling for containerized hpc workloads in kubernetes clusters. pages 275–284, 2022.