Stateless Job Offloading for Mobile Robots in Kubernetes

Stateless Job Offloading for Mobile Robots in Kubernetes

Daniele Cacciabue
daniele.cacciabue@polito.it
Politecnico di Torino
Torino, Italy

Francesco Aglieco
francesco.aglieco@linksfoundation.com
Links Foundation
Torino, Italy

Domenico Perroni
domenico.perroni@italdesign.it
Italdesign
Torino, Italy

Fulvio Risso
fulvio.risso@polito.it
Politecnico di Torino
Torino, Italy

ABSTRACT

Edge devices are increasingly requiring more and more intelligence, hence asking for an amount of computing power that is not always sustainable on the individual device itself. In particular, this applies to IoT devices where battery consumption and low processing power are constraints that limit the amount and complexity of tasks that can be performed on the edge, hence demanding for a tighter *edge-to-anything* interaction for job offloading. This paper focuses on mobile robots running the ROS operating system, presenting an offloading approach for long-lived stateless services based on a switching approach, an algorithm that optimizes for a faster switching between local and remote execution, minimizing downtime of the service.

CCS CONCEPTS

• **Computer systems organization** → **Robotics**; **Reliability**; **Availability**; **Redundancy**; • **Networks** → *Cloud computing*.

KEYWORDS

ROS2, Task Offloading, Cloud Offloading, Zenoh

ACM Reference Format:
Daniele Cacciabue, Francesco Aglieco, Domenico Perroni, and Fulvio Risso. 2024. Stateless Job Offloading for Mobile Robots in Kubernetes. In *Proceedings of 1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (MECC 2024).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Task offloading is getting more and more relevant. In today's computing paradigm, more and more devices are getting connected to the internet and require access to services exposed by remote servers. Especially in IoT devices where battery consumption and low-powered processors are constraints that limit the amount and complexity of tasks that can be performed on the edge. This paper presents an offloading approach based on a switching approach, an algorithm that optimizes for a faster switching between local and remote execution, minimizing downtime of the service.

The main value of this paper stands in documenting the algorithm used to minimize the time between stopping the local instance of the task and its activation on the remote server. Moreover, this paper also introduces two possible software architectures that can be used to connect the robot to the cloud.

Among the many possible offloading use cases to work on, this study will focus on the simplest of the scheduling cases, in which there are only two actors, one is a robot that offloads a task and the other is a server that can execute the task. On the other hand, such scenario represents also the most difficult of the task migration cases, a low-latency live migration. In this scenario, the task can either be executed locally on the robot or remotely on the server and should seamlessly switch to the system that a scheduling algorithm has deemed as optimal.

2 SERVICE MIGRATION CHALLENGES

The following challenges are considered to be out-of scope, and will be left to future works.

- **Peer discovery**: A robot needs to be able to discover what servers or robots and other devices providing a task offloading service that are around it. This study will start from the assumption that the remote cluster has a fixed IP address.
- **Locating the service**: One service can be run on multiple machines, the selection of the correct machine where to execute it can be done using either heuristics or Machine-learning-based strategies. This study will analyze a very simple configuration consisting of a only a remote and a local cluster.
- **Scheduling of tasks**: When deciding where to migrate the execution of a task, one must also decide the priority to be given to every migrated task. For example, if on the local cluster we have task A and B running and at a certain point during execution, the location algorithm decides that both tasks need to be migrated to a remote cluster, we need to specify if the first task to be switched will be task A or task B. It is a matter of deciding which of the running tasks is the most critical for the system. This paper will be focusing on the switching of a single task and not on multiple ones.
- **State synchronization**: When synchronizing the state of execution, as soon as more than two machines are considered,

MECC 2024, April 22nd, 2024, Athens

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

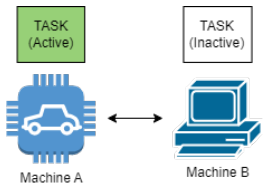

Figure 1: Task is running on both machine A and machine B, but the active instance is by default the one on A.

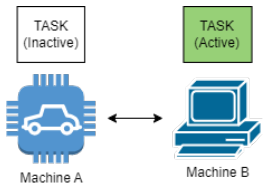

Figure 2: The task on machine A can be deactivated and the one on machine b is activated instead.

one must start worrying about data reliability and consensus which can exponentially increase the complexity of the problem. For this reason, this study will involve the case in which there are only two machines the tasks can be run on. In this configuration, every data has a single clearly-defined source of truth which avoids problems of consensus.

