FogROS2-FT: Fault Tolerant Cloud Robotics

†Kaiyuan Chen[1], Kush Hari[1], Trinity Chung[1], Michael Wang[2], Nan Tian[2], Christian Juette[2], Jeffrey Ichnowski[3], Liu Ren[2], John Kubiatowicz[1], Ion Stoica[1], and Ken Goldberg[1,4]

***Abstract*— Cloud robotics enables robots to offload complex computational tasks to cloud servers for performance and ease of management. However, cloud compute can be costly, cloud services can suffer occasional downtime, and connectivity between the robot and cloud can be prone to variations in network Quality-of-Service (QoS). We present FogROS2-FT (Fault Tolerant) to mitigate these issues by introducing a multi-cloud extension that automatically replicates independent stateless robotic services, routes requests to these replicas, and directs the first response back. With replication, robots can still benefit from cloud computations even when a cloud service provider is down or there is low QoS. Additionally, many cloud computing providers offer low-cost "spot" computing instances that may shutdown unpredictably. Normally, these low-cost instances would be inappropriate for cloud robotics, but the fault tolerance nature of FogROS2-FT allows them to be used reliably. We demonstrate FogROS2-FT fault tolerance capabilities in 3 cloud-robotics scenarios in simulation (visual object detection, semantic segmentation, motion planning) and 1 physical robot experiment (scan-pick-and-place). Running on the same hardware specification, FogROS2-FT achieves motion planning with up to 2.2x cost reduction and up to a 5.53x reduction on 99 Percentile (P99) long-tail latency. FogROS2-FT reduces the P99 long-tail latency of object detection and semantic segmentation by 2.0x and 2.1x, respectively, under network slowdown and resource contention. Videos and code are available at `https://sites.google.com/view/fogros2-ft`.**

I. Introduction

The complexity of foundational models [1], [2], [3] and sophisticated robot algorithms [4], [5] exceed most onboard robot computing capabilities. Cloud robotics provides shared access to on-demand resources and services with boosted performance and simplified management, enabling the deployment of compute-intensive algorithms on low-cost, mobile robots without powerful on-board hardware, such as GPU, TPU, and high-performance CPU. In previous research, we developed FogROS2, which enables unmodified robotics code in Robot Operating System 2 (ROS2) to offload heavy computing modules to an independent set of cloud hardware resources and accelerators. FogROS2 used *on-demand* servers that guarantee dedicated computing resources with high uptime (e.g., 99.99 % [6]). However, the network quality of service (QoS) between robots and the cloud can vary, and during rare cloud outages, robots lose all cloud-computing benefits. Additionally, as on-demand instances can be expensive, many cloud providers offer *spot VMs*[1] at a significantly reduced price with the caveat that they can shut down unpredictably—making them (without fault tolerance) unsuitable for many robotics applications. In this work, we introduce FogROS2-FT, a fault-tolerant extension to FogROS2 [7] that provides robust performance against variable network QoS, infrastructure unavailability, and stochasticity of the robotic algorithms, increasing the reliability and responsiveness of cloud robotics. By adding redundancy to cloud computation, we enable cloud-robotics tasks to continue operating effectively despite the following failures:

(A) *Resource Unavailability*: Although cloud services have high uptime and are managed by dedicated experts, outages can still occur. For example, an AWS outage affected the availability of the iRobot applications [8]. In addition, spot VMs may shut down unpredictably.

(B) *Resource Oversubscription*: The cloud enables flexible usage of computational resources. For example, one can oversubscribe to a system by allocating fewer resources than the sum of resources required by all robots, based on the

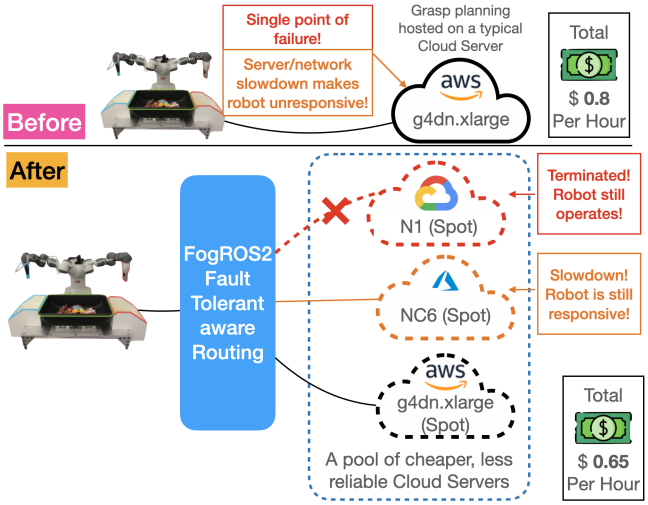

Fig. 1: **FogROS2-FT Overview. (Top)** Cloud robotics applications, such as grasp planning, when deployed on a single cloud server become a single point of failure. **(Bottom)** Instead, FogROS2-FT provides a cost-efficient and fault-tolerant solution that deploys unmodified ROS2 applications to multiple low-cost cloud servers, making cloud-robotics applications resilient to individual server termination and network slowdowns.

[1]Department of Electrical Engineering and Computer Science
[2]Robert Bosch Research and Technology Center North America, Sunnyvale, CA, USA
[3]Robotics Institute, Carnegie Mellon University
[4]Department of Industrial Engineering and Operations Research
[1,4]University of California, Berkeley, CA, USA
†For correspondence and questions: kych@berkeley.edu

[1]In Google Cloud Platform (GCP) and Microsoft Azure, these are called *Spot Virtual Machines*. In Amazon Web Services, these are called *Spot Instances*. More generally, they are also known as *preemptible* or *transient* machines.

expectation that robots rarely use all the resources simultaneously. Oversubscribing can improve resource utilization, but if too many robots contend for resources at once, all robots will experience performance degradation or failures.

