An Industrial Robotics Application with Cloud Computing and High-Speed Networking

R. Rahimi*, C. Shao†, M. Veeraraghavan*, A. Fumagalli†,
J. Nicho‡, J. Meyer‡, S. Edwards‡, C. Flannigan‡ and P. Evans‡

*University of Virginia (UVA), Charlottesville, VA, USA
†The University of Texas at Dallas (UTD), Dallas, TX, USA
‡Southwest Research Institute (SwRI), San Antonio, TX, USA

***Abstract*—This paper describes an industrial cloud robotics distributed application that was executed across a high-speed wide-area network. The application was implemented using ROS libraries and packages. The purpose of the application is to enable an industrial robot to perform surface blending. A Kinect sensor, a surface blending tool and a laser scanner are mounted on the robot arm. The arm is moved under software control to scan a work bench on which metal parts of variable size can be laid out at any orientation. The collected point cloud data is processed by a segmentation algorithm to find the surface boundaries. A Cartesian path planning algorithm is executed to determine paths for the robot arm to execute the blending action and a laser scan on a selected surface. A new ROS package was implemented to collect CPU, memory and bandwidth usage for each significant ROS node in this distributed application. To emulate a scenario in which computing resources at a remote datacenter can be used for the segmentation and path planning algorithms in conjunction with the robots located on a factory floor, a software-defined network testbed called GENI was used to distribute compute-heavy ROS nodes. Measurements show that with TCP tuning, and high-speed end-to-end paths, the total execution time in the Cloud scenario can be reasonably close to a local scenario in which computing is collocated with the robot.**

Keywords: ROS; ROS-I; Industrial robotics; Cloud computing; High-speed networks

I. Introduction

Applications and platforms for the use of cloud computing for service robots have been proposed and developed [1], [2]. This paper addresses the use of cloud computing for industrial robots. With increased usage of sensors for robotic vision, high processing power is required to run 3D object reconstruction algorithms. Cloud computing offers a low-cost solution to meet this need. However, if cloud-computing datacenters are located at distant sites from factory floors where robots operate, high-speed wide-area networks (WAN) are required. This paper explores the impact of network latency on an industrial robotics application when executed in a distributed mode on computers located across the WAN.

A team of developers from SwRI, Boeing, Caterpillar, Wolf Robotics, and TU Delft implemented an application called *Godel* for automating the task of metal surface blending using an industrial robot ABB IRB 2400 [3]. Blending is the operation of smoothing metal surfaces down to an even finish. The robot is mounted on a work bench, and its arm, which is equipped with sensors and the metal surface blending disk tool, can be moved on the work bench under software control. The application uses Scan-N-Plan technologies, which are "a suite of tools that enable real-time robot trajectory planning from 3-D scan data" [4]. The application demonstrates how machine vision can be leveraged to offer flexibility in the size of metal surfaces and their placement on the robot's work bench.

The Godel application requires high processing power for two operations to: (i) identify the location and boundaries of the metal surfaces on the work bench, and (ii) plan a path for the robot arm to move in a manner that the surface blending tool can execute blending on the selected metal surface. Godel uses the Robot Operating Sytem (ROS) Industrial (ROS-I) [5], [6] software framework, which allows for distributed execution. However, Godel testing was mostly done in a configuration in which all the ROS nodes were executed in a single local computer collocated with the robot.

The *objective* of this work was to distribute the ROS nodes to several computers, some of which are located in a remote data center, and to characterize the performance of the application. To achieve this objective, minor modifications were necessary to the Godel code, and a new ROS package called `Resource Monitor` was implemented to measure CPU and memory usage, and sending and receiving rates for each of the main ROS nodes.

For our experiments, we used the NSF Global Environment for Network Innovations (GENI) testbed [7], which is a national federated testbed with 58 racks distributed throughout the USA and at some international locations. This testbed uses OpenFlow/Software Defined Network (SDN) innovations to allow for a flexible sharing of this testbed by over 9000 distributed systems/networks researchers. Users reserve "slices" that can consist of virtual machines interconnected by rate-specified VLANs. There are different types of GENI racks such as InstaGENI and ExoGENI.

