Latency-aware Scheduling in the Cloud-Edge Continuum

Latency-aware Scheduling in the Cloud-Edge Continuum

Cristopher Chiaro * Doriana Monaco * Alessio Sacco * Claudio Casetti * Guido Marchetto *
* Department of Control and Computer Engineering, Politecnico di Torino, Italy

***Abstract*—In recent years, containerized deployment models have gained favor across many domains of applications. Kubernetes, the de-facto standard for containers orchestration, can efficiently manage heterogeneous devices, but fails to adapt to possibly stringent requirements, as it only considers computing metrics for scheduling decisions. In addition, the rising prominence of distributed cloud environments, which enable the development of highly available, performant solutions, requires modifications to the default Kubernetes scheduler. To address these challenges, we introduce LAIS, a multi-cluster Kubernetes scheduler optimized for end-to-end latency measurements to enhance user Quality of Experience (QoE). Unlike existing approaches, we define a geographically distributed environment and deploy a solution that satisfies user-specified intents in terms of latency. Depending on user needs, LAIS can either meet a specific latency constraint or schedule pods in the cluster with the lowest latency. After implementing LAIS in a multi-cluster environment, we found it highly effective in accommodating a range of user intents, outperforming the default Kubernetes scheduler in this regard.**

I. Introduction

Container virtual technology is increasingly adopted for the development of microservice-based applications. In such a context, orchestrators simplify the management of container applications by overseeing the entire life cycle of containers. However, they are usually limited by infrastructure-level management policies. For example, Kubernetes, the de-facto standard orchestrator, enables the automation of application deployment, scaling, and operations, including the management of pods, the smallest deployable units in a Kubernetes cluster. Nonetheless, the optimization strategies adopted are limited to computing capacity. However, the rapid advancement of technology is pushing the boundaries of digital transformation and creating an enormous dependency on network-based services. In this evolving landscape, network latency plays a crucial role, influencing the user experience and ultimately impacting the overall success of services.

In addition, as the edge and fog computing paradigms gain prominence to address issues related to geographic proximity, reduced latency, and enhanced privacy [1]–[3], these strategies are gradually expanding their reach to encompass smaller data centers situated at the network periphery. This expansion leverages consistent foundational elements to promote service flexibility and an edge-to-cloud continuum [4], [5], with the increasing popularity of multi-regional cloud resources.

For this reason, novel solutions for distributed environments are being developed. Many orchestrators prioritize resource optimization, often incorporating auto-scaling features as seen in works like [6]–[8]. These studies propose architectures that establish a federated Kubernetes domain, making use of Network Service Mesh (NSM) tools for enhanced functionality.

An essential element in the management of microservices within Kubernetes is the scheduler, handling the intricate task of arranging and executing the distribution of pods across nodes. Originally designed for conventional cloud settings that depend on centralized, high-capacity data centers, the typical Kubernetes scheduling techniques are primarily focused on optimizing computational resources like CPU and memory [9]. Nevertheless, within the edge-cloud continuum, scheduling decisions can have a significant impact on the performance achieved by applications managed via Kubernetes. Despite the advanced scheduling mechanisms provided by Kubernetes, such as load balancing or the round-robin algorithm [10], a critical observation is the absence of innate capabilities to dynamically adapt to variations in network metrics and the absence of real-time metrics in decisions.

Several approaches to overcome such limitations exist. Recent solutions aim at minimizing the end-to-end delay of applications clustering together dependent microservices, either estimating the latency via the amount of traffic to be exchanged [11], [12] or collecting intra-node latencies [13], [14]. Other approaches focus on reducing the deployment time [15], [16] or schedule pods close to a specified target location [17]. However, only a few of these schedulers make informed decisions based on user-perceived latency. [18] designs a scheduler's architecture to encompass initial scheduling based on predetermined metrics and enable dynamic re-scheduling in response to real-time latency data. In contrast, we design a multi-cluster scheduler that not only meets user-specified intents but also focuses on user-perceived latency, effectively leveraging the cloud-edge continuum.