(C) *Stochastic Algorithmic Latency*: Many robotic algorithms demonstrate stochastic timing behavior, such as asymptotically-optimal rapidly-exploring random tree in motion planning [9] that relies on random sampling. Large or foundational models are also affected by the stochasticity of the operating system scheduling and buffers [10], [11].

Fault tolerance typically demands an in-depth understanding of the algorithm and specific adaptations against faults, and it involves complex deployment and management processes that require extensive engineering expertise. FogROS2-FT provides fault tolerance for heterogeneous stateless robotic algorithms without requiring ROS2 application modifications. It simultaneously dispatches requests to identical services deployed in multiple cloud data centers and uses the first response received from the replicated services. This model significantly increases the probability of getting timely responses as long as at least one replica and network remains operational and responsive. We design a replication-aware routing network that allows resilient and location independent connectivity that persists even if the service is restarted on a different machine.

To reduce the cost of launching independent replicated services, FogROS2-FT can deploy on spot VMs. With FogROS2-FT, even time-sensitive robotics tasks can be executed on those cost-effective servers with per-request fault tolerant characteristics, enjoying the benefits of lower expenses without suffering from the drawbacks of unplanned server shutdown. FogROS2-FT resiliently manages a pool of spot VMs across multiple cloud and regions, and recreates a new spot VM whenever a replica gets terminated.

We evaluate FogROS2-FT on 4 cloud-robotics applications: object detection with YOLOv8 [12], semantic segmentation with Segment Anything (SAM) [2], motion planning with Motion Planning Templates (MPT) [9], and a physical pick-and-place task with a UR10e. In experiments, FogROS2-FT reduces latency by up to 1.16x in motion planning, including a 5.53x reduction on 99 Percentile (P99), a metric for long-tail latency faults; and FogROS2-FT reduces average inference latency by 2.1x and P99 latency of object detection by 3.9x under network slowdown. FogROS2-FT improves SAM's P99 latency by 1.96x when compute resources is contended.

This paper makes four contributions: (1) an open-source fault-tolerant extension to FogROS2 that enables robots to use replicated, independent cloud resources across different clouds for capacity and availability and stay available as long as one of the replicas and subnetworks is available; (2) cost-effective deployment using spot VMs; (3) cost, fault tolerance, and latency data from an experimental evaluation of FogROS2-FT on 3 simulated cloud robotics applications; (4) experimental evaluation with a physical robot performing a scan-pick-and-place task.

II. Related Work

Cloud and Fog Robotics: The use of cloud computing resources for robots, conceptualized as cloud robotics [13], has become increasingly relevant as large models (e.g., NeRF [14], SAM [2] and LERF [3] for visual perception) and other computationally demanding algorithms (e.g., MPPI for path planning) are integrated in robotic applications. Following the Fog Computing paradigm [15], Fog Robotics [16] utilizes edge resources to improve performance, of cloud computing for a multitude of robotics applications, including grasp planning [5], motion planning [4], visual servoing [17], and human-robot interaction [18]. In the FogROS2 series of work, we address several concerns of using cloud compute for robotics. FogROS2 [19] is a cloud robotics framework officially supported by ROS2 [20]. FogROS2 focused on optimizing for a single cloud and robot using a Virtual Private Network (VPN). Extensions of this work have addressed the questions of connectivity, latency, and cost. FogROS2-SGC (Secure and Global Connectivity) [21] enables secure communication between distributed ROS2 robot nodes.

Multi-Cloud Robotics: FogROS2 is the first multi-cloud robotics that offloads robotics applications to multiple cloud service providers. FogROS2-Config [22] extends FogROS2 to navigate available cloud machine selection that meets user-specified time and cost per request. FogROS2-FT uses multi-cloud Spot VMs to reduce the cost of the deployment. Spot VMs are one such cost-saving purchasing option offered by major cloud service providers that are up to a 90 % discount off the standard on-demand pricing, because they can be *preempted*[2] *unpredictably* [23]. This creates trade-off between the reduced price of spot VMs and having to build an infrastructure handling shutdowns and restarts [24]. The rates of preemption are highly variable across regions and instance types, from 3 % to over 20 % chance of preemption per day. Spot VM prices are also variable, and can fluctuate over the course of a day. Extensive research has been done to predict spot pricing using statistical models [25], [26] and learning methods [27], [28]. Due to their unreliable nature, spot VMs were not considered as part of the machine selection in FogROS2 and FogROS2-Config.

Fault Tolerance for Latency Sensitive Applications: A fault tolerant system is typically based on *failover*, dynamically switching to one of the machines upon failure of one machine, and *redundancy* by duplicated machines ensuring operation in case of a failure on one system. FogROS2-LS (Latency Sensitive) [29] implements failover strategy by enabling robots to flexibly connect with one of many servers, but the system takes time to discover and recover from faults by switching to another server that meets latency requirements. Lee *et al.* [30] enables fault tolerance on spot VMs for deep learning by actively checkpointing and recovering from the failure. On redundancy, Schafhalter *et al.* [31] improves the responsiveness of autonomous vehicles by performing operations on both vehicle and cloud. On fault tolerance of spot VMs, Voorsluys *et al.* [32], Poola *et al.*

[2]*Preempted*, *shut down* and *interrupted* are used interchangeably.

[33], and Neto *et al.* [34] demonstrate spot instances can be used as a cost-efficient setup. However, they focus on the cost optimization on a single server. Ali-Eldin *et al.* [35] uses redundancy of spot VMs on web services. However, fault tolerance of continuous and latency-sensitive robotic operations is understudied. With FogROS2-FT, off-the-shelf robotics applications can run in fault-tolerant environment without the awareness of the robotic application developers.

III. System Assumptions and Features

A. System Assumptions

We assume the application can be partitioned as a *robot* that sends sensor data as requests and awaits control instructions as response from a *service* that encapsulates the algorithm. We assume that all servers, networks, and faults are independent. The services need to be stateless to achieve fault tolerance transparently, or applications needs to use well-defined interfaces from FogROS2-FT to make the states consistent.