We ran Godel in three different configurations: (i) across an InstaGENI-UVA testbed configured with a wide-area virtual private network (VPN), (ii) across a local-area UTD testbed, and (iii) across the InstaGENI-UVA testbed with an improved high-speed path across the WAN. In our two WAN distributed experiments on the InstaGENI-UVA testbed, the ROS nodes that performed surface detection and path planning were executed on InstaGENI hosts at U. Utah, while the ROS nodes directly associated with the robot that published sensor data and accepted actuator commands to move the robot arm were executed on a UVA computer.

Our key findings are as follows: (i) segmentation algorithms used for surface detection and the path planning algorithms benefit from using high-performance compute nodes, (ii) wide-area network throughput to transfer large files (such as point cloud data generated by sensors) is an important factor in cloud robotics. On the WAN VPN configuration, a Godel run took 323 sec, while on the local UTD configuration, the application needed only 72 sec. The transfer throughput for the point cloud data across the WAN VPN configuration was about 12 Mbps, but the throughput was 120 Mbps in the local UTD testbed. With modifications, we were able to increase the throughput in the WAN configuration to 120 Mbps.

Section II provides a brief review of related work. Section III describes the Godel application, while details of the software architecture are provided in Section IV. Section V describes our experiments in the three configurations to compare performance. The paper is concluded in Section VI.

II. Background and Related work

Robot Operating System (ROS) and ROS-Industrial are open-source software frameworks that allow robotics researchers to develop large-scale applications rapidly through component reuse. ROS allows applications to have multiple execution units named `nodes`. Nodes use the communication functions provided by ROS to send messages to each other. There are two possible methods for nodes to communicate: `service calls` and `topics`. Service calls use the client/server architecture suitable for point-to-point communications, while topics are better suited for broadcast messages from a single publisher node to multiple subscribers.

Besides ROS, other platforms for supporting cloud robotics include Robot as a Service (RaaS) [1], Rapyuta [8] and Robot Web Tools [9].

One of the main reasons for using cloud computing in robotics is to run applications with high processing requirements. Hu et al. [2] state that three robotic applications: Simultaneous Localization and Mapping (SLAM), Grasping, and Navigation (e.g., path planning), benefit from cloud computing. A team of mobile robots can collect information about the environment and send this data to a SLAM application, which can create a map that is then stored in a database and accessed by other robots. As most environments with mobile robots will not remain static, SLAM needs to be run often. Vision-based SLAM is both data intensive and compute intensive, and hence a Grid based FastSLAM was implemented in Hadoop, a commonly available platform in cloud computing datacenters [10]. The impact of the number of parallel nodes used on the execution time of FastSLAM was presented in this paper, but there was no discussion of network resources.

A cloud-based solution for grasping was implemented by Kehoe et al. [11] for a Willow Garage PR2 robot with onboard color and depth cameras. The solution used Google's proprietary object recognition engine, Point Cloud Library (PCL), and GraspIt! toolkit. Offline object recognition from 3D CAD models and offline grasp analysis, was combined with online image capture, object recognition, pose estimation and grasp selection. The authors reported a recall rate of 80% in object recognition, a failure rate under 14% for pose estimation, and a failure rate under 23% for grasp analysis. The work showed the advantage of using cloud resources to increase accuracy and decrease failure rate, but there was no discussion on the required compute and network resources.

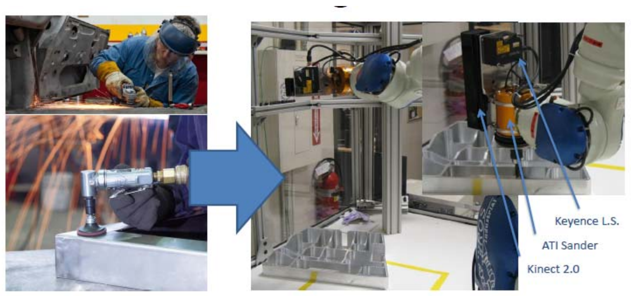


Figure 1: Industrial robotic surface blending [3]

The use of cloud computing for a real-time video tracking application was investigated by Liu et al. [12]. The performance of the application, in terms of frames per sec, was compared on four configurations: local 2-core machine, cloud 2-core VM, cloud 4-core VM, and cloud 6-core VM as a function of the number of simultaneous requests.

Complementing these above-described papers, our contribution to this burgeoning field of cloud robotics is to study the impact of network latencies on application performance.