In particular, we propose **L**atency-**A**ware **I**ntent **S**cheduler (LAIS), a custom Kubernetes scheduler that dynamically (de)allocates pods based on real-time latency measurements to satisfy users needs. LAIS considers a multitude of distributed peering clusters and schedules pods in the closest cluster to reduce the user-perceived latency. The system is modeled as a closed loop that continuously monitor the latency, in order to update scheduling decisions and meet user mobility needs.

We deployed LAIS in a real-world cluster system hosted at Politecnico di Torino, and compared our solution to the default Kubernetes Scheduler to demonstrate its effectiveness in reducing the latency. In addition, we compared its intent-based behaviors and studied their impact on performance and convergence time.

II. Scheduling Solution

Virtualized infrastructures are increasingly distributed across multiple geographic locations. Traditional single-cluster solutions cannot be adopted in such environments because of the inherent restriction in cluster size. Moreover, managing resources spanning numerous sites introduces complexities in scheduling, resource allocation, and other critical aspects emphasizing the need for a more scalable, robust approach. Multi-cluster solutions are necessary to ensure high availability and application isolation, while dealing with geographic regulations, especially considering the nature of edge devices. However, multi-cluster scheduling comes with challenges: distributed resources increase fragmentation and balancing the workload is complex for heterogeneous dynamic clusters.

Our solution enhances scheduling decisions while managing a multi-cluster environment, as shown in Fig. 1. As in other multi-cluster architectures [19]–[21], one cluster has the role of master: it is equipped with our Latency-Aware Scheduler and it is connected to other (slave) remote clusters as geo-distributed Kubernetes nodes. Each cluster consists of a control plane node and multiple worker machines. The control plane is in charge of orchestrating the cluster and performs scheduling actions for the allocation of pods, while worker nodes run the application containers. In this context, remote pods can communicate with each other as if they are all executed in the same cluster, either with or without NAT translation. However, while for pods the remote or local communication is transparent, this is not the case for the applications on top, where the proximity of pods to the user play a significant role. Whenever a user sends a request (Phase 1), it communicates with LAIS API, which internally queries the Latency-Aware Scheduler. Thanks to Latency Meter containers, real-time measurements are collected and used to identify the appropriate cluster that meets the latency requirements. To balance the workload across nodes, a resource-aware Scheduler is deployed inside each cluster. When the pods are eventually allocated (Phase 2), a peer communication between the user and the pods is established (Phase 3).

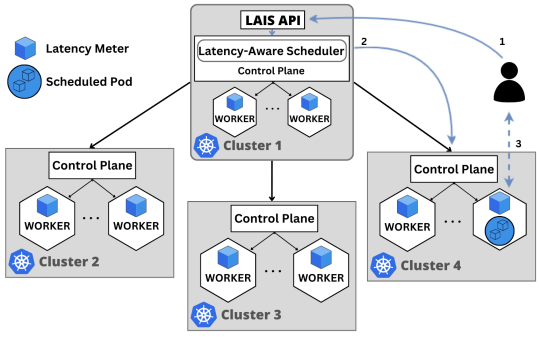


Fig. 1: Multi-cluster scheduler. LAIS leverages edge computing to deploy pods close to the user.

A. How to Consider the Latency in LAIS Orchestration

To perform latency-based actions, we leverage two bespoke Golang applications: the Latency Meter and a Custom Latency Aware Scheduler. The *Latency Meter* is deployed across all worker nodes as a *sentinel* container in each pod replica and acts as a proxy: it intercepts each user request to measure and store real-time network latency between users and nodes. Upon receiving a request, it compares server and client timestamps to compute the latency, then stores it in a volatile memory *LatencyMeasurements (LM)*. These measurements can be periodically fetched by the custom Latency Aware Scheduler.