We assume the application is implemented in ROS2, the de-facto platform for building robotics applications. In ROS2, the computational units (*client* and *service*) are abstracted into *ROS2 nodes*. FogROS2-FT assumes the nodes are connected by ROS2 service communication model [36].

B. Fault Tolerance Properties

FogROS2-FT achieves fault tolerance with the following properties:

1) **Zero Downtime at Faults**. FogROS2-FT enables fine-grained request-level fault tolerance that ensures a request can be fulfilled as long as at least one replica and network remains operational.
2) **Algorithm-Agnostic**. FogROS2-FT operates independently of the specific algorithms used in applications as long as it is stateless.
3) **Failure Cause-Agnostic**. FogROS2-FT does not need to be tailored to specific failure, and remains functional as long as at least one service is available and connected.
4) **Hardware-Agnostic**. FogROS2-FT is agnostic to heterogeneous hardware resource options provided by the cloud, supporting simultaneous use of different resource types and fault tolerance level.
5) **Multi-cloud, Multi-region**. Since the failure may occur to a specific region or cloud provider, FogROS2-FT, as part of the Sky Computing paradigm [37] [38], offers a unified interface for interacting with various cloud service providers and uses cloud computing resources across different clouds simultaneously.

C. FogROS2-FT Failure Qualification

Multiple cloud servers can provide fault tolerance to region or server-specific compute or network failures. We assume the robot has a persistent and stable connection to at least one of the cloud servers.

The probability of a VM failing at any moment in time is

$$P_{\mathrm{VM}_i}(\text{failure}) = \frac{\text{recovery\_time}_{\mathrm{VM}_i}}{\text{uptime}_{\mathrm{VM}_i} + \text{recovery\_time}_{\mathrm{VM}_i}},$$

and the probability of a system failure with N spot VMs is

$$P_{\text{system}}(\text{failure}) = \prod_{i \in N} P_{\mathrm{VM}_i}(\text{failure}).$$

Given a desired maximum failure rate for the system and the failure rate for all VMs used, we can calculate the required number of VM replicas as

$$N = \left\lceil \frac{\log P_{\mathrm{VM}}(\text{failure})}{\log P_{\text{system}}(\text{failure})} \right\rceil.$$

To reduce the probability of failure, one can either increase the number of replicas, or use a combination of spot and on-demand instances. For example, a spot VM is preempted every 15 hours on average from experiments by Skypilot paper [38] and re-creating and initiating a new instance with FogROS2 can take up to 20-minute downtime [19], the probability of simultaneous preemptions with two spot VMs is less than 0.05 %, fulfilling the Service Level Agreement of AWS on non-spot VMs [6].

D. New Features

FogROS2-FT is distinguished from previous work with the following features:

1) **Cost-effectiveness**. FogROS2-FT can provide significantly lower cost by using spot VMs.
2) **Scalability** FogROS2-FT allows robots to launch a adjustable number of machines to reduce the probability that all the replica service deployments are not available.
3) **Flexibility** FogROS2-FT is flexible to the choice of available hardware resources. For example, one can mix low-specification on-demand cloud machines for up-time with high-specification spot VMs for speed.
4) **Adaptive and Resilient recovery** FogROS2-FT automatically recovers from service interruptions, preserving the intended level of fault tolerance automatically.

IV. FogROS2-FT Design

This section describes how FogROS2-FT (1) achieves transparent fault tolerance for ROS2 applications and (2) resiliently maintains a pool of cost-effective cloud servers.

A. Overview

Figure 2 shows an overview of how FogROS2-FT achieves both fault tolerance and cost effectiveness. FogROS2-FT sends requests to multiple replicated spot VMs, and routes the first response back to the robot. It resiliently manages spot VMs to recover from unpredictable terminations.

Interface. A user interfaces to FogROS2-FT is through extensions to the ROS2 launch system. The interface is identical to standard ROS2 launch scripts, other than specifying the hardware requirements and desired fault tolerance level. Listing 1 shows an example of a launch script for fault tolerant grasp planning service with FogROS2-FT. We embrace the Sky [37] multi-cloud paradigm that user can directly specify generic hardware requirements instead of specific cloud machine types, which are compatible across heterogeneous cloud service providers.

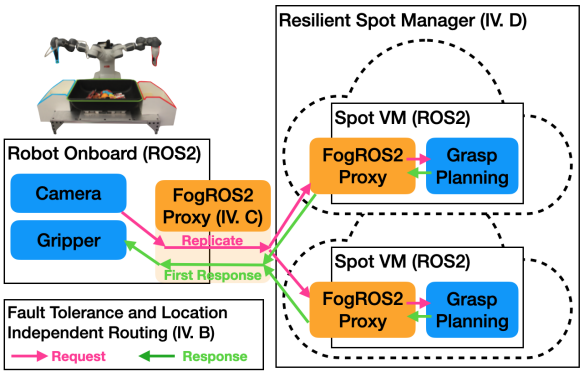

Fig. 2: **System Overview of FogROS2-FT** FogROS2-FT transparently proxies ROS2 communication. It sends requests to multiple replicated spot VMs, and routes the first response back to the robot. FogROS2-FT manages spot VMs to resiliently recover from unpredictable terminations.

```python
def generate_launch_description():
  # Define ROS2 nodes on cloud
  grasp_planner_node = Node(
    package="grasp", executable="planner")

  spec_1 = CloudMachine(
    region = "us-west-2",
    hardware = "cpu:16",  # 16 cores CPU
  )
  spec_2 = SpotMachine(
    region = "us-west-1",
    hardware = "T4",  # Nvidia T4 GPU # use Spot VM
  )

  # launch fault tolerant grasp_planner node
  # on the heterogenous regions and hardware
  FogCuster(
    cloud_nodes = [grasp_planner_node],
    deployments = [spec_1, spec_2])

  return LaunchDescription([
    Node(package="grasp", executable="client")
  ])
```

Listing 1: **FogROS2-FT Launch Script Example.** This example launches two nodes, a grasp motion client node that runs locally with standard ROS2 interface, and a fault tolerant grasp planning node with FogROS2-FT. It allow users to specify the cloud ROS2 nodes and desired fault tolerance level and hardware with mixed on-demand and spot VMs.

