

FogROS: An Adaptive Framework for Automating Fog Robotics Deployment

Kaiyuan Chen¹, Yafei Liang¹, Nikhil Jha¹, Jeffrey Ichnowski¹, Michael Danielczuk¹, Joseph Gonzalez¹, John Kubiawicz¹, Ken Goldberg^{1,2}

Abstract—As many robot automation applications increasingly rely on multi-core processing or deep-learning models, cloud computing is becoming an attractive and economically viable resource for systems that do not contain high computing power onboard. Despite its immense computing capacity, it is often underused by the robotics and automation community due to lack of expertise in cloud computing and cloud-based infrastructure. Fog Robotics balances computing and data between cloud edge devices. We propose a software framework, FogROS, as an extension of the Robot Operating System (ROS), the de-facto standard for creating robot automation applications and components. It allows researchers to deploy components of their software to the cloud with minimal effort, and correspondingly gain access to additional computing cores, GPUs, FPGAs, and TPUs, as well as predeployed software made available by other researchers. FogROS allows a researcher to specify which components of their software will be deployed to the cloud and to what type of computing hardware. We evaluate FogROS on 3 examples: (1) simultaneous localization and mapping (ORB-SLAM2), (2) Dexterity Network (Dex-Net) GPU-based grasp planning, and (3) multi-core motion planning using a 96-core cloud-based server. In all three examples, a component is deployed to the cloud and accelerated with a small change in system launch configuration, while incurring additional latency of 1.2s, 0.6s, and 0.5s due to network communication, the computation speed is improved by 2.6 \times , 6.0 \times and 34.2 \times , respectively. Code, videos, and supplementary material can be found at <https://github.com/BerkeleyAutomation/FogROS>.

I. INTRODUCTION

Power, weight, and cost considerations often mean robots do not include computing capabilities capable of running large-scale multi-core CPU-based, graphics processing unit (GPU)-based, field-programmable gate array (FPGA)-based, or tensor processing unit (TPU)-based algorithms. For example, a light-weight drone with an attached gripper that uses a GPU-based grasp-planning module to compute grasp points for picking up objects [3] or perching [33], requires access to a GPU that the drone would not have onboard. While nearby computers can provide the necessary computing capabilities, this practice can be complex to set up, scale, and is prone to over-provisioning. Instead, we propose a framework based on the *Fog Robotics* [11, 28, 40] idea of balancing between the compute available at the edge and in the cloud. This framework, *FogROS*, is an extension of the Robot

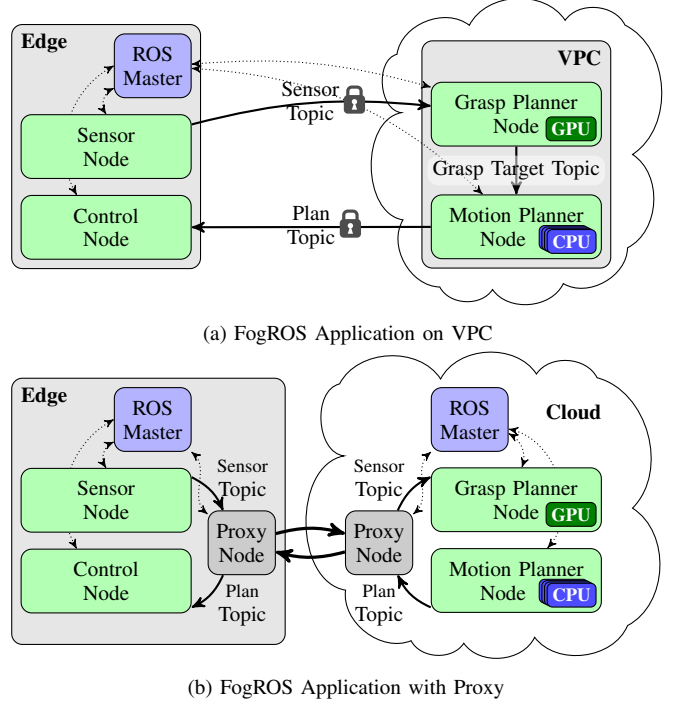


Fig. 1: **FogROS Applications and Communications.** ROS applications using FogROS run nodes on the cloud or on an edge computer with a small change to the configuration script. FogROS sets up the cloud computers and the secure communication channels between the edge computer and the cloud, either through a virtual private cloud (VPC) or through transparent proxying.

Operating System (ROS) [32] that, with minimal effort, allows researchers to deploy components of their software to the cloud, and correspondingly gain access to additional computing cores, GPUs, FPGAs, and TPUs, as well as predeployed software made available by other researchers.

ROS, at its core, is a platform in which software components (*nodes*) communicate with each other via a publication/subscription (pub/sub) system. Individual nodes can publish messages to named *topics* and subscribe to other named topics to get messages published by other nodes. In practice, these nodes all run on the robot and perhaps a nearby computer. For example, on a robot, a sensor node publishes to a sensor topic, a planning node subscribes to the sensor topic and compute a plan based on the sensor messages, and then publishes messages that another node uses to execute the plan (Fig. 1).