III. An Industrial Robotics Application

Fig. 1 shows a surface blending disk in operation in the lower left image. The upper left image shows a worker doing manual blending. Workers who blend parts by hand suffer from repetitive stress injuries. The far right image shows the Scan-N-Plan robotic configuration used in the Godel application. Two scanners, Kinect 2.0 and Keyence Laser Scanner (L.S.), and one ATI Sander, are mounted on the arm of an ABB IRB 2400 industrial robot. The middle image shows the robot arm in a 90-degree rotated position.

The robotic blending *Godel* application works as follows. Parts that need to be blended can be laid out in arbitrary locations and orientations on the robotic work bench (cell). The Kinect 2.0 sensor is an RGB-depth camera, which is moved under the control of software, to scan the robot work cell. The software can be configured to have the Kinect sensor scan the work cell at specified positions by instructing the robot arm to move accordingly. The collected point cloud data is sent to the application software.

From the point cloud data, the boundaries of surfaces of metal parts to be blended are detected using a segmentation algorithm. A GUI presents icons representing the metal parts

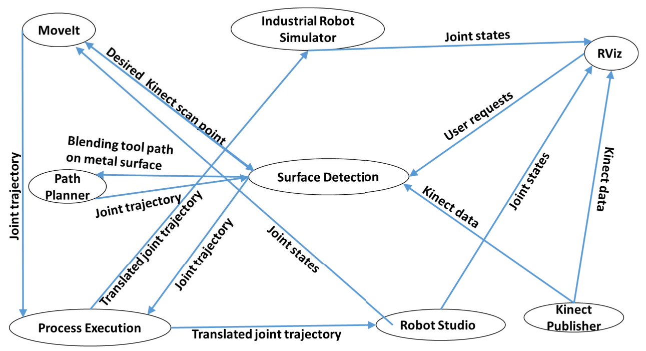


Figure 2: Godel ROS nodes with links representing communications between nodes

that are to be blended to a human operator, who then moves the cursor and clicks on the icon representing the specific part to blend next. Once the metal surface that needs to be blended is identified, a Path Planner computes a path for the blending tool using the Descartes planner (boost graph solver) [13]. The Path Planner finds valid and optimal solutions through a sequence of trajectory points, specific to the robot model. Optimum smooth trajectories that avoid collisions, while considering joint limits and singularities, are computed.

When the blending is completed, for Quality Assurance (QA), the software instructs the robot arm to be re-oriented for executing another path. This second path is computed for the Keyence laser scanner. The laser scanner image is rasterized for the human operator and a QA score is assigned by the software (the human operator can define parameter values for a high-quality blend).

The robot arm is instructed by the software to return to a safe state between the execution of these three paths (for the Kinect scan, surface blending and laser scan). When blending is completed successfully on a metal surface, the robot can be tasked by the software to blend another metal surface that is laid out on the robot work cell.

IV. Godel Software Architecture

Fig. 2 illustrates the various ROS nodes in the Godel application. In addition, `RosCore`, a basic ROS component used for communications and health monitoring of the ROS nodes, was run.

Industrial Robot Simulator: In the Godel application, it is important to check the computed paths before execution on the real industrial robot. One safe method is to test paths on a simulated robot. The `Industrial Robot Simulator` is used for this purpose. Fig. 2 shows that the `Industrial Robot Simulator` receives trajectories from the ROS node called `Process Execution`, and publishes joint states of the simulated robot to `RViz` to allow an operator to visualize the simulated robot arm movement.

MoveIt: `MoveIt` is a ROS package that performs collision-aware path planning for a robot [14]. `MoveIt` is used to move the robot arm to reach specified points. This ROS node is initialized with the parameters of the physical robot, e.g., robot arm size, joint types and joint limits. After the `Surface Detection` ROS node determines the locations on the robot work cell at which the Kinect scanner should capture images, the `Surface Detection` node sends the current state of the robot and the desired scan point to `MoveIt`. The `MoveIt` node sends the joint trajectory to the `Process Execution` node, which then generates and sends a translated joint trajectory to the `Industrial Robot Simulator` ROS node first, and then, if successful, sends the translated joint trajectory to the real robot or ABB RobotStudio as shown in Fig. 2. After sending the joint trajectory, `MoveIt` monitors the joint states of the robot, which allows it to know when the robot arm has reached the goal. At this point, `MoveIt` notifies `Surface Detection`, allowing the latter to obtain the Point Cloud data collected by the Kinect sensor at the specified scan point from the `Kinect Publisher`. When the data has been collected, `Surface Detection` sends a message to `MoveIt` to move the robot arm to the next scan point.