The *Latency-Aware Scheduler* operates in the control plane, replacing the default scheduler and leveraging latency metrics for dynamic, informed pod scheduling. This custom scheduler comprises a scheduler and a descheduler. The *Scheduler* extends the default scheduler behavior with a mechanism that prioritizes unexplored nodes to collect new latency measurements. The *Descheduler* reads the LM and decides whether a pod needs to be reallocated.

Whenever a user request is received, an exploration phase to collect measurements begins. At first, no measurements are available and the requested replicas are scheduled at random. If the current configuration violates the latency constraints, the Descheduler takes action by deallocating the involved pods. These pods are then reallocated to previously unexplored nodes in order to gather new latency measurements. A steady state is achieved once the latency constraints are met. However, it is worth noting that the perceived latency can fluctuate due to dynamic network conditions or user mobility. As a result, the entire process operates within a control loop that continuously monitors latency levels, leading to adaptive updates in scheduling decisions as needed.

B. LAIS Node Selection

The scheduling process outlines a framework for allocating pods to worker nodes. Typically, it employs a request-based approach that initially filters out nodes lacking sufficient resources, then allocates the pod to one of the remaining available nodes. This makes the scheduler a critical component responsible for resource management and workload adaptation decisions. Additionally, it plays a vital role for latency-sensitive microservices executing within the cloud-edge *continuum*.

LAIS performs scheduling decisions based on user's latency needs, expressed as intents. Indeed, for each Kubernetes deployment the user can specify hard, soft, or no constraint through a YAML file. Specifically, within the *spec:template:metadata:annotations* section of the YAML, users can specify up to two values: `hard_constraint` and `soft_constraint`. Since we assume a reduced latency variation inside a cluster, LAIS first performs latency-aware decisions to select a suitable cluster, and later applies load-balancing decisions to choose the node.

Based on user-defined constraints, the LAIS Scheduler classifies the N_tot clusters into three distinct types, based on the suitability of each cluster for hosting the pods:

- Soft Valid: *latency* $<$ *soft_constraint*
- Hard Valid: *soft_constraint* $<$ *latency* $<$ *hard_constraint*
- Invalid: *latency* $>$ *hard_constraint*

This approach allows users to fine-tune LAIS and trigger different behaviors depending on specific needs.

Setting a hard constraint implies enforcing a strict latency threshold. In this case, if the strict threshold is not respected, the Descheduler immediately releases the involved replicas and requests a reassignment, until the constraint is satisfied. Here, the system prioritizes a fast and efficient mechanism over frequent pod migrations. The focus is on a trade-off between minimizing latency and maintaining operational stability. If the constraint cannot be satisfied, the scheduler will select the lowest latency cluster and notifies the user.

If a soft constraint is specified, Soft Valid clusters are prioritized over Hard Valid ones. In addition, if the hard constraint is already satisfied and a sufficient number of Soft and Hard Valid clusters is used, then the Soft Condition mechanism can be activated. The triggering condition of this mechanism is determined by:

$$\text{N\_soft} + \text{N\_hard} > \frac{\text{N\_tot}}{2}, \tag{1}$$

where N_soft the number of Soft Valid clusters that currently host the replicas (resp. N_hard, Hard Valid). The Soft Condition mechanism consists of descheduling replicas from Hard Valid clusters in favor of Soft Valid ones to further reduce the latency. However, because of the higher number of measurements and reallocations needed, and the prioritization of Soft clusters over the other ones, satisfying the soft constraint comes at the expense of convergence time and load balancing.

If no constraint is defined, our scheduler will minimize the user-perceived latency. To do so, it collects measurements from all the clusters. Therefore its primary strategy is to allocate pod replicas on a variety of nodes, while the Descheduler continuously monitors the latency. Once the Descheduler stores measurements from all the application pod replicas, a loop mechanism begins. As a result, pods in the worst cluster, the one with highest latency, are removed. This triggers the allocation of new replicas in nodes that are not yet visited. New measurements are then collected and the worst cluster is again identified. Once an adequate number of measurements is collected, the entire set of stored values is considered to schedule replicas on the global lowest latency clusters.