With FogROS, a researcher can use the same code, and

¹Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA ²Department of Industrial Engineering & Operations Research, University of California, Berkeley, CA, USA {kych, debbieliang-123, nikhiljha, jeffi, mdanielczuk, jegonzal, kubitron, goldberg}@berkeley.edu

make a small change to a configuration file to select components of the edge computer software to deploy to cloud-based computers. On launch, FogROS provisions a cloud-based computer, deploys the nodes to it, and then transparently passes the pub/sub communication between the edge computer and the cloud. The only observable differences are: (1) the pub/sub latency increases, and (2) the cloud-deployed components can compute faster given the additional computing resources. The increased latency means that not all components will benefit from being deployed to the cloud, in particular, any component with real-time requirements (e.g., a motor controller) or any component that requires little computing power, should not be deployed to the cloud. On the other hand, for many applications, the increased computation speed may enable new robot capabilities, speed up tasks, and allow for higher accuracy in tasks such as object detection or segmentation due to the use of larger models.

FogROS also supports launching pre-built automation container images. These container images contain all the software and dependencies required to run a program. To date, many academic and industrial open-source communities leverage container services, such as Docker [8], to distribute their applications. FogROS can deploy robot automation containers to the cloud without explicitly configuring the environment and hardware, facilitating ease of containerized software reuse.

This paper makes three contributions: (1) FogROS, an open-source extension to ROS that allows user-friendly and adaptive deployment of software components to cloud-based computers; (2) a method to pre-deploy containerized FogROS software that allows commonly-used software to be quickly integrated into applications; and (3) application examples evaluating the performance of FogROS deployment.

A. Design Principles

FogROS aims to adhere to the following design principles:

a) Transparent to software: FogROS should preserve ROS abstractions and interfaces. Applications should notice no difference between cloud-deployed and on-board nodes (other than the latency of message processing).

b) Flexible computing resources: Different nodes require different computing capabilities. Some nodes benefit from additional computing cores, while others benefit from access to a GPU. FogROS should make selecting the appropriate configuration simple.

c) Minimal configuration required: Running software nodes on a cloud-based computer should be as easy as running them on the edge computer.

d) Pre-deployed nodes: Some useful nodes require extensive setup, installation of a dependency structure, and may have conflicting dependency versions. FogROS should make it possible to use pre-deployed containerized software through configuration.

e) Flexible Networking: Different networking options may have different availability, performance, setup time, and costs associated. FogROS should allow the user to select the networking options best suited to their application.

f) Security and Isolation: The communication between cloud and on-board nodes should be secure, and FogROS should close ports that expose software to compromise.

II. RELATED WORK

Cloud computing has emerged as an attractive and economically viable [14] resource to offload computation for robot automation systems with minimal onboard computing power. Kehoe *et al.* [19] survey the capabilities, research potential, and challenges of cloud robotics, as well as applications such as grasp planning, motion planning, and collective robot learning, that might benefit from the computational power of the cloud. Grasp planning and motion planning have both shown to be amenable to cloud computation. Ben *et al.* [5], Nan *et al.* [28], and Li *et al.* [22] generate robot grasp poses in the cloud by implementing parallelizable Monte-Carlo sampling of grasp perturbations [6, 18, 20] while Mahler *et al.* [24] explore cloud grasp pose computation that maintains privacy of proprietary geometries. In motion planning, Lam *et al.* [21] introduce path planning as a service (PPaaS) for on-demand path planning in the cloud and use Rapyuta to share plans among robots. Bekris *et al.* [4] and Ichnowski *et al.* [12] both devise methods for splitting motion planning computation between the cloud and the edge computer [1, 11]. In addition to providing computing resources, the cloud can also facilitate sharing and benchmarking of algorithms and models between edge computers [39] for grasping, motion planning, or computer vision.

To leverage cloud resources, many in academia and industry endeavor to connect edge computers to the cloud. Example approaches include using SSH port forwarding [9] or VPN-based proxying [23] to support unmodified ROS applications to share a single ROS master. FogROS builds on these approaches, and adds automation of ROS node deployment to the cloud and a virtual private cloud (VPC), saving time over prior approaches that require manual configuration of network access rules and IP addresses. For example, setting up a VPN-based proxying requires more than 12 steps for configuration and 37 steps for verification [9]. The complex manual configurations scale poorly and are error-prone. ROSRemote [31] and MSA [43] replace the ROS communication stack with custom Pub/Sub designs. Although edge computers can communicate with nodes owned by other ROS masters, these systems require heavy code changes to ROS applications. FogROS, as an option, leverages rosbridge [7], an open-source webserver that enables an edge computer to interact with another ROS environment with JSON queries. Given diverse attempts to connect edge computers to the cloud, Wan *et al.* [42] and Saha *et al.* [35] call for a unified and standardized framework to handle cloud-robot data interactions. FogROS aims to be a painless solution to this open issue by allowing unmodified ROS applications to be launched on the cloud with minimal additional configurations.

Sharing a similar vision as FogROS, RoboEarth [41] is a successful example where edge computers share infor-

mation on the cloud. However, in their use cases, edge computers mainly use the shared database on the cloud, and do not benefit from the powerful cloud computing resources. Rapyuta [26] and AWS Greengrass [2] provide pipelines to deploy pre-built ROS nodes to edge computers or robots. Both platforms build ROS nodes or Docker images on the cloud, and push the built images to robots that are registered with their platforms. FogROS considers the reversed direction of Rapyuta and AWS Greengrass. Instead of pushing the computation from cloud to robots, FogROS is a lightweight platform that allows developers to rapidly prototype applications and gain quick access to extensive computing resources, without conforming to an additional framework.