Challenges in scope. The goal of this paper is to study a possible way to switch the execution of a task from one cluster to another one, minimizing the downtime. This is different from a service migration because:

- **Service migration**: the task T starts running on machine A. At one point during execution, machine B is elected as the most suitable one for the task. The task code and current data is moved to the machine B. Execution is stopped on machine A and started on machine B.
- **Switching**: the task T has two running instances. T_a runs on machine A and T_b runs on machine B. The state of T_a and T_b is kept synchronized so that one instance of the task can work as a backup of the other. In this case, T_a is the one that is currently active and T_b is used as a backup service (figure 1). At one point during execution, machine B is selected as a better candidate to run the task, so T_b becomes the active task and T_a becomes the backup task (figure 2).

Switching and migration differences. One could view switching as a subset of the migration problem. In fact, low downtime migration algorithms already use a switching step. They work by preemptively moving the service code and data to the remote machine and proceed to switch the execution by shutting down the old instance and starting the new one only when the state of the two instances is synchronized [7]. Switching may at first appear just as a less efficient version of migration because it requires running two instances (or more) of the task, doubling the required resource usage, this paper has been written in order to prove that such resource consumption is justified for specific, critical use cases that require a service to have very low downtime backup in order to guarantee business continuity.

The main difficulties when it comes to deciding how to implement the switching are mainly related to the switching delay, as a good switching algorithm requires that the time spent to switch execution from a machine to another one to be minimized, reducing the impact of the switching operation on the overall system.

3 STATE OF THE ART

The problem space involved in performing the kind of offloading explained in this paper requires three main components:

- **Switching**: the procedure of disconnecting the output of a task and turning on the output of another one, mantaining the switched-off task ready to resume as a fallback mechanism.
- **State synchronization**: the process by which two tasks running simultaneously on two different systems are able to maintain a common knowledge of the data they are working on.
- **Locating the best peer**: consists of selecting when and where to switch computation to, depending on given parameters to be determined by each use case.

Previous works on switching. Many studies [4][10][8] have analyzed the problem of optimal location and scheduling of tasks among available peers, generally involving the usage of either heuristic approaches or deep neural networks to maximize the throughput (the number of tasks executed in a given time). The majority of task offloading studies focuses on the scheduling problem and not on the migration problem usually with the purpose of minimizing the resource usage of a possible migration [9] or minimizing the delay that it introduces[2].

In the context of live-migration there are techniques to minimize the time required for migrating a workload, usually consisting of preemptively moving the container image on the other server [7]. The solution proposed by this paper differentiates itself from these kind of studies because it does not focus on migrating the service, but on switching its execution from a local cluster to a remote one, keeping both the copies running, one actively, the other passively, as a fallback. This kind of approach does not involve any downtime related to stopping and restarting the execution.

As of time of writing, no paper was found focusing on the switching problem. Previous similar works have mainly been performed in the context of live-migration, which, as previously stated, can be viewed as a superset of switching.

Low-downtime switching has been achieved in previous work after synchronizing local and remote state using a distributed key-value store [3] or using container check pointing and file system layering [7]. Since this paper will not tackle the state synchronization challenge and will instead focus more on the business continuity aspect, the previously stated papers have not been used as a basis for the work but rather as a reference.

4 BACKGROUND

The scenario under study will involve switching the execution of a task from a machine A to a machine B analyzing its performance

and behaviour in case of a switch determined by a dummy location algorithm.

In our experiment instead of using two machines we will use two Kubernetes clusters, one representing the robot and one representing the cloud. In this first iteration, state synchronization between the two clusters will not be taken into consideration as only stateless services will be used.

In order to implement such scenarios, many technologies can be used, this section contains the technologies that have been used and the reason why they have been chosen.

ROS2. ROS2[6] has been chosen because it is de-facto standard as far as programming robots is concerned. The field of robotics is growing, and it is a field where novel applications such as the envisioned switching algorithm can be the most useful, since it can provide both reliability and a fast handover.

Moreover, ROS2 uses DDS as the communication protocol, which has a publish-subscribe paradigm. This aspect makes an easier to be migrated because, given a topic, there can be more than one publisher that uses it and published values can be received by more than one listener at a time. This aspect allows publishers and listeners to be replaced while the system is running, without requiring to change the communication details.