B. Application-Agnostic Fault Tolerance through Replication

To launch (unmodified) ROS2 nodes in the cloud, FogROS2-FT is a *multi-cloud launcher* that facilitates cloud initialization and *a replication-aware proxy* that connects ROS2 robot client and cloud service with a global fault-tolerant network.

The multi-cloud launcher initializes cloud servers with a ROS2 environment, and provisions secure communication through FogROS2-FT *proxies*. To local ROS2 network, the proxy serves the requests as local ROS2 service and interacts with nodes (e.g., sensors and controllers) on the robot as if the ROS2 service nodes on the cloud were all on the same local computer. On a new request, the proxy of FogROS2-FT sends the request to all replicated service nodes. When the proxy receives responses from the replicas, it passes only the first response to the robot. This is agnostic to applications and causes of the fault. As long as one response comes back to the robot, the request is fulfilled. To the cloud server, the proxy receives the request from the network, and invokes the ROS2 service on the cloud.

Multi-Cloud Fault Tolerant Launching Process. Initializing fault tolerant cloud robotics includes the following steps (1) The robot provisions multiple Cloud servers across different regions and data centers. FogROS2-FT can automatically select the region based on network latency and cloud operating cost. The user can override the selection with configuration. FogROS2-FT uses SkyPilot [38] to interface with heterogeneous cloud service providers. (2) The robot initializes all the cloud servers with ROS2 and the robot service application dependencies. FogROS2 [7] details the initialization procedure. (3) Robot and all Cloud servers generate and share communication security credentials (4) Robot and all Cloud servers run FogROS2-FT proxy (5) All proxies discover each other automatically, establish global and resilient connectivity with the generated security credentials and desired topology.

By default, all robotics services have at least two replicas at different cloud data centers. Critical services can use more replicas and adapt based on changing conditions (such as time of the day, budget). Without terminating the existing cluster, one can use Command Line Interface (CLI) to scale up and down the number of replicas dynamically:

```
ros2 fog scale [up/down] [args]
```

Replication-Aware Fault Tolerant Connectivity. Figure 3 shows the workflow of FogROS2-FT on handling requests with fault tolerance. A ROS2 application sends a request via the local ROS2 network, where the robot's proxy captures, extracts its serialized content a unique identifier from the ROS2 middleware layer (rmw). The proxy stores the identifier with a *handle* and sends the request to the cloud. The handle incorporates a callback function called on response arrival and on timeout. The cloud proxy invokes the corresponding cloud-based ROS2 service to compute and send the response back. The robot's proxy verifies the response with the unique identifier, then either marks the request as completed and sends back to the robot, or discards as a duplicate.

C. Resilient and Flexible Connectivity

The fault tolerance workflow of FogROS2-FT is established on a global peer-to-peer network, a fabric formed by interconnected proxies. The connections are resilient to service changes and flexible to various network topologies.

When a cloud machine is interrupted and rebooted, FogROS2-FT launches another cloud virtual machine. The launched cloud machine is typically different cloud machine with changed network connectivity information (such as IP address). The connections between proxies should be resilient to such interruption that even if the server gets interrupted and relaunched at a different physical machine, it can still maintain its connection to the robot.

FogROS2-FT achieves this by assigning a globally unique identifier to every peer-to-peer connection between proxies. This identifier can be generated deterministically between the proxies that no other ROS2 service can produce the same identifier. FogROS2-SGC [21] provides details about the identifier construction process and guarantees. In this work, we extend

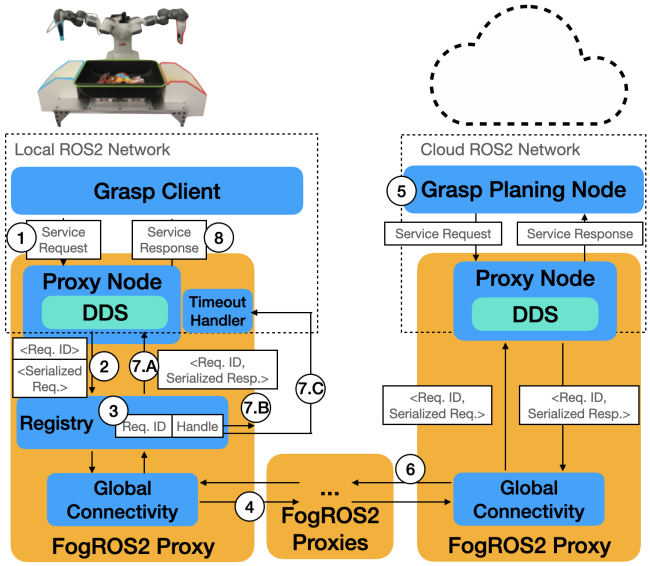

Fig. 3: **Flow diagram of FogROS2-FT on handling new requests** The FogROS2-FT replication-aware proxy handles ROS2 grasp planning request with fault tolerance guarantees with multiple steps. (1) The ROS2 application sends a request on the local ROS2 network. (2) The proxy running on the robot receives the request and extracts the content and a unique identifier from ROS2 middleware (rmw) layer buffer. (3) The proxy registers the unique ID from rmw with the *handle*, which includes a callback function if the response of the request arrives and a callback function for timeout. (4) The proxy securely sends the request to proxies running on replicated Cloud machines. There can be multiple proxy hops between the robot and the server that hosts the desired ROS2 service. The request message carries the unique identifier and the proxy adds an entry in the registry table. (5) The proxy running on the cloud converts the message to a standard ROS2 request message, and invokes the ROS2 service on the cloud and gets the response. (6) The proxy on the cloud sends the response back to the proxy on the robot. (7.A) The robot checks if it handled the response with the unique identifier; (7.B On the duplicated responses) The proxy drops the response if it was already handled. (7.C On timeout) The proxy calls the timeout handler (such as returns with empty response) and cleans up the registry table. (8) The robot sends the response to the application on the robot through standard ROS2 protocol.

the identifier of FogROS2-SGC to network connections, such that even if the original machine is interrupted and restarted at a different place, it can still deterministically generate the identifier and resume the connectivity. FogROS2-FT connects the proxies on the robot and on the cloud with the same globally unique identifier by a metadata server. The metadata server exchanges network information, monitors the status of the connections, and cleans up the connectivity information if one of the proxy reports its peer as disconnected. The metadata server can be hosted on lightweight, low-bandwidth and accessible cloud servers. The server only facilitates connectivity, and no application data flows through the server, so a failure of the server does not lead to a system failure.

Flexible and Scalable Topology For Bandwidth-Limited Robots. We consider mobile robots with low network bandwidth that may be bottlenecked if sending the request to multiple cloud servers. In this case, we need an intermediate server with higher bandwidth as gateway to forward the requests to other proxies. The server can be on the cloud or edge. The transport of FogROS2-FT is not constrained to having the client and server be directly connected. In FogROS2-FT, the proxy can be in a tree-like structure, where

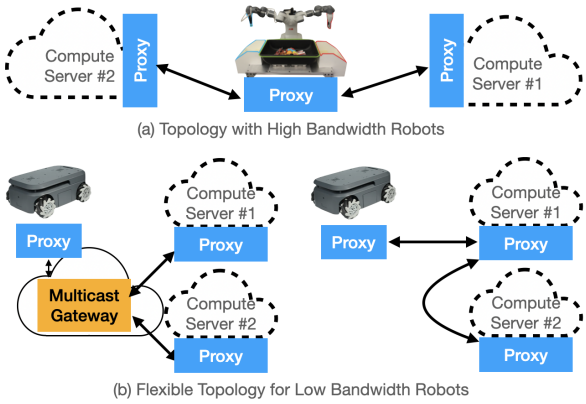

Fig. 4: **Flexible Topology for Different Bandwidth of Robots (a)** Since FogROS2-FT sends replicated requests to multiple cloud machines, it demands more network bandwidth than conventional cloud-robotics deployments. **(b)** FogROS2-FT allows flexible topology so that low-bandwidth robot can leverage cloud machines with higher bandwidth to forward to replicated services. One can either use dedicated gateway machine (left) or existing compute servers (right).

each edge the tree connects between an intermediate hop and the proxy. The proxy finds and establishes connections with its peers by flattening the tree. Figure 4 (b) shows two possible topology examples, which one can launch a lightweight gateway server to facilitate the message replication, or directly use a proxy on the compute service node.

D. Resilient and Cost-Efficient Spot VMs

FogROS2-FT can use spot VMs to reduce the cost of the replicated cloud server deployment. Since spot VMs can be up 90 % lower cost than their equivalent on-demand VMs, running two spot VMs at the same time can still enjoy as much as 80 % cost reduction. Earlier studies [39] show a bioinformatics task with 24 spot VMs on Google Cloud Platform experiences a preemption every 36 minutes on average. Because of the fault tolerance extension of FogROS2, we are able to guarantee per-request availability that as long as there is one spot VM not preempted and available to serve the request. FogROS2-FT launches and manages spot VMs across multiple cloud service providers and regions. FogROS2-FT regularly monitors the status of the spot VMs. When a spot VM is interrupted, FogROS2-FT re-launch the service node on a new spot VM and re-establishes connectivity.

V. Experiments

A. Setup

Without modifying the application code, we apply FogROS2-FT to three cloud robotics applications: visual object detection with YOLOv8 [12], Semantic Segmentation with Segment Anything [2] and motion planning with Motion Planning Templates (MPT) [9]. We use SkyPilot [38] to select servers given a hardware specification. For all the experiments, we use Amazon Web Services (AWS) spot VMs with two replicas in different regions, us-west-1 (California) and us-west-2 (Oregon). The workstation connects with spot servers with Ethernet connection.

Metrics. We compare FogROS2-FT with baseline single-server deployment by average latency collected by 100 trials

Application	Server Core	Single Server (USD per Hour)	FogROS2-FT (USD per Hour) On-Demand	Spot VM
YOLO	16	0.40 (1.17x)	0.80 (2.34x)	**0.34** (1x)
SAM	4*	0.53 (1.21x)	1.06 (2.42x)	**0.44** (1x)
MPT-Home	32	1.79 (1.05x)	3.58 (2.10x)	**1.69** (1x)
MPT-Cubicles	32	1.79 (1.05x)	3.58 (2.10x)	**1.69** (1x)
MPT-TwistyCool	64	3.58 (2.13x)	7.16 (4.26x)	**1.68** (1x)
Pick-Scan-Place	16	0.40 (1.17x)	0.80 (2.34x)	**0.34** (1x)

TABLE I: **FogROS2-FT hourly cost (USD per Hour) comparison to a Single-Server (one on-demand cloud machine)** FogROS2-FT can use two on-demand machines deployment or spot VMs. In SAM, we use Nvidia T4 GPU accelerator for the inference. FogROS2-FT is cheaper to run 2 spot VMs as opposed to a single server deployment up to 2.13x.

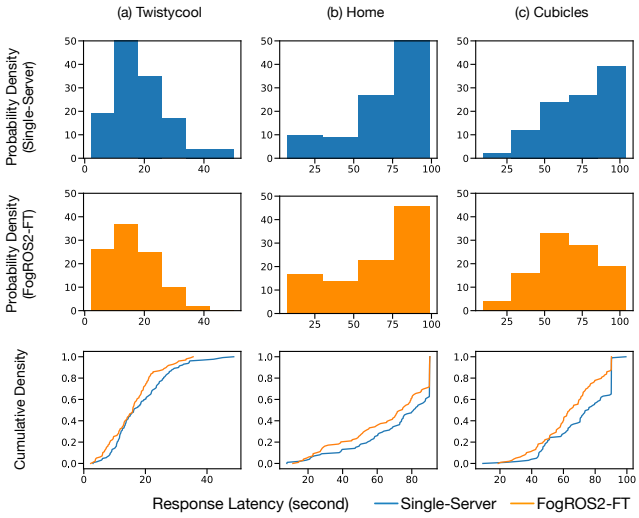

Fig. 5: **FogROS2-FT Latency on Motion Planning Template** We tested FogROS2-FT on 3 different motion planning environments (columns (a), (b), and (c)). Due to the stochastic nature of the algorithm, we aggregated results for each scenario and server configuration over 100 trials with a 100 s timeout. The **(top row)** shows the frequency histogram for the scenario when run with Single-Server (in blue). The **(middle row)** shows the frequency histogram for the scenarios when run on 2 servers with FogROS2-FT (in orange). With all scenarios, the shift left of FogROS2-FT histograms (in orange) relative to their corresponding single-server histograms (in blue) indicates improved latency performance when running on replicated servers. The **(bottom row)** compares the cumulative distribution functions (CDF) for single-server (in blue) and two servers (in orange). The two-server CDF is left relative to the single-server CDF indicating an overall improved performance with lower average latency for all scenarios.

for motion planning and 300 trials for robot vision tasks. We quantify *long-tail* anomalous latency faults with 99 Percentile (P99) latency, the runs with the top 1 % latency.

Cloud Cost Table I shows the US$ per hour cost of FogROS2-FT compared to typical cloud robotics single-server setup. With Spot VMs, FogROS2-FT is up to 2.13x cheaper than conventional single-server cloud robotics setup.

B. Case Study: Parallel Motion Planning

We perform a latency analysis of FogROS2-FT for parallel motion planning tasks on 3 different scenarios of varying complexity provided by Open Motion Planning Library (OMPL) [40]. Given a robot's initial state and goal state, the motion planner computes the waypoints required to move the robot to the goal while avoiding the obstacles. The motion planning algorithms we test are iterative optimization problems with stochastic solve times.

Latency Analysis We compare the latency probability

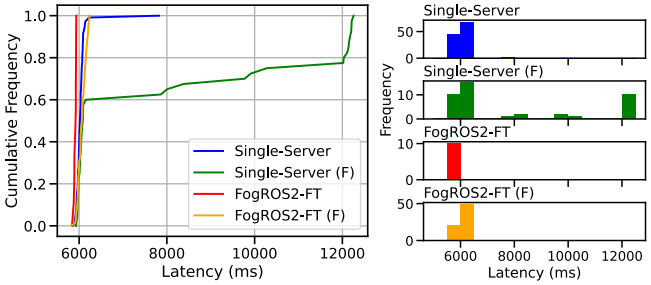

Fig. 6: **FogROS2-FT on Semantic Segmentation with SAM under Faults (F) by Compute Resource Oversubscription** FogROS2-FT improves 30 % of the long-tail P99 latency for SAM. As cloud GPU resources are typically oversubscribed and shared by multiple concurrent clients, we emulate such resource contention by running another periodic and concurrent client that alternates to send requests to FogROS2-FT server. FogROS2-FT significantly reduces the long-tail latency caused by the resource contention by 1.96x.

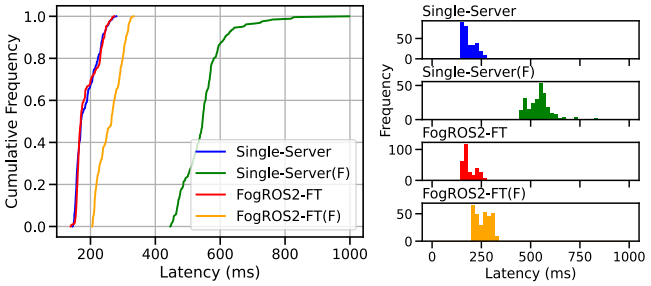

Fig. 7: **FogROS2-FT on Object Detection with YOLOv8 under Faults (F) by Regional Network Slowdown** We use FogROS2-FT to launch a multi-cloud cluster that includes us-west-1 (California) and us-west-2 (Oregon). FogROS2-FT and single-server deployments demonstrates similar latency under good network conditions, as the additional latency of sending to another replicated service is amortized FogROS2-FT using the first response. To simulate network congestion that can occur when a robot has trouble connecting to a specific data center, we introduce 100 ms latency on all our servers in us-west-1 (labeled with (F) for 'fault'). With the network slowdown, FogROS2-FT is 2.1x faster on average than non-fault tolerant deployment, and 3.9x faster on P99 latency.

between single-server and FogROS2-FT with 2 servers in Fig. 5. The results for the motion planning latency analysis are summarized in Figure 5. For all 3 scenarios, FogROS2-FT reduces average latency by up to 1.22x on Cubicles. FogROS2-FT significantly mitigates anomalous long and randomized latency by 1.42x (TwistyCool) and 5.53x (Cubicles) on P99 long-tail latency, because the probability of simultaneous anomalous high latency is rare.

C. Case Study: Robot Vision with YOLOv8 and SAM

We evaluate FogROS2-FT with SAM and YOLOv8. SAM is a computationally expensive algorithm that requires long computational time even with GPU. The result in Fig. 6 (Single-Server) shows the latency of running SAM on a typical setup with one cloud server, SAM shows a long-tail and significant P99 latency. FogROS2-FT improves the P99 latency by 1.3x by selecting the first response from the duplicated services. Compared to SAM, YOLO has lower inference latency, thus more network-intensive. In this case, FogROS2-FT demonstrates similar latency as Single-Server deployment for the tradeoff that FogROS2-FT consumes more network bandwidth but the additional latency is amortized by more stable response time.

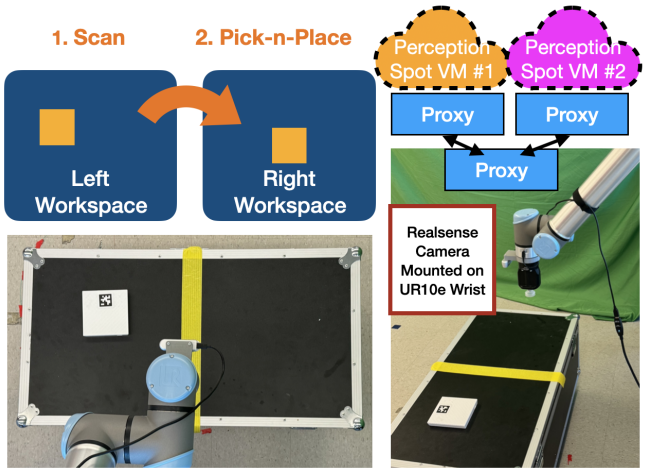

Fig. 8: **Physical Setup and System Diagram. (Left)** In the scan-pick-and-place robotic task, the goal is to move an object from one workspace to a randomly generated location on the other workspace. We repeat the task (from left to right, from right to left) for hours to show the continuity and robustness of FogROS2-FT in a physical robot system. **(Right)** The system diagram has two spot VMs providing Apriltag localization services to a UR10e robot with a RealSense Camera mounted on its wrist. The robot and camera connect to an edge computer that runs a robot motion planner and a FogROS2-FT proxy to connect to the replicated cloud services.

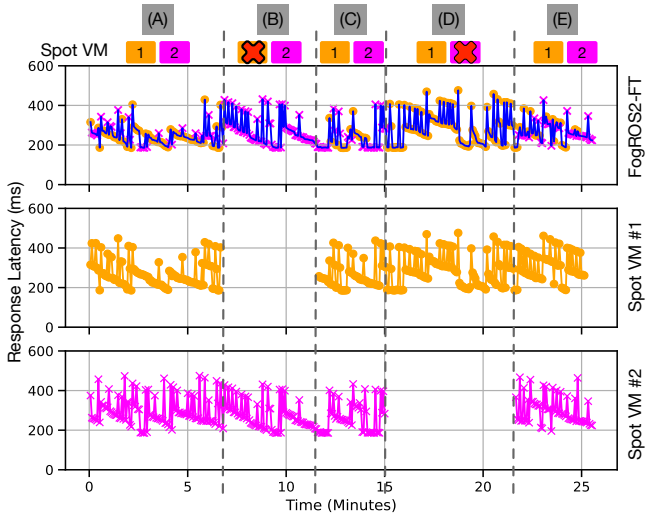

Fig. 9: **FogROS2-FT on Physical Fault Tolerant Pick-Scan-Place** (A) FogROS2-FT achieves reliable latency by choosing the first response with two active servers (B) We manually terminate one spot VM to emulate a fault, and FogROS2-FT uses the responses from spot VM #2. (C) FogROS2-FT automatically recovers spot VM #1 from failure by re-initializing user's environment and connectivity (D) We terminated spot VM #2 to emulate spot VM preemption (E) FogROS2-FT recovers and uses responses from both servers. The recovery downtime can be improved by tools from FogROS2 [7].

Faults at Compute Resource Oversubscription We evaluate FogROS2-FT against the faults of resource contention if many robots are contending on few resources, where one may choose to run multiple services on the same physical machine. By comparing faults (F) between Single-Server (F) and FogROS2-FT (F) in Fig. 6, FogROS2-FT reduces the latency by 1.31x and long-tail P99 latency by 1.96x.

Faults at Regional Network Slowdown. Sometimes robot may experience slowdown when connecting to a specific cloud data center. This can be caused by a periodic and regional network congestion or physical location. By comparing faults (F) between Single-Server (F) and FogROS2-FT (F) in Fig. 7, FogROS2-FT reduces the latency by 2.1x and long-tail P99 latency by 3.9x.

D. Case Study: Physical Scan-Pick-and-Place

We evaluate FogROS2-FT with cloud-based scan-pick-and-place with a fundamental physical robotic skill for many robot tasks, such as random bin picking, sorting, kitting, and conveyor belt pick-n-place. By offloading visual perception services, FogROS2-FT improves the responsiveness and reliability in spite of faults.

Experiment Setup Fig. 8 shows the physical setup of the Scan-Pick-and-Place evaluation. The Universal Robots arm (UR10e) is mounted with an Intel RealSense D435i on the wrist. We use a suction gripper to grasp a plastic CD player marked with an Apriltag. The system executes a cyclical procedure that begins with the robot moving to a predefined joint position. We implement a perception ROS2 service that uses Apriltag [41] for pose estimation. The ROS2 service takes in 640x480 resolution image frames and returns a 6D pose of the target, and the robot picking motion is generated and executed with the MoveIt2 motion planning tool on the edge computer, completing the feedback loop. The robot arm alternate to pick-and-place a box between the left section and the right section of a designated platform.

Latency Analysis We conducted the scan-pick-and-place repeatedly and manually interrupt the spot VMs to demonstrate the robustness of the system. Fig. 9 shows the latency timeline of FogROS2-FT on Physical Fault Tolerant Pick-Scan-Place. We manually terminated one of the two spot instances (scenario B and D). There are two takeaways: (1) FogROS2-FT can achieve reliable latency by choosing the first response with two active servers (2) FogROS2-FT can recover from server failures, such as termination. As long as one of the services is available and responsive, it can provide continuous operation for latency-sensitive applications.

VI. Conclusion

In this work, we explore concurrent execution across identical and stateless service deployments on different cloud machines with FogROS2-FT, and use the first received response to achieve the tolerance against independent faults. Evaluation shows FogROS2-FT can reduce the long-tail latency by up to 5.53 times. It can be deployed on cost-efficient spot VMs that the fault tolerant system can be up to 2.1x cheaper than conventional cloud robotics setup.

In future work, we will use multi-path connection, for example, having mobile robot to use 5G cellular connection, Wi-Fi and Ethernet connection simultaneously to avoid failures on single network connection. This resolves the assumption of FogROS2-FT that servers and faults to be independent and prevents the same network link between robot and cloud to be the single point of failure.

Acknowledgements

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR)

Lab. The authors were supported in part by donations from Robert Bosch Research. Cloud credits for experiments are provided by Amazon AWS.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] A. Kirillov *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[3] A. Rashid *et al.*, "Lifelong lerf: Local 3d semantic inventory monitoring using fogros2," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[4] J. Ichnowski *et al.*, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2020, pp. 4232–4238.

[5] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, IEEE, 2019, pp. 4559–4566.

[6] *AWS Service Level Agreement*, `https://aws.amazon.com/compute/sla/`, Accessed: 2024-02-15.

[7] J. Ichnowski *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," in *Proceedings IEEE International Conference on Robotics and Automation*, 2023.

[8] *IRobot Affected by AWS Outage*, `https://twitter.com/iRobot/status/1331667670383685635`, Accessed: 2024-02-15.

[9] J. Ichnowski and R. Alterovitz, "Motion planning templates: A motion planning framework for robots with low-power CPUs," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019.

[10] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu, "Rm3: On-device real-time deep reinforcement learning for autonomous robotics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2023, pp. 131–144.

[11] Y. Li, Z. Li, W. Yang, and C. Liu, "Rt-lm: Uncertainty-aware resource management for real-time inference of language models," *arXiv preprint arXiv:2309.06619*, 2023.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[13] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[14] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, "NeRF: Neural radiance fields without known camera parameters," *arXiv preprint arXiv:2102.07064*, 2021.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16.

[16] S. C. Gudi *et al.*, "Fog robotics: An introduction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[17] N. Tian *et al.*, "A fog robotic system for dynamic visual servoing," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1982–1988.

[18] S. L. K. C. Gudi, S. Ojha, B. Johnston, J. Clark, and M.-A. Williams, "Fog robotics for efficient, fluent and robust human-robot interaction," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2018, pp. 1–5.

[19] K. E. Chen *et al.*, "FogROS: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2021, pp. 2035–2042.

[20] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022.

[21] K. Chen *et al.*, "FogROS2-SGC: A ROS2 cloud robotics platform for secure global connectivity," *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2023.

[22] K. Chen *et al.*, "FogROS2-Config: A toolkit for choosing server configuration for cloud robotics," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[23] A. Alfaro, E. Carlson, K. Jarrett and R. Sheinberg, *Overview of Amazon EC2 Spot Instances*, `https://docs.aws.amazon.com/pdfs/whitepapers/latest/cost-optimization-leveraging-ec2-spot-instances/cost-optimization-leveraging-ec2-spot-instances.pdf`, Accessed: 2024-03-05, 2021.

[24] Z. Li *et al.*, "Spot pricing in the cloud ecosystem: A comparative investigation," *Journal of Systems and Software*, vol. 114, pp. 1–19, 2016.

[25] B. Javadi, R. K. Thulasiramy, and R. Buyya, "Statistical modeling of spot instance prices in public cloud environments," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp. 219–228.

[26] V. K. Singh and K. Dutta, "Dynamic price prediction for amazon spot instances," in *2015 48th Hawaii International Conference on System Sciences*, 2015, pp. 1513–1520.

[27] M. Baughman, C. Haas, R. Wolski, I. Foster, and K. Chard, "Predicting amazon spot prices with lstm networks," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ser. ScienceCloud'18, Tempe, AZ, USA: Association for Computing Machinery, 2018.

[28] V. Khandelwal, A. K. Chaturvedi, and C. P. Gupta, "Amazon ec2 spot price prediction using regression random forests," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 59–72, 2020.

[29] K. Chen *et al.*, "FogROS2-LS: A location-independent fog robotics framework for latency sensitive ROS2 applications," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[30] K. Lee and M. Son, "DeepSpotCloud: Leveraging cross-region GPU spot instances for deep learning," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, Jun. 2017, pp. 98–105.

[31] P. Schafhalter, S. Kalra, L. Xu, J. E. Gonzalez, and I. Stoica, "Leveraging cloud computing to make autonomous vehicles safer," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 5559–5566.

[32] W. Voorsluys and R. Buyya, "Reliable provisioning of spot instances for compute-intensive applications," in *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, 2012, pp. 542–549.

[33] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 4, Feb. 2016.

[34] J. P. A. Neto, D. M. Pianto, and C. G. Ralha, "MULTS: A multi-cloud fault-tolerant architecture to manage transient servers in cloud computing," *Journal of Systems Architecture*, vol. 101, p. 101 651, 2019.

[35] A. Ali-Eldin, J. Westin, B. Wang, P. Sharma, and P. Shenoy, "SpotWeb: Running latency-sensitive distributed web services on transient cloud servers," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19, Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 1–12.

[36] *Understanding ROS2 Services*, `https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html`, Accessed: 2024-03-15.

[37] S. Chasins *et al.*, *The Sky Above The Clouds*, arXiv:2205.07147 [cs], May 2022.

[38] Z. Yang *et al.*, "SkyPilot: An intercloud broker for sky computing," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 437–455.

[39] I. Stoica and S. Shenker, "From cloud computing to sky computing," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21, Ann Arbor, Michigan: Association for Computing Machinery, 2021, pp. 26–32.

[40] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[41] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 4193–4198.