III. BACKGROUND

In this section, we provide a brief background on the building blocks of FogROS, including (A) cloud-based computing, (B) ROS and its pub/sub system, and (C) how ROS-based robotic systems are configured and launched.

A. Cloud Computing

Cloud-based computing services, such as Amazon Web Services (AWS), Google Cloud, and Microsoft Azure, offer network accessible computers of various specifications to be rented on a per-time-unit basis. Setting up a service typically requires a one-time registration and a credit card. Registered users can setup, reconfigure, turn on, turn off, and tear down virtual computers in the cloud. This can be done either through a web-browser interface, or programmatically through a network-based application programming interface (API). Computer configuration options include: amount of memory, amount of processing cores, type and amount of GPUs, and inclusion of custom processing hardware such as field-programmable gate arrays (FPGAs) and tensor processing units (TPUs). FogROS uses the AWS cloud service API to setup a cloud-based computer, deploy ROS and the code, secure network communications, and then run the node.

B. ROS and Pub/Sub

In ROS, *nodes* (software components) communicate with each other using a pub/sub (publication and subscription) system. Nodes register as publishers and/or subscribers to named communication channels called *topics*. Each topic has message type that determines what data is sent over the channel. For example, a ROS node that monitors the joint state (e.g., angles) through sensors, would publish messages of type `JointState` on an appropriately named topic, and that topic would only contain `JointState` messages. When a node publishes a sequence of messages to a topic, all registered subscribers will receive the message in the same sequence they were published.

Coordination of the publishers and subscribers to topics is maintained by the ROS *Master* [34]. The ROS Master exposes network API that allows nodes to connect over a network and register/unregister themselves as publishers and subscribers to topics. During the registration process,

publishers get the current list of subscribers, and subscribers get the current list of publishers. Publishers may then connect directly to already-registered subscribers, and subscribers may connect directly to already-registered publishers.

Once a publishing and subscribing nodes are directly connected to each other¹, publishing nodes serialize message-specific data structure to a sequence of bytes and sends the bytes over the connection. When subscribing nodes receives the sequence of bytes, they deserialize the bytes to the message-specific data structure and process the message.

However, existing ROS pub/sub communication has several limitations: (1) all the nodes have to share the same master to communicate (inter-master communication is not supported by ROS pub/sub protocol stack, and one has to use out-of-band protocols for communicating across masters); (2) although it is possible to join nodes from multiple machines to share a single master, the communication for ROS is not secured, and users must configure security protocols.

C. ROS Launch Scripts

Robot systems can be comprised of a complex graph of nodes communicating with each other via pub/sub. To consolidate an automation system deployment into a single file, ROS supports a launch configuration file. This file specifies which nodes are to be launched by code entry point, and allows for optional remapping of topic names (e.g., so that code written to process a standard message type can produce it from a topic with a name not known/specified when the code was written). Listing 1 is an example launch script that launches a client node and a server node from the `mpt_ros` package locally.

Listing 1: ROS Launch Script Example

```
<launch>
  <!-- Run client node -->
  <node name="client" pkg="mpt_ros" type="client"
        output="screen" />

  <!-- Run server node-->
  <node name="server" pkg="mpt_ros" type="server"
        output="screen" />
</launch>
```

FogROS extends the launch script capabilities to allow the specification of which nodes to deploy in the cloud and on what machine type.

IV. APPROACH

To meet the design principles, FogROS (1) extends ROS launch scripts to include an option of where to deploy and run a ROS node, the only place that requires user configurations; (2) provisions cloud-based computers, securely pushes the code or containers to them, and runs the code; (3) sets up one of two networking options (VPC or Proxy) to transparently and automatically proxy the pub/sub communication between the edge computer and the cloud; (4) provides introspection infrastructure for monitoring network conditions; and (5)

¹As an implementation optimization, ROS nodes on the same machine can communicate via a shared-memory queue, instead of using a network.

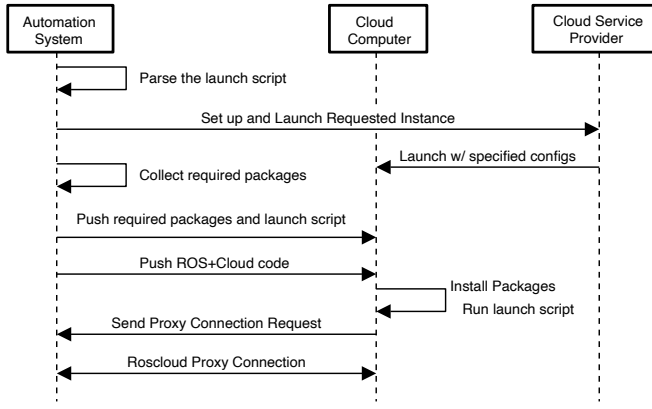


Fig. 2: Sequence Diagram of FogROS Deployment Process. Users only need to input the launch file, and FogROS automates the provisioning, deployment, code execution, and network setup sequence.

supports launching containerized FogROS nodes from pre-built Docker images.

A. Launch Script Extensions

FogROS uses standard ROS launch scripts as the user interface. Users specify which nodes are to be deployed and what type of cloud computing instance is used in the same launch file as the nodes that users want to deploy locally. They can push multiple nodes to the cloud at the same time by providing the path to a separate launch script. FogROS parses the launch script, finds and collects all the packages in the script, and pushes them to the cloud computer. As part of the configuration process, users can optionally specify a bash script that installs dependencies outside of the FogROS launch process (e.g., mirroring the steps to install dependencies on the edge computer).

Listing 2 provides an example of a FogROS launch script that serves the same functionality as Listing 1, but with the server node running on a cloud computer. Local ROS nodes, such as `client`, are launched as before. With FogROS, the user specifies the launch file that contains the server node (`server.launch`), the type of cloud computer (`c5.24xlarge`), and optionally, a setup script (`init.bash`).

Listing 2: FogROS Launch Script Example

```

<launch>
  <!-- Run client node as before -->
  <node name="client" pkg="mpt_ros" type="client"
    output="screen" />

  <!-- Run server node w/ FogROS-->
  <node name="server" pkg="fogros" type="
    fogros_launch.py" >
    <rosparam>
      instance_type: c5.24xlarge
      launch_file: server.launch
      env_script: init.bash
    </rosparam>
  </node>
</launch>
  
```

B. Cloud-Computer ROS Nodes

When FogROS launches cloud-based nodes, it performs the following sequence of steps that result in ROS nodes running on a cloud computer with messages being transparently proxied between the edge computer and the cloud computer:

- 1) Provision and start a cloud computer with the capabilities from the launch file and pre-loaded with ROS
- 2) Push code for ROS nodes to the cloud computer
- 3) Run the environment setup script
- 4) Set up secure networking (via Proxy or VPC)
- 5) Launch the pushed code

Before FogROS provisions a cloud computer, it uses the cloud service provider API to create security rules to set up a secure computing infrastructure suitable for ROS application configuration. It closes network ports not needed for communication between nodes. Then it provisions the cloud computer with a specified location and type. To speed up the launching process, FogROS specifies an image pre-loaded with the core ROS libraries to run on the cloud computer. As part of the launch process, FogROS generates and installs secure credentials on the cloud computer, and gets its public internet protocol (IP) address.

Once the computer is started, using the IP address and secure credentials, FogROS recursively copies the ROS code to the cloud computer securely over a secure shell (SSH) [44] connection, optionally runs the user-specified setup script, and builds the code on the cloud. With the code ready to run, FogROS then starts the configured secure networking components for VPC (Sec. IV-C) or proxying (Sec. IV-D), and runs the ROS nodes in the cloud.

C. Networking: Virtual Private Cloud

To allow the edge computer and cloud-based computers to communicate securely with each other, FogROS automates the setup of a Virtual Private Cloud (VPC). A VPC secures point-to-point communication between cloud computers by assigning private IPs that are only accessible for other nodes within the VPC. FogROS creates a Virtual Private Network (VPN) between the edge computer and the VPC. A VPN is a secure network communication channel provided by the operating system. With this setup, from the perspective of a ROS node, all nodes appear as though they are on the same private network.

FogROS automates the setup of the VPC and the VPN when it provisions the cloud computers to run the ROS nodes, by using the cloud service providers API to: (1) create a VPC instance and a security group for it, (2) establish credentials for the cloud-computers that will participate in the VPC, (3) configure the cloud computers to use the VPC for cloud-to-cloud communication, and (4) set up a VPN endpoint to which the edge computer will connect. Once set up, the cloud service provider manages the VPC, while FogROS manages the VPN. As part of the setup process, FogROS sets a unique private IP address for each of the computers participating, so that the ROS nodes can establish connections between computers.

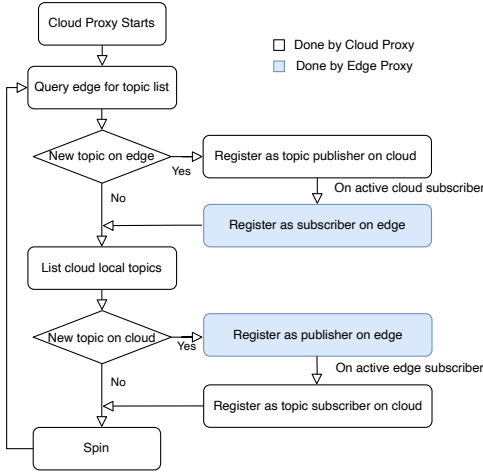


Fig. 3: Automatic Pub/Sub Proxy. The proxy tunnels the traffic only if there is an active subscriber.

D. Networking: Pub/Sub Proxying

In addition to VPC networking, FogROS also supports a proxied-network option that enable communication between the edge computer and the cloud. This option is available for cases where the VPC solution may be unavailable due to service provider restrictions or costs, or when an additional level of isolation between the edge computer and the cloud is desired. There are also performance differences (see Section V) when considering the network options, and a user of FogROS may wish to measure performance in their application before choosing a suitable option.

In FogROS, a proxy consists of two ROS nodes, one running in the edge computer and one running on the cloud. These nodes connect directly to each other via a secure network connection, and register as publishers and subscribers to topics on the ROS Master running on each computer. When a proxy node receives a message from a subscription, it sends it to the other proxy node, which then publishes it to the subscribers registered on its ROS Master.

There are two options for FogROS to identify topics to proxy: (1) user-specified in the configuration file, or (2) automated. If topics are specified by the user in the configuration file, FogROS subscribes and publishes to the topics specified. If the user does not specify topics, FogROS communicates with the ROS Master on each end and identifies which topics have registered subscribers and publishers. When a topic has a publisher on one end, and a subscriber on the other, the ROS proxy nodes coordinate with each other to proxy the associated topic (see Fig. 3). While the automated process is simpler to setup for the developer, it may result in increased setup time as the proxy nodes coordinate the setup of proxied topics, or wasted bandwidth on topics that do not need proxying.

E. Network Monitoring

With the proxying network option, FogROS also provides interfaces to monitor network conditions via ROS topics `/fogros/latency` and `/fogros/throughput`

on both the edge computer and the cloud. These interfaces do not introduce additional overhead unless an active subscriber subscribes to them. Users can also inspect and interact with ROS topics with standard ROS tools such as `rostopic`. In addition, FogROS provides the same fault tolerance as ROS running locally, where ROS nodes can re-join the pub/sub communication after network interruption.

F. Pre-Built ROS Nodes

FogROS also supports launching containerized ROS nodes with a similar interface to the FogROS launch script extension described in Section IV-A. While an increasing number of ROS developers are using pre-built docker images to host ROS nodes, this functionality is not natively supported by ROS. With FogROS, users can specify the name of a publicly-available image on DockerHub as well as the destination machine on which they want to launch it. FogROS then uses a template environment setup script to pull and run the image on the specified machine. It analyzes the machine type and configures the docker run command to match the hardware (e.g., GPU) available on the computer.

Listing 3 shows an example launch script for a Dex-Net grasp planning node in a docker image. FogROS provisions and starts a cloud computer (`g4dn.xlarge`) with a GPU, pulls the `dexnet:gpu` image from DockerHub [8], and attaches the docker container to the GPU, and runs it.

Listing 3: FogROS Docker Container Example

```

<launch>
  <!-- Dex-Net Docker Image w/ FogROS -->
  <node name="dexnet" pkg="fogros" type="
    fogros_docker.py" output="screen">
    <rosparam>
      docker_image: keplerc/dexnet:gpu
      machine_type: g4dn.xlarge
    </rosparam>
  </node>
</launch>

```

V. EVALUATION

Here we present three example applications on FogROS: (A) visual SLAM, (B) Dex-Net grasp planning and (C) multi-core motion planning. The nodes, topics, and split between a single-core edge computer with 2GB RAM and the cloud are shown in Fig. 4. In addition to showing the network latency and performance with FogROS, we highlight the simplicity and minimal configuration of deploying these applications.

A. Visual SLAM Service

ORB-SLAM2 [27] is a visual simultaneous localization and mapping system that uses monocular video input. In this experiment, a Camera Node publishes a 640×480 resolution video with each frame 48 KiB on average to the cloud (Fig. 4a). On the cloud an ORB-SLAM2 node subscribes to the video feed [37] and computes a pointcloud map along with the current estimated location within the map, which are sent back to the robot. For more details on the ORB-SLAM2 algorithm, we refer readers to the paper and open-source code available from Mur-Artal *et al.* [27].

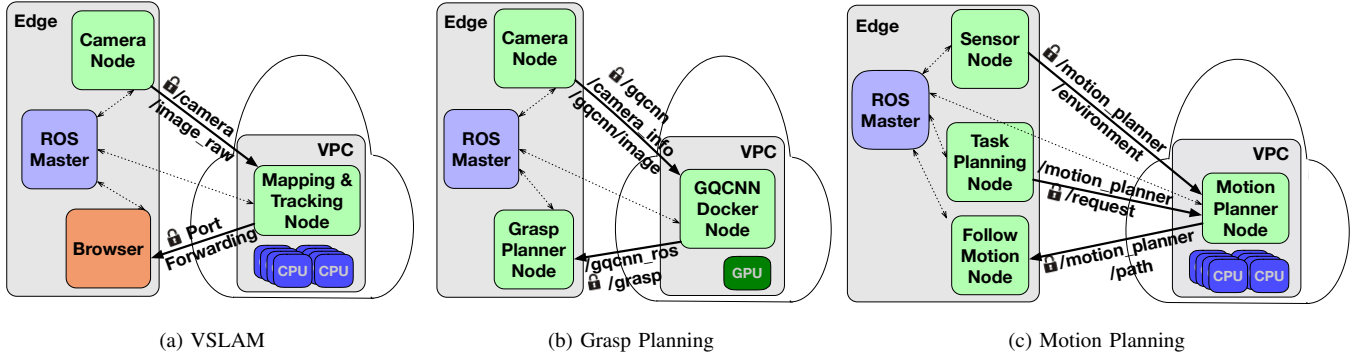


Fig. 4: **Example FogROS applications in experiments.** In experiments, we run 3 sample applications, each with one node accelerated by cloud computing, with the nodes and topics shown here. For brevity, we only depict the VPC-based solution here, but we also experiment with the proxy-based solution.

Scenario	Edge		Cloud — FogROS		Network	
	FPS	Create (s)	FPS	Create (s)	VPC (s)	Proxy (s)
fr1/xyz	16.6	2.9	25.2	1.2	0.4	0.6
fr2/xyz	14.7	2.1	30.2	0.8	0.4	0.6
fr1/desk	15.6	2.8	23.8	2.0	0.4	0.6

TABLE I: SLAM with FogROS. We benchmark FogROS on 3 different visual SLAM scenarios, and record the frame-per-second (FPS) and the latency in creating a new map (Create) in seconds on the local edge computer (using one-core CPU) and a cloud-computer with 36-core CPU. We also record the average network time in seconds for transmitting raw video frames to the cloud computer. FogROS demonstrates up to 2.0x improvement on FPS and 2.6x improvement on new map creation time than using only edge computer.

To configure FogROS to work with ORB-SLAM2, we build ROS docker images and push them to Dockerhub [8]. We wrote a bash script to pull and run the docker image and include its path as in Listing 2; FogROS then runs the script when configuring the environment. After initialization, FogROS sets up and secures communication between the robot and the cloud SLAM server.

To evaluate the performance of FogROS when deploying ORB-SLAM2, we compare the cloud-deployed performance to an edge-computer-only implementation. We select a 36-core cloud-computer (AWS c4.8xlarge) for the ORB-SLAM2 node, and compare it with ORB-SLAM2 running on a one-core edge computer. We report frames-per-second (FPS) and latency that creates the first map (in seconds) [29]. Table I suggests that cloud-based SLAM can achieve higher FPS, meaning that it can aggregate more data and produce higher quality maps in a real-time setting. Cloud-based SLAM also has less latency in generating a new map.

B. Dex-Net Grasping Service

Grasp analysis computes the contact point(s) for a robot gripper that maximize grasp reliability—the likelihood of successfully lifting the object given those contact points. To plan grasps on rigid objects in industrial bins using an overhead depth camera, we use an open-source implementation of the fully-convolutional grasp-quality convolutional neural network (FC-GQ-CNN) [25, 36] from Dex-Net[17]. We wrap FC-GQ-CNN in a ROS node and deploy it to the cloud along

with pretrained neural-network weights as a Docker image. We refer the reader to Satish *et al.* [36] and Mahler *et al.* [25] for details and code for the neural network and grasping environment.

This node subscribes to 3 input topics containing a scene depth image and mask for objects to be grasped, and a message of type `sensor_msgs/CameraInfo` containing camera intrinsics. Internally, the node feeds this to FC-GQ-CNN, which outputs a grasp pose and associated estimate of grasp quality. The node wraps these outputs, along with the gripper type and coordinates in image space, into a `gqcnn_ros/GQCNNGrasp` message, and publishes it.

While the node can be run both locally or in the cloud, using cloud GPU instances as opposed to a CPU for neural-network inference can greatly reduce computation time. In either case, the node is wrapped inside of a Docker container, reducing the need for resolving dependency issues between deep-learning libraries, CUDA, OS, and ROS versions. The pretrained models in the image is intended for a setup similar to that shown in Figure 5; variations in camera pose, camera intrinsics, or gripper type may require retraining the underlying model for accurate predictions.

We run FogROS with the Dex-Net docker image using the launch file in Listing 3. We compare grasp planning times across 10 trials using both the CPU onboard the edge computer and FogROS with the Docker images on the cloud. We also show compute times when using a compressed depth image format to transfer images instead of transferring raw images to the cloud directly. For the latter case, images are compressed and decompressed using the `republsh` node from the `image_transport` ROS package [13]. Table II shows the results for both compressed and uncompressed image transport between the nodes.

C. Multi-Core Motion Planning

Motion planning computes a collision-free motion for a robot to get from one configuration to another. Sampling-based motion planners randomly sample configurations and connect them together into a graph, rejecting samples and motions that are in collision. These planners can be scaled with additional computing cores.

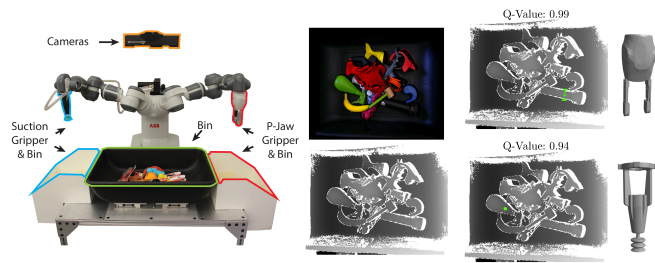


Fig. 5: **Grasp Planning:** A robot with an overhead depth camera and either a suction or parallel jaw gripper (left) must plan a grasp on one of the objects in the bin beneath it given an RGBD image observation (middle). Examples of planned grasps (green) and their q-values are shown for both parallel jaw and suction grippers (right).

Scenario	Edge	Cloud	FogROS VPC		FogROS Proxy	
	Only	Compute	Network	Total	Network	Total
Compressed	7.3	0.6	0.6	1.2	0.8	1.4
Uncompressed	7.5	0.6	0.7	1.3	0.9	1.5

TABLE II: Dex-Net Grasp Planning with FogROS. We benchmark Dex-Net on 10 trials, and record the compute time in seconds on the local edge computer (using CPU only), and the total (compute + network) time for using a 4-core cloud computer with a single Nvidia T4 GPU. FogROS demonstrates up to 6.0x improvement with VPC and 5.7x improvement with proxy than using only edge server.

Using FogROS, we deploy a multi-core sampling-based motion planner [10, 16] to a 96-core computer in the cloud to solve motion planning problems from the Open Motion Planning Library (OMPL) [38] (see Fig. 6). This planner node subscribes to topics for the collision model of the environment and motion plan requests (Fig. 4c). When the planner node receives a message on any of these topics, it computes a motion plan, and then publishes it to a separate topic. For more details on the multi-core motion planner, we refer the reader to the paper and the open-source code by Jeffrey *et al.* [16]. To configure FogROS to work with multi-core motion planner, we record the steps we use to setup the dependencies (e.g., FCL [30] and Nigh [15]) in a script. By providing the script, we configure FogROS similar to Listing 2.

We compare the planning time as the difference between publishing a motion plan request message, and receiving the plan result message, and show the results in Table III. The same motion planning problem is solved in a fraction of the time on the cloud when compared to using the edge computer. However, when the network latency (between 0.3 s and 0.6 s) is longer than the motion planning computing

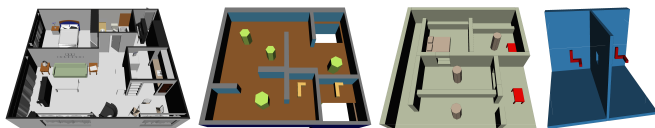


Fig. 6: **Motion Planning Scenarios.** We run OMPL [38] motion planning problems as benchmarks. Left-to-right: Apartment, Cubicles, Home, and Twistycool. In these problems, the robot is a rigid-body object that must, through rotation and translation, find a collision-free path through the environment, from a start pose to a goal pose.

Scenario	Edge	Cloud	FogROS VPC		FogROS Proxy	
	Only	Compute	Network	Total	Network	Total
Apartment	157.6	4.2	0.4	4.6	0.7	5.0
Cubicles	35.8	1.4	0.3	1.7	0.6	2.1
Home	161.8	6.2	0.3	6.5	0.6	6.8
TwistyCool	167.9	5.1	0.4	5.5	0.6	5.7

TABLE III: Multi-core Motion Planning with FogROS. We benchmark FogROS on 6 different motion planning scenarios using the same multi-core motion planner, and record the compute time in seconds on the local edge computer, and the total (compute + network) time for using a 96-core computer in the cloud. FogROS demonstrates up to 34.2x improvement with VPC and 31.52x improve with proxy than using only edge server.

time. For simpler planning problems, there may be little to no benefit to a cloud deployment. If the motion planner is asymptotically-optimal (finds shorter/better plans the longer it runs and with more CPU cores), then one could potentially run the motion planner for the same amount of time but get a better path using the cloud. Anand *et al.* [1] explored and shown the benefit of using the tradeoff between more cores and the resulting motion plan optimality.

VI. CONCLUSION

We present FogROS, a user-friendly and adaptive extension to ROS that allows developers to rapidly deploy portions of their ROS system to computers in the cloud. FogROS sets up a secure network channel transparent to the program code, allowing applications to be split between edge and the cloud with little to no modification. In experiments, we show that the added latency associated with pushing software components to the cloud is can be small when compared to the time gained from using high-end computers with many cores and GPUs in the cloud. However, in some simple tasks, using high-end cloud servers may lead to marginal benefits and can be considered as an overkill.

In future work, we will address the interactions of multiple hardware systems with different ROS masters, and handle the decentralized communication efficiently and securely. We will also support real-time compression on the proxy connection between edge computer and cloud to help reduce latency especially on low-bandwidth connections.

ACKNOWLEDGEMENTS

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, and the CITRIS “People and Robots” (CPAR) Initiative. The authors were supported in part by donations from Google, Siemens, Toyota Research Institute, Autodesk, Honda, Intel, Hewlett-Packard, and VMware. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1752814 and NSF/VMware Partnership on Edge Computing Data Infrastructure (ECDI), NSF award 1838833. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. We thank our colleagues who provided helpful feedback and suggestions, in particular Joseph M. Hellerstein and Alvin Cheung.

REFERENCES

- [1] R. Anand, J. Ichnowski, C. Wu, J. M. Hellerstein, J. E. Gonzalez, and K. Goldberg, "Serverless multi-query motion planning for fog robotics," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, IEEE, 2021.
- [2] AWS IoT Greengrass, <https://aws.amazon.com/greengrass/>, Accessed: 2021-02-15.
- [3] S. B. Backus, L. U. Odhner, and A. M. Dollar, "Design of hands for aerial manipulation: Actuator number and routing for grasping and perching," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2014, pp. 34–40.
- [4] K. Bekris, R. Shome, A. Kroutiris, and A. Dobson, "Cloud automation: Precomputing roadmaps for flexible manipulation," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 41–50, 2015.
- [5] K. Ben, M. Akihiro, C. Sal, K. James, and G. Ken, "Cloud-based robot grasping with the google object recognition engine," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2013, pp. 4263–4270.
- [6] K. Ben, B. Dmitry, and G. Ken, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2012, pp. 576–583.
- [7] C. Crick, G. Jay, S. Osentoski, and O. C. Jenkins, "Ros and rosbridge: Roboticians out of the loop," in *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2012, pp. 493–494.
- [8] Docker, <http://docker.com/>, Accessed: 2021-02-15.
- [9] S. S. H. Hajjaj and K. S. M. Sahari, "Establishing remote networks for ROS applications via port forwarding: A detailed tutorial," *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, p. 1729881417703355, 2017.
- [10] J. Ichnowski and R. Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Trans. Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.
- [11] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2020, pp. 4232–4238.
- [12] J. Ichnowski, J. Prins, and R. Alterovitz, "Cloud-based motion plan computation for power-constrained robots," in *Workshop on the Algorithmic Foundation of Robotics (WAFR)*, Springer, 2016.
- [13] *image_transport*, wiki.ros.org/image_transport.
- [14] I. Jeffrey, P. Jan, and A. Ron, "The economic case for cloud-based computation for robot motion planning," in *Robotics Research*, Springer, 2020, pp. 59–65.
- [15] I. Jeffrey and A. Ron, "Concurrent nearest-neighbor searching for parallel sampling-based motion planning in SO(3), SE(3), and Euclidean spaces," in *Workshop on the Algorithmic Foundation of Robotics (WAFR)*, Springer, 2018.
- [16] —, "Motion planning templates: A motion planning framework for robots with low-power CPUs," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019.
- [17] M. Jeffrey, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," in *Proc. Robotics: Science and Systems (RSS)*, 2017.
- [18] B. Kehoe, D. Berenson, and K. Goldberg, "Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip," in *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*, 2012, pp. 1106–1113.
- [19] 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.
- [20] B. Kehoe, D. Warrier, S. Patil, and K. Goldberg, "Cloud-based grasp analysis and planning for toleranced parts using parallelized Monte Carlo sampling," *IEEE Trans. Automation Science and Engineering*, vol. 12, no. 2, pp. 455–470, 2014.
- [21] M.-L. Lam and K.-Y. Lam, "Path planning as a service PPaaS: Cloud-based robotic path planning," in *Proc. IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, 2014, pp. 1839–1844.
- [22] P. Li, B. DeRose, J. Mahler, J. A. Ojea, A. K. Tanwani, and K. Goldberg, "Dex-Net as a service (DNaaS): A cloud-based robust robot grasp planning system," in *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*, 2018, pp. 1420–1427.
- [23] J. Z. Lim and D. W.-K. Ng, "Cloud based implementation of ROS through VPN," in *Int. Conf. on Smart Computing & Communications (ICSCC)*, IEEE, 2019, pp. 1–5.
- [24] J. Mahler, B. Hou, S. Niyaz, F. T. Pokorny, R. Chandra, and K. Goldberg, "Privacy-preserving grasp planning in the cloud," in *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*, 2016, pp. 468–475.
- [25] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, eaau4984, 2019.
- [26] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Trans. Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, 2014.
- [27] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [28] T. Nan, M. Matthew, M. Jeffrey, Z. Y. Xiang, S. Samantha, C. Christopher, Z. Steven, L. Qiang, Z. Robert, and G. Ken, "A cloud robot system using the dexterity network and Berkeley robotics and automation as a service (BRASS)," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2017, pp. 1615–1622.
- [29] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. Kelly, A. J. Davison, M. Luján, M. F. O'Boyle, G. Riley, et al., "Introducing slambench, a performance and accuracy benchmarking methodology for slam," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2015, pp. 5783–5790.
- [30] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2012, pp. 3859–3866.
- [31] A. B. M. Pereira, R. E. Julio, and G. S. Bastos, "ROSRemote: Using ROS on cloud to access robots remotely," in *Robot Operating System (ROS)*, Springer, 2019, pp. 569–605.
- [32] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009, p. 5.
- [33] P. Ramon-Soria, A. E. Gomez-Tamm, F. Garcia-Rubiales, B. C. Arrue, and A. Ollero, "Autonomous landing on pipes using soft gripper for inspection and maintenance in outdoor environments," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 5832–5839.
- [34] *ROS Master*, <http://wiki.ros.org/rosmaster>.
- [35] O. Saha and P. Dasgupta, "A comprehensive survey of recent trends in cloud robotics architectures and applications," *Robotics*, vol. 7, no. 3, p. 47, 2018.
- [36] V. Satish, J. Mahler, and K. Goldberg, "On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks," *IEEE Robotics & Automation Letters*, 2019.
- [37] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.
- [38] 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.
- [39] A. K. Tanwani, R. Anand, J. E. Gonzalez, and K. Goldberg, "RILaaS: Robot inference and learning as a service," *IEEE Robotics & Automation Letters*, vol. 5, no. 3, pp. 4423–4430, 2020.
- [40] 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.
- [41] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfiring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, et al., "RoboEarth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.
- [42] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *IEEE Access*, vol. 4, pp. 2797–2807, 2016.
- [43] B. Xu and J. Bian, "A cloud robotic application platform design based on the microservices architecture," in *Int. Conf. on Control, Robotics and Intelligent System*, 2020, pp. 13–18.
- [44] T. Ylonen and C. Lonvick, "The secure shell (SSH) protocol architecture," RFC Editor, RFC 4251, Jan. 2006, pp. 1–29.