Kubernetes. When looking for solutions on how to orchestrate workloads across different machines, Kubernetes has been selected as the most suitable candidate. ROS2 processes are usually long-running and ROS2 allows configuring them to start automatically, but it does not include by default ways to orchestrate its processes dynamically, such as starting and stopping a service and managing replicas. Such functionality can be added also without using K8s, but K8s brings a cloud-native development model and additional tools to simplify the implementation of the task switching.

In our proposed solution, K3s is used on the robot and K8s is used on the remote server. The decision to use K3s on the robot is due to it being lightweight and integrating Flannel as the container networking interface (CNI). The reason why Flannel is preferred over solutions such as Cilium or Calico is that it does not stop multi-cast traffic when run on a single node, allowing for a simpler architecture.

Liqo. When deciding how to handle the task offloading and make it as seamless as possible, Liqo[5] was selected as the most suitable candidate for the following reasons:

- **Simplified management**: Two clusters can be managed as if they were a single cluster, transparently, in particular, in our configuration, the k3s cluster will be the main cluster which borrows resources from the cloud.

  It enables the use of Kubernetes-native features to handle the offloading, for instance the network policies, and allows us to have a unified control of the offloaded task state (e.g the two tasks can be selectively managed using their Fully Qualified Domain Name).
- **Security**: Liqo uses tunneling to connect robot and the remote cluster, securing the connection end-to-end.

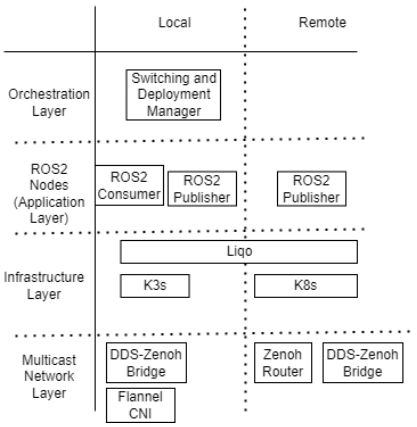

Figure 3: ROS2-based applications run on the high-level architecture shown in this picture.

5 ARCHITECTURE

The high level architecture of the system can be divided in four functional layers (see figure 3) that will be described in the following section.

Infrastructure layer. It is responsible for managing the system state, handling ROS2 nodes state and life cycle (K8s). In case of Multiple clusters it also manages the life cycle of the nodes on the cloud using Liqo.

Orchestration Layer. The main component of the orchestrating logic is also a process that from now on will be referred to as Switching and Deployment Manager (SDM). The SDM can be implemented in various ways, from a ROS2 node to a Kubernetes process, and for the purposes of this paper, it will only need to implement its main function, consisting of enacting the switching procedure. Its current implementation consists of a process that simply triggers the switch every 5 seconds.

ROS2 processes. They are the main logic of the robot. They communicate with each other using DDS mainly through a publish-subscribe paradigm. Some of them, by design, cannot be offloaded to the cloud. For instance the nodes interfacing directly with hardware, such as sensors and actuators on the robot can and will only be able to function if running on the robot itself. Other nodes, which main function does not involve direct communication with hardware but instead revolves around data processing will be able to be moved to the cloud, benefiting from a larger resource pool.

5.1 Multicast network layer

By default, ROS2 nodes talk to each other using DDS, which allows them to discover each other by using UDP multi-cast. Sadly, most K8s CNIs filter out multicast traffic, making communication between ROS2 nodes impractical. If all nodes run inside the same K8s pod in a single-node cluster, the communication is still able to function without additional steps. Instead, enabling the communication between different pods in the same or different clusters requires additional solutions.

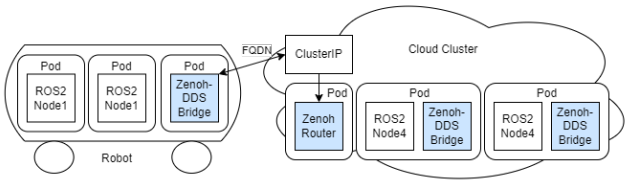

Figure 4: CNI-based Architecture used during our experiments. The Zenoh router is present only in the remote machine and the multicast network is expanded from the robot to the cloud via a single Zenoh bridge in the robot.

One downside of using K8s with ROS2 is that the CNI blocks multi-cast traffic, making the built-in DDS discovery protocol fail and ROS2 nodes are unable to talk to each other, even on the same cluster. The Flannel CNI has been found not to filter DDS traffic when run in a single-node K3s cluster, so it was chosen in order to extend the DDS multicast from the intra-pod to the K3s node level.

CNI-based solution. In order to allow seamless DDS discovery also in the switched mode, a Zenoh-based solution was devised, as depicted in figure 4. The choice of CNI on the robot side is something which makes sense to do as the developers usually have access to their development platform and can freely decide which CNI to include in it. Different is the case for the cloud, where managed solutions exist and there is no guarantee that tenants will be allowed to change CNI at their will.

The solution involves the usage of a single Zenoh bridge in a robot k3s pod, which connects to the Zenoh router located on the remote cluster. A Zenoh bridge has been used in each K8s pod in the remote cluster so as to connect any switched ROS2 process to the multicast network, independently on the choice of cloud CNI. The main advantages of this solution are:

- Transparency: Using Zenoh, ROS2 nodes need not to be changed, the solution is transparent as it does not involve additional modifications.
- Connection loss handling: In case the connection between local and remote cluster, the Zenoh bridge on the robot will automatically try to reestablish a connection with the remote cluster without additional configuration.

On the contrary, the disadvantage of this solution stands on the fact that the pod definition must be different in the robot and in the cloud as the ones in the cloud must include the *zenoh-dds-bridge* sidecar. This kind of asymmetry brings management complexity that will need to be addressed in future implementations.

CNI-independent solution. Another possible solution (figure 5 consists in handling the robot the same way the cloud has been handled in the previous solution. This has these advantages:

- It is a symmetric solution, which eases the management of the containers across the two clusters. A pod that works on the robot can be taken as-is and moved to the cloud.
- It does not rely on using a specific CNI that allows multicast, which makes it more portable and suitable not only for scenarios of the type "robot-cloud" but also to "cloud-cloud" where arbitrarily changing CNI may be problematic.

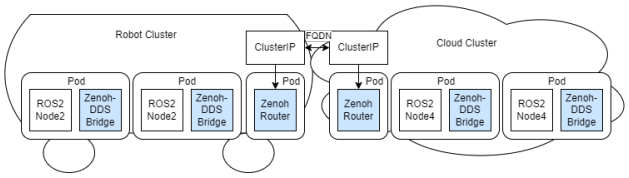

Figure 5: CNI-independent architecture. The architecture is completely symmetric, which makes it easier to manage but less efficient in terms of resources.

The two problems with this kind of solution are:

- Higher resource consumption on the robot because we need an additional pod and a sidecar for each of the ROS2 nodes.
- The peering between Zenoh routers requires that both routers know where to find the other and when connection is lost, both will try to reestablish it. This paradigm is not suited to the robot-cloud use case as it does not scale with the number of robots. In fact, in case of disconnection, we would like only the robot to try reestablishing the connection and not the cloud. Logically, the robot should behave as a client and not as a server.

5.2 System Modes

The designed system can be in two possible modes which will be explained in this subsection. One in which the computation is performed locally on the robot and one in which it is performed remotely on the cloud.

Default mode. In this default mode, all the logic runs locally on the single-node robot cluster (K3s) using flannel as the CNI of choice. This allows the ROS2 nodes to discover each other without additional efforts. This kind of mode can be used as a backup mode, in case the link quality robot-cloud becomes a limiting factor.

Switched/Offloaded/Remote mode. In the switched mode, the situation on the robot cluster remains similar to the one of the default mode. The main differences are: A subset of the ROS2 nodes have been moved to the cloud cluster, and communication between the nodes is transparently handed over by extending the multicast network layer. The multicast network layer has been extended using a solution based on the Flannel CNI, a set of Zenoh routers and Zenoh bridges, which will be explained more in detail later. The multi-cluster orchestration is enabled using Liqo, allowing the handover to be seamless. The cloud cluster is multi-node and uses K8s instead of K3s, since the cloud is less constrained resource-wise.

6 IMPLEMENTATION

This section will describe how the architecture previously described has actually been implemented, layer by layer, leveraging selected components delivered from the FLUIDOS[1] EU-funded project.