Path Planner: This ROS node receives a path along which the blending tool should smooth the metal surface from the `Surface Detection` node. The `Path Planner` then computes trajectories for the robot arm joints to make sure that the blending tool follows the path computed by `Surface Detection` to execute the surface blending. A ROS package called `Descartes` is used for the computation of joint trajectories. As noted in the `Descartes` Web site, “While `MoveIt` plans free space motion, i.e., move from A to B, `Descartes` plans robot joint motion for semi-constrained Cartesian paths, i.e., ones whose way-points may have less than a fully specified six Degree of Freedom (6DOF) pose.” The `Path Planner` sends the computed joint trajectory to `Surface Detection`, which saves this trajectory and waits to receive a user request from `RViz` to simulate the robot arm movements on the `Industrial Robot Simulator` or to execute the robot arm movements in the real robot or `ABB RobotStudio`. At this point, `Surface Detection` sends the joint trajectory to `Process Execution`.

Surface Detection: The `Surface Detection` ROS node is responsible for the operation of the whole system. It receives service calls and sends messages to other ROS nodes to plan and execute various components of the Godel application. The `Surface Detection` node determines the specific locations for Kinect scanning based on human-operator provided input (which is received from the `RViz` node), and uses the `MoveIt` node to physically move the robot arm to the scan points so that the Kinect sensor mounted on the arm can capture RGB-D images. After the `MoveIt` node has successfully caused the robot arm to move from its current location to the first scan point and the `Surface Detection` node receives notification of the successful motion, the `Surface Detection` node reads one frame from the `Kinect Publisher` node. The

`Surface Detection` node then transforms the received points (x, y, z) from the Kinect-sensor local frame to a global frame. The above operation is repeated for each scan point. Finally, the node runs an algorithm to remove outlier points, and to find a plane (surface) that fits the remaining points. The human operator is shown the detected surfaces through the `RViz` GUI using a different color from the rest of the work cell. The `Surface Detection` node also computes a path along which blending should occur on the metal surface.

The human operator selects one of these visualized icons representing the metal surfaces for the blending operation. Upon receiving a message from `RViz`, the `Surface Detection` node sends the blending tool path to use on the metal surface to be blended to the `Path Planner` node along with the state of the robot arm.

After the blending process, if the human operator requests a laser scan of the blended metal surface through the `RViz` GUI, the `Surface Detection` node receives a message from `RViz`, and then sends a message to the `Path Planner` node to compute joint trajectories for the robot arm to perform laser scanning.

Process Execution: This node receives joint trajectories computed by the `Path Planner` node, and codes these trajectories in the language of the robot controller. Specifically, the ABB robot controller uses the RAPID programming language [15]. The `Process Execution` node uploads the file containing instructions for joint movements to the robot controller (which directly controls the robot) via FTP. A simple RAPID program is run on the robot controller to execute the instructions in the newly uploaded file in order to move the robot arm.

ABB RobotStudio: This Windows software package is provided by the robot vendor ABB [16], and closely emulates the real industrial robot ABB IRB 2400. One important difference between the ABB RobotStudio software and the Industrial Robot Simulator is that the ABB RobotStudio runs the same code as the code executed on the real robot, while the Industrial Robot Simulator is a simulation of the robot. This makes the ABB RobotStudio highly valuable for experiments.

Kinect Publisher: This node publishes the Kinect sensor data with different resolutions using different topics (one topic for each resolution). A ROS node that subscribes to one of these topics will receive the Kinect data with corresponding resolution. In Godel, only the `Surface Detection` node requires this data. It is also possible to configure the `RViz` node to receive and visualize the Kinect data, if needed.

RViz: `RViz` is a ROS Visualizer package. In robotic application development, it is critical to have a visualization tool to monitor the robot state and sensor values. `RViz` accepts as input the robot physical dimensions and a CAD model, and displays a 3D graphical image of the real robot. `RViz` offers the human operator the ability to change the viewpoint in 3D space. `RViz` also accepts application-specific plugins to integrate with the basic visualization tool. These plugins can be used to receive input from the human operator for control actions. In Godel, a software plugin was written for `RViz` to allow the human operator to interact with the ABB IRB 2400 robot by selecting actions, such as Kinect scanning, metