
Kaiyuan (Eric) Chen1, Yafei Liang1, Nikhil Jha1, Jeffrey Ichnowski1, Michael Danielczuk1, Joseph Gonzalez1, John Kubitowicz1, Ken Goldberg1,2

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, that allows existing applications to gain access to additional computing cores, graphics-processing units (GPUs), field-programmable gate arrays (FPGAs), and tensor-processing units (TPUs) available on commercial cloud-based services. This framework is built on Robot Operating System (ROS), the de-facto standard for creating robot automation applications and components. 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 (SLAM), (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 1.2 s, 0.6 s, and 0.5 s due to network communication, the computation speed is improved by 4.5×, 5.2× and 31.5×, 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 multicore CPU-based, graphics processing unit (GPU)-based, or tensor processing unit (TPU)-based algorithms. For example, a lightweight drone with an attached gripper that uses a GPU-based grasp-planning module to compute grasp points for picking up objects [2] or perching [26], 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 an extension to the Robot Operating System (ROS) 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 communicate with each other via a publication/subscription (pub/sub) system. Individual components can publish messages to named topics and subscribe to other named topics to get messages published by other components. For example, a planning component might subscribe to a sensor topic, compute a plan based on data contained in the sensor messages, and then publish messages that another component would then use to execute the plan (Fig. 1).

With FogROS, a researcher can use the same code, and 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 software components to it, and then transparently passes the pub/sub communication between the edge computer and the cloud. The only observable...
has emerged as an attractive and economically viable [11] computing models in robot automation, cloud computing should be secure, and FogROS should close ports that expose networking options best suited to their application. Costs associated. FogROS should allow the user to select the networking options that are best suited to their application.

Some useful software components 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 references in the launch file.

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

d) Pre-deployed components: Some useful software components 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 references in the launch file.

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 channel should be secure, and FogROS should close ports that expose software to compromise.

II. RELATED WORK

With the recent abundance of deep learning and parallel-computing models in robot automation, cloud computing has emerged as an attractive and economically viable [11] resource to offload computation for systems with minimal onboard computing power. Kehoe et al. [16] 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 greatly from the computational power of the cloud. Grasp planning and motion planning have both shown to be amenable to cloud computation. Kehoe et al. [15], Tian et al. [33], and Li et al. [19] generate robot grasp poses in the cloud by implementing parallelizable Monte-Carlo sampling of grasp perturbations [13, 14, 17] while Mahler et al. [21] explore cloud grasp pose computation that maintains privacy of proprietary geometries. In motion planning, Lam et al. [18] 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. [3] and Ichnowski et al. [10] both devise methods for splitting motion planning computation between the cloud and the edge computer [9]. In addition to providing computing resources, the cloud can also facilitate sharing and benchmarking of algorithms and models between edge computers [32] 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 [6] or VPN-based proxying [20] to support unmodified ROS applications to share a single ROS master. FogROS builds on the 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 [6]. The complex manual configurations scale poorly and are error-prone. ROSRemote [25] and MSA [36] 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 [4], 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. [35] and Saha et al. [28] call for a unified and standardized framework to handle cloud-robot data interactions. FogROS seeks for 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 [34] is a successful example where edge computers share information on the cloud. However, in their use cases, edge computers mainly use the shared databases on the cloud, but don’t benefit from the powerful cloud computing resources. Rapyuta [23] aims to offload computation to the cloud and it is extensible to 3000+ ROS packages. With a Master Task set, Rapyuta is also able to dynamically allocate computing units for tasks. However, while Rapyuta is running on the cloud, it serves as a connection between edge computers
and ROS nodes. Compared to Rapyuta, FogROS is lighter-weight. FogROS places one ROS node on the edge computer and deploys another on the cloud to serve as the point of contact on each side, saving the developer time from setting up middleware infrastructure. While Rapyuta allows businesses to build and deploy an entire pipeline for their robotic applications, FogROS 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 (1) cloud-based computing, (2) ROS and its pub/sub system, and (3) 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 programatically 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, software components are called nodes, and they communicate with each other using a pub/sub system. Nodes register as publishers or subscribers to named communication channels called topics. Each topic has an associated message type that determines what data is sent over the channel. For example, ROS node that monitors the joint state 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 [27]. 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 publisher and subscriber are directly connected to each other1, publishers send messages by serializing a message-specific data structure into a sequence of bytes and sends the bytes over the connection. When the subscriber receives the sequence of bytes, it deserializes it to the message data structure and passes it to the program for processing.

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 in ROS2.

C. ROS Launch Scripts

Robot systems can be comprised of a complex graph of nodes communicating with each other via the pub/sub system. 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

```xml
<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 researchers to specify which nodes should be deployed in the cloud and which machine type should be used.

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

1 As an implementation optimization, ROS nodes on the same machine can communicate via a shared-memory queue, instead of using a network.
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 by finding and collecting all the packages in the script. The collected packages are ready to be pushed 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: FogROS Launch Script Example

```xml
<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>
```

Listing 2 provides an example of a FogROS launch script that serves the same functionality as Listing 1, while also running the server node to a cloud computer. Users can launch local ROS nodes, such as client as before. With FogROS, users provide the path to the launch file that contains the server node, server.launch and the type of cloud computer, c5.24xlarge. Users can optionally provide a script (here, init.bash) to install server dependencies and setup server environment variables.

B. Cloud-Computer ROS Nodes

When FogROS launches cloud-based nodes, it performs a 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. This process has the following steps:

1) Provision and start a cloud computer pre-loaded with ROS, and with the computing capabilities from the launch file.
2) Push core 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 pushes the ROS code to the cloud-computer securely over a secure shell (SSH) [37] connection. This process recursively copies the code, libraries, and dependencies from the edge computer to the cloud, and builds the code on the cloud. FogROS then optionally runs a user-specified script to install the library dependencies and to set up environment variables.

With the code ready to run, FogROS then starts the secure networking components for VPC (Sec. IV-C) or proxying (Sec. IV-D) (depending on configuration), and finally starts the code 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) modify the cloud computer setup 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 point-to-point connections that make up the VPC, while FogROS manages the edge-computer-to-cloud 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.

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 determining which option suits their application.

In FogROS, a proxy consists of two ROS nodes, one running in the edge computer and one running on the robot. 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 subscriber, 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 /roscloud/latency and /roscloud/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 Command Line Interfaces (CLIs), 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 supports launching containerized ROS nodes with a similar interface as the FogROS launch script extension described in Section IV-A. While an increasing number of ROS developers are using prebuilt docker images to host ROS nodes, this functionality is not natively supported by ROS. With FogROS, users can specify the image name and the destination machine on which they want to launch it. Given the image name and tag, FogROS uses a template environment setup scripts to pull and run it. 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. Given the name of the prebuilt docker image, and machine type, FogROS provisions and starts a cloud computer g4dn.xlarge with a GPU. FogROS then uploads and runs bash commands to pull dexnet:gpu from DockerHub [5]. Since the computer has a GPU, FogROS attaches the docker container to the GPU, and runs it.

\[
\text{Listing 3: FogROS Docker Container Example}
\]

\[
<\text{launch}>
\begin{align*}
\text{<node name="dexnet" pkg="fogros" type="}\hfill \\
\text{fogros_docker.py" output="screen">}
\text{<rosparam>}
\text{docker_image: kepler/dexnet:gpu}
\text{machine_type: g4dn.xlarge}
\text{</rosparam>}
\end{align*}
\end{\text{launch}>
\]

Listing 3 shows an example launch script for a Dex-Net grasp-planning node in a docker image. Given the name of the prebuilt docker image, and machine type, FogROS provisions and starts a cloud computer g4dn.xlarge with a GPU. FogROS then uploads and runs bash commands to pull dexnet:gpu from DockerHub [5]. Since the computer has a GPU, FogROS attaches the docker container to the GPU, and runs it.

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 the edge computer 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.
real-time setting. We also recorded the time taken for Bundle Adjustment (B.A.) in seconds) [24]. While cloud-based SLAM requires additional computer. We report frames-per-second (fps) and latency (in seconds) [24].

To configure FogROS to work with OpenVSLAM, we accumulate more data and produce higher quality maps in a faster and more frequent B.A. can mean lower error.

### A. Visual SLAM Service

OpenVSLAM [31] is a visual simultaneous localization and mapping system that uses monocular video input. In this experiment, a Camera Node publishes a 1920 x 960 resolution video to the cloud (Fig. 4a). On the cloud an OpenVSLAM node subscribes to the video feed, successfully adds the feed to the VSLAM database, and computes a pointcloud map along with the current estimated location within the map. The VSLAM node then publishes pointcloud and location updates back to the robot.

To configure FogROS with OpenVSLAM, we build ROS docker images and push them to Dockerhub [5]. We write the bash commands that pull and run the docker images as bash script. We include the path to the bash script as in Listing 2, and FogROS runs the script when configuring the environment. After initialization, FogROS automatically finds the topics on the robot and proxies the messages to the cloud SLAM server.

To evaluate the performance of FogROS when deploying OpenVSLAM, we compare the cloud-deployed performance to an edge-computer-only implementation. We select a 36-core cloud-computer for the OpenVSLAM node, and compare it with OpenVSLAM running on a one-core edge computer. We report frames-per-second (fps) and latency (in seconds) [24]. While cloud-based SLAM requires additional latency, it is able to achieve a higher FPS, meaning that it can aggregate more data and produce higher quality maps in a real-time setting. We also recorded the time taken for Bundle Adjustments (B.A.) as an important metric in measuring OpenVSLAM performance, as VSLAM uses B.A. to reduce accumulated tracking error in the map, and faster and more frequent B.A. can mean lower error.

### 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. We wrap an open-source implementation of the fully-convolutional grasp-quality convolutional neural network (FC-GQ-CNN) [22, 29], which plans grasps on rigid objects in industrial bins using an overhead depth camera, in a ROS node and deploy it to the cloud along with pretrained network weights as a Docker image.

This node subscribes to three input topics containing a scene depth image and mask for objects to be grasped, as well as a message of type sensor_msgs/CameraInfo containing camera intrinsics. Internally, the ROS node feeds this to FC-GQ-CNN, which outputs a grasp pose and associated q-value. These outputs are wrapped as gqcnn_ros/GQCNNGrasp message, containing the planned grasp’s pose, estimated reliability, gripper type, and coordinates in image space.

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: large 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 republish node.
from the image_transport ROS package [12]. 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 do this by randomly sampling configurations and connecting them together into a graph when motions are collision free. These planners can be scaled with additional computing cores.

Using FogROS, we deploy a multi-core sampling-based motion planner [7, 8] to a 96-core computer in the cloud to solve motion planning problems from the Open Motion Planning Library (OMPL) [30] (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.

To configure FogROS to work with multi-core motion planner, we record the steps we use to setup the dependencies 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 time, e.g., for simpler planning problems, there may be little to no benefit to a cloud deployment. If the motion planner is a multi-core asymptotically optimal motion planner (a planner that 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 shorter 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 applications to the cloud is small when compared to the time gained from using high-end computers with multiple cores and GPUs in the cloud.

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. For low-bandwidth connections, this compression could greatly reduce latency when using FogROS, especially for image sequence (video) topics.

REFERENCES