Since a cluster is composed of nodes that are geographically close to each others, especially in edge computing environments, we assume that the latency variability inside a cluster is minimal, thus we deploy a resource-aware scheduler for node selection. To distribute pods across a variety of nodes, LAIS creates a priority list. A node's priority is based on the number of pod replicas of the same application it hosts: the fewer the replicas, the higher the priority. If nodes have the same priority, the one with less resource usage (like CPU and RAM) is chosen. In case of a tie, the selection becomes random. It is important to note that any network-aware solution for intra-cluster scheduling can be integrated with LAIS.

III. Evaluation

We test our LAIS over CrownLabs [22], a specialized cloud environment provided by Politecnico di Torino. We create 9 Kubernetes-driven clusters and 18 nodes in Kubernetes 1.28, each worker node runs Ubuntu 20.04.6 LTS with 2 GB RAM, 2 CORE CPU and 20 GB disk, `containerd` is the default container-runtime, and the Container Network Interface (CNI) is `Flannel`. Clusters interconnection is enabled by Liqo [19] through Out-of-band peering, which means that the Liqo control plane traffic, including initial authentication and communication with remote Kubernetes API servers, flows outside the VPN tunnel established between clusters. This approach supports dynamic synchronization between clusters and allows them to interact independently under different administrative domains, while ensuring secure communications via TLS.

To collect real-time metrics we leverage an Nginx application incorporating a Latency-Meter container, while we use the `tc` command to set the latency range for each cluster. Note that we empirically assessed an inherent baseline latency within clusters between 15 and 25 ms.

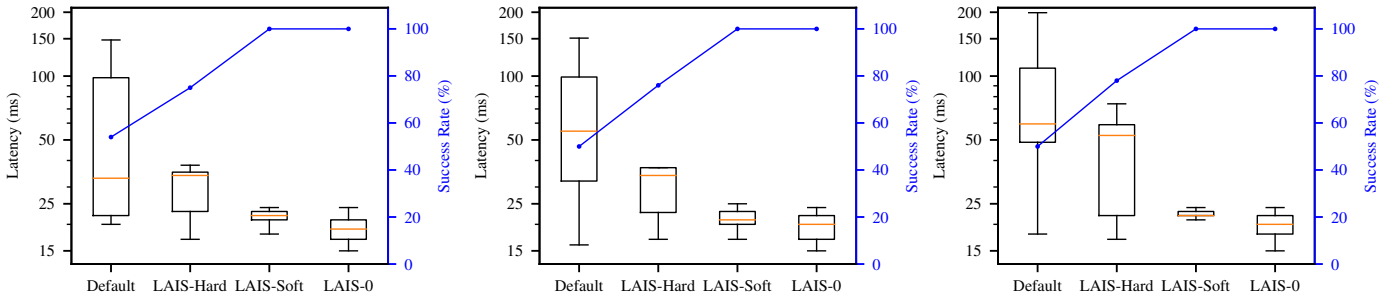
We design three scenarios to study the success rate and latency distribution for diverse network conditions: *Scenario 1* enforces strict latency constraints, which is challenging for a large number of clusters to meet, exemplifying a high-demand, low-tolerant application setup. *Scenario 2* relaxes the latency limits and consists of low latency clusters, reflecting a more accommodating network environment. In contrast, *Scenario 3* is characterized by clusters with higher latency but acceptable latency bounds. This diversity in scenarios, reached by varying the latency via `tc`, allows us to assess the capability of schedulers in deploying pods in the most suitable clusters.

We set the latency meter to collect values every 5 seconds, and the descheduler to operate every 30 seconds, to avoid frequent migrations and consider a reliable evolution of latencies. In each condition, we performed 100 tests and reported the average values.

A. Convergence and Accuracy of LAIS

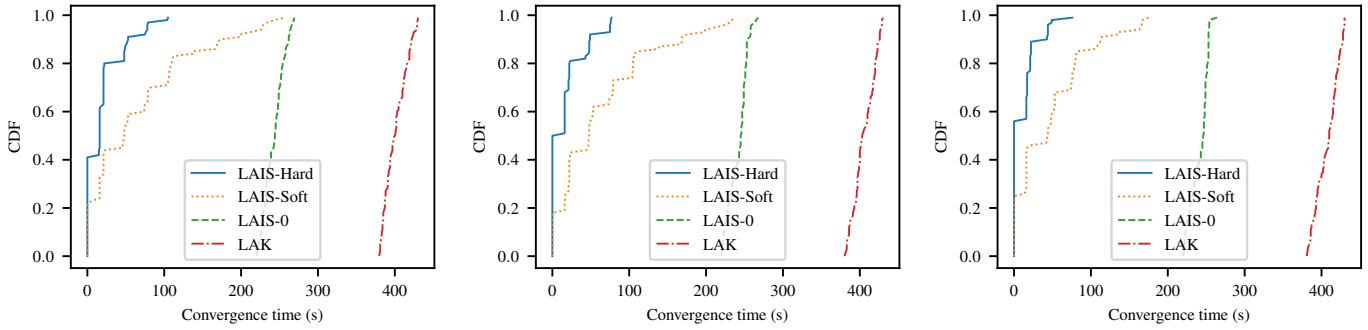
In this set of experiments we compare the Default Kubernetes Scheduler and three variants of LAIS: *LAIS-Hard*, where our latency-aware scheduler only aims at satisfying the hard constraint, *LAIS-Soft*, where the scheduler also attempts to meet the soft constraint, and *LAIS-0*, where the scheduler just selects the node that minimizes the user latency.

We start by comparing the ability of schedulers to meet the required latency. Fig. 2 shows the distribution of latency values (box plot) and the success rate for soft constraints (blue curve) for all the tested schedulers in multiple scenarios. We can observe how the Default Scheduler exhibits mediocre performance across the scenarios with an average latency of 70.52 ms, the highest among all schedulers. In addition, the latency fluctuates considerably among runs since pods are scheduled in a latency-agnostic way. Concerning its success rate, it hovers around 50%. This indicates it has merely a 50% likelihood of placing a pod in a valid cluster, basically performing tantamount to a random decision.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 2: Latency (box plot) and success rate (blue curve) for all testing scenarios.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 3: CDF of converge time for the three scenarios. Compared to a different latency-aware scheduler a LAK, LAIS reduces the convergence time.

LAIS-Hard showcases a significant reduction in latency as opposed to the Default Scheduler across all use cases. It has a success rate of 76.5%. It does not reach 100% since it trades off "hard" and "soft" constraints. LAIS-Soft and LAIS-0 consistently maintained remarkably low latency, below 22 ms, with LAIS-0 slightly outperforming LAIS-Soft in most scenarios. Conversely, both LAIS-Soft and LAIS-0 hit an impressive 100% success rate, denoting they are extremely reliable in assuring task execution.

Clearly, LAIS-0 demands more time, so we define the *convergence time* as the period required to satisfy the user intent (meanwhile services are consistently available). For this reason, we also evaluate and report the cumulative distribution function (CDF) of convergence time of all LAIS schedulers in Fig. 3. We compare against a similar latency-aware scheduler, namely [18] referred to as Latency-Aware Kubernetes (LAK) in the following. The variance in convergence time among the LAIS schedulers belies different behaviors. LAIS-Hard regularly recorded the quickest convergence times, proving its swift adaptability to system fluctuations. LAIS-Soft took instead longer to adapt but was considerably quicker than LAIS-0, which displayed prolonged convergence times, often exceeding 241 seconds. LAK, instead, by not considering the union of clusters but treating them individually, takes much longer to find suitable nodes to host the applications in all three scenarios. This result validate our proposed fluid architecture.

In summary, the Default Scheduler does not implement any latency mechanism, instead, it randomly choose nodes in a cluster, resulting in either correct or incorrect selections. In contrast, the LAIS schedulers demonstrate a clear advantage. On the one hand, LAIS-Hard offers a balance between swiftness and performance, making it ideal for dynamic environments. On the other hand, LAIS-Soft and LAIS-0 are geared towards highly-performing solutions. Hence, based on the requirements - adaptability or swift action - an appropriate scheduler can be chosen.

IV. Conclusion

In this work, we tackle the limited adaptability of the default Kubernetes scheduler by introducing LAIS, a latency-aware scheduler that uses real-time measurements to fulfill user-defined intents. Our scheduler offers unified management for geographically distributed clusters and leverages edge computing capabilities to allocate pods in clusters with reduced latency. Within the cluster, we adopt a load-balancing strategy to evenly distribute pods across nodes.

Depending on the user's specified intent, LAIS can deliver a balanced mix of responsiveness and performance, or zero in on providing the lowest possible latency. Continuous monitoring accommodates user mobility, allowing for timely updates to scheduling decisions as needed.

Acknowledgment

This work has received funding from EU Horizon Europe R&I Programme under Grant Agreement no. 101070473 (FLUIDOS).

REFERENCES

[1] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] A. Sacco, F. Esposito, P. Okorie, and G. Marchetto, "LiveMicro: An Edge Computing System for Collaborative Telepathology," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge '19)*. USENIX, 2019.

[4] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–21, 2019.

[5] A. Sacco, F. Esposito, and G. Marchetto, "Resource Inference for Sustainable and Responsive Task Offloading in Challenged Edge Networks," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 3, pp. 1114–1127, 2021.

[6] N. T. Nguyen and Y. Kim, "A Design of Resource Allocation Structure for Multi-Tenant Services in Kubernetes Cluster," in *27th Asia Pacific Conference on Communications (APCC '22')*. IEEE, 2022, pp. 651–654.

[7] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "mck8s: An Orchestration Platform For Geo-distributed Multi-cluster Environments," in *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–10.

[8] L. Osmani, T. Kauppinen, M. Komu, and S. Tarkoma, "Multi-cloud connectivity for kubernetes in 5g networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 42–47, 2021.

[9] O. Tomarchio, D. Calcaterra, and G. D. Modica, "Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks," *Journal of Cloud Computing*, vol. 9, pp. 1–24, 2020.

[10] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[11] Ł. 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 *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–9.

[12] A. Marchese and O. Tomarchio, "Network-aware container placement in cloud-edge kubernetes clusters," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 859–865.

[13] J. Santos, C. Wang, T. Wauters, and F. De Turck, "Diktyo: Network-aware scheduling in container-based clouds," *IEEE Transactions on Network and Service Management*, 2023.

[14] K. Govindarajan, C. Govindarajan, and M. Verma, "Network aware container orchestration for telco workloads," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 397–406.

[15] W.-K. Lai, Y.-C. Wang, and S.-C. Wei, "Delay-aware container scheduling in kubernetes," *IEEE Internet of Things Journal*, 2023.

[16] F. Rossi, V. Cardellini, F. L. Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Computer Communications*, vol. 159, pp. 161–174, 2020.

[17] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[18] C. Centofanti, W. Tiberti, A. Marotta, F. Graziosi, and D. Cassioli, "Latency-aware kubernetes scheduling for microservices orchestration at the edge," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023, pp. 426–431.

[19] L. Team, "Liqo: Seamless multi-cloud and cluster federation," 2023, accessed: 2023-05-27. [Online]. Available: https://liqo.io/

[20] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend Cloud to Edge with KubeEdge," in *2018 IEEE/ACM Symposium On Edge Computing (SEC)*. IEEE, 2018, pp. 373–377.

[21] K. Team, "Karmada: Open, multi-cloud, multi-cluster kubernetes orchestration," 2023, accessed: 2023-10-20. [Online]. Available: https://karmada.io/

[22] "CrownLabs," https://crownlabs.polito.it, Accessed: 2023-09-28.