ROS2 processes. The experiments were performed using ROS2 nodes based on the *turtlesim* ROS2 package. In particular, the turtlesim simulation was run on the robot cluster to simulate the

[1] https://www.fluidos.eu/

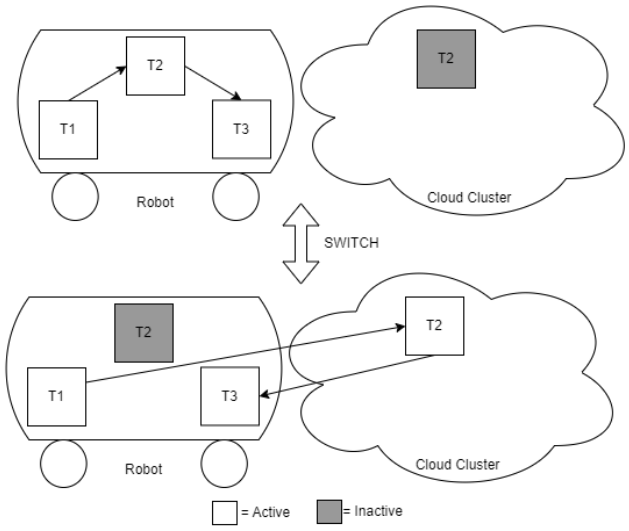

Figure 6: The local system runs three tasks by default. One of them is switched to run in the remote machine.

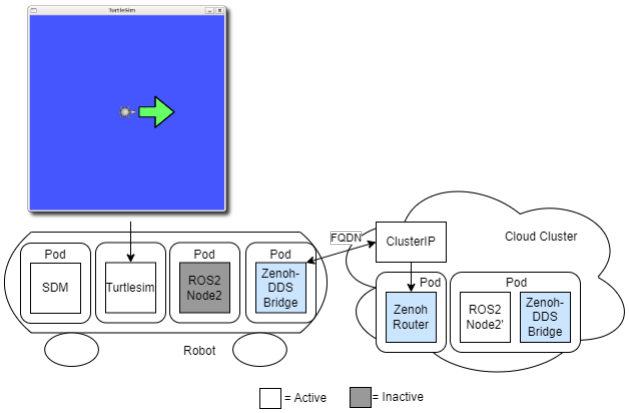

Figure 7: When the ROS publisher on the robot is active the turtle goes to the right.

robot actuators and a publisher on the topic *cmd_vel* was used as the ROS2 process to be offloaded. The processes involved in the switching differ in the value of the velocity being published, one of them moves the turtle to the right (figure 7), one of them moves the turtle to the left (figure 8).

Infrastructure. K8s and K3s were used to start the ROS2 nodes on the cloud and robot side respectively and handle their respective life cycles. Liqo was used as an higher-layer orchestrator to handle both clusters, in particular the K3s cluster (robot) has been configured as the main cluster and has been peered with the K8s one (cloud).

6.1 The switching process

Two SDMs were implemented, while referring to the ROS2 node/s to be switched I will refer to it as P_r in case it is the process on

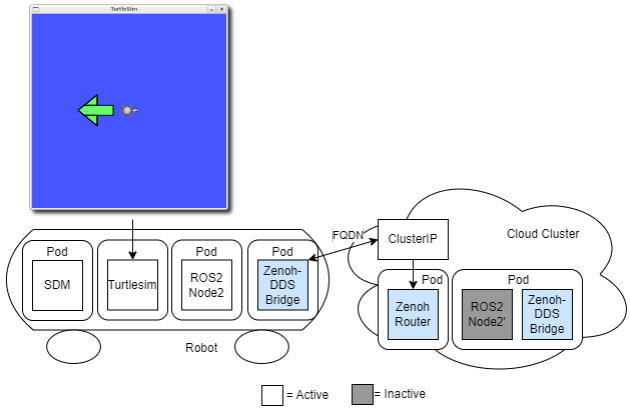

Figure 8: When the ROS publisher on the cloud cluster is active the turtle goes to the left.

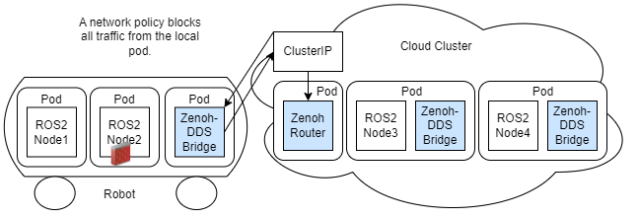

Figure 9: How the switched mode is enforced using network policies.

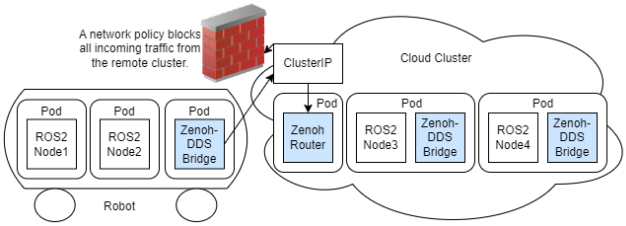

Figure 10: How the default mode is enforced using network policies.

the robot or P_c in in case it is the process on the cloud. The two implementations were developed in order to test which one of the two was actually better in performance.

Based on Network-policies. The change between default and switched mode is handled by using network policies. This solution has the advantage of not being tied to the ROS2 ecosystem and potentially work with a broader range of tasks. Network policies are an abstraction that can be used in Kubernetes in order to block traffic, which can be done either intra-pod, inter-pod or intra-namespace. In particular we need to handle two cases:

- **Default mode**: A network policy is enforced on the Liqo name space level, preventing communication from the cloud

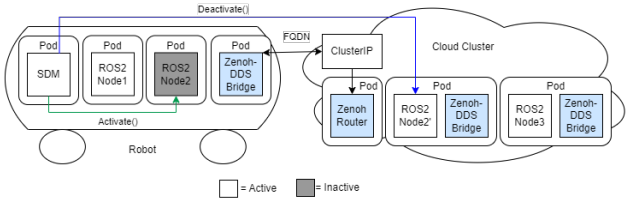

Figure 11: The image shows our system running in switched mode transitioning to local mode by using ROS2 lifecycle nodes.

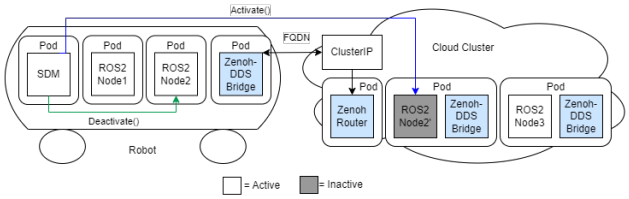

Figure 12: The image shows our system running in local mode transitioning to switched mode by using ROS2 lifecycle nodes.

Zenoh router service to the robot. This network policy prevents P_c from reaching the local cluster. No network policy is applied to the P_r (figure 10).

- **Switched mode**: A network policy is enforced on the pod as to prevent the P_r traffic to reach the multicast layer, effectively disabling P_r . No network policy is enforced on P_c, which allows it to reach the multicast network layer (figure 9).

One problem which arose during the implementation of such scenarios is that Flannel does not support network policies by default. It was possible to circumvent the problem by using kube-router [1], which already comes bundled with K3s and employs a network policy controller.

Based on ROS2 life cycle nodes. The change between default and switched mode can also be handled using a ROS2 feature called life cycle nodes. Life cycle nodes is implemented via a ROS2 service (DDS-based) on each ROS2 process that allows it to be deactivated and reactivated. By using such interface, the SDM is able to activate and deactivate the respective tasks. This solution is less general than the one based on network policies, but it is simpler to implement and it already takes care of deactivating the ROS2 process by putting it in a light-sleep state, which reduces resource consumption.

- **Default mode**: the SDM deactivates the remote task and enables the local one.
- **Switched mode**: the SDM deactivates the local task and activates the remote one.

7 CONCLUSIONS

The study was able to achieve three main results, which can be used as preliminary work toward a computing continuum that spans not only from a single robot to a datacenter, but from a robot to anything.

Two types of architectures. The results of this paper, that have been achieved in a virtualized, small-scale experiment, can potentially be extended to work on a much larger scale, not only in robot-to-cloud interactions but also in case of robot-to-robot communication. The study has shown and verified two architectures that enable ROS2 DDS communication generating on a local robot cluster to reach services located in a pod located in a different cluster located in the cloud or, potentially, another robot. Such architectures enable more robots to cooperate with each other because it can be used as a communication channel where to share information. Such a communication may be key to enable use cases like fleet management and the fusion of the perception data from different robots resulting in a common, shared understanding of a given scenario. Of course, many steps still need to be addressed in order to achieve such a result. Probably the biggest among them is security, as such architecture allows any robot to listen to what the others are communicating, making the problem of trust and confidentiality a major concern.

Two types of switching. The implementation of the switching algorithm explained in this paper demonstrated that it is actually possible to move long-lived ROS2 processes from one cluster to another one. Two possible implementations were tested and found to be working. As of the time of writing, no benchmarks are available to prove which of the two techniques provides the best performance and in which context.

Extending the architecture to support stateful tasks. The kind of approach explained in this paper can be used to handle stateless tasks that are CPU-bound or GPU-bound such as object recognition and other stateless AI applications. One key feature that is missing is the ability to switch the execution of tasks that require state information. This is a very important feature to be added to this solution in the future because there are many use cases involving stateful tasks (e.g. path-planning) that, in order to work correctly, need up-to-date contextual information. When one not only needs to replicate the task but also its execution state, many challenges arise, especially in case of multiple data replication sites, and in the presence of byzantine actors. To support such tasks the current architecture will need to be expanded.

Improving on the SDM. In order to accomplish the results of this paper, it was necessary for the SDM to have the ability of triggering and enacting the switching. New features will be added to this component in order to tackle the main service migration challenges such as the management of multiple Liqo peers and selecting the best peer where to locate the task based on the monitoring of parameters like link quality estimation or the resources available on the peer.

ACKNOWLEDGMENTS

A big thank you to Giuseppe Galluzzo, which master thesis laid the ground work for achieving the results outlined in this paper.

This work was partially supported by European Union's Horizon Europe research and innovation programme under Grant 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System).

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 117/2023.

REFERENCES

[1] 2024. cloudnativelabs/kube-router. https://github.com/cloudnativelabs/kube-router original-date: 2017-04-17T04:58:06Z.

[2] Cristopher Chiaro. 2023. *Latency-aware task scheduling in the cloud continuum.* laurea. Politecnico di Torino. https://webthesis.biblio.polito.it/29414/

[3] Tung V. Doan, Zhongyi Fan, Giang T. Nguyen, Hani Salah, Dongho You, and Frank H. P. Fitzek. 2020. Follow Me, If You Can: A Framework for Seamless Migration in Mobile Edge Cloud. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 1178–1183. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162992

[4] Hao Hao, Wei Ding, and Wei Zhang. 2024. Time-continuous computing offloading algorithm with user fairness guarantee. *Journal of Network and Computer Applications* 223 (March 2024), 103826. https://doi.org/10.1016/j.jnca.2024.103826

[5] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. 2022. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Transactions on Cloud Computing* (2022), 1–18. https://doi.org/10.1109/TCC.2022.3229163 arXiv:2204.05710 [cs].

[6] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics* 7, 66 (2022), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

[7] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. 2018. Live Service Migration in Mobile Edge Clouds. *IEEE Wireless Communications* 25, 1 (Feb. 2018), 140–147. https://doi.org/10.1109/MWC.2017.1700011 Conference Name: IEEE Wireless Communications.

[8] Yushen Wang, Tianwen Sun, Guang Yang, Kai Yang, Xuefei Song, and Changling Zheng. 2023. Safety-Critical Task Offloading Heuristics for Workflow Applications in Mobile Edge Computing. *Journal of Circuits, Systems and Computers* 32, 11 (July 2023), 2350186. https://doi.org/10.1142/S0218126623501864 Publisher: World Scientific Publishing Co..

[9] Huaming Wu, Yi Sun, and Katinka Wolter. 2020. Energy-Efficient Decision Making for Mobile Cloud Offloading. *IEEE Transactions on Cloud Computing* 8, 2 (April 2020), 570–584. https://doi.org/10.1109/TCC.2018.2789446 Conference Name: IEEE Transactions on Cloud Computing.

[10] Huaming Wu, Qiushi Wang, and Katinka Wolter. 2013. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 IEEE International Conference on Communications Workshops (ICC)*. 728–732. https://doi.org/10.1109/ICCW.2013.6649329 ISSN: 2164-7038.

Received 29 February 2024; revised 14 March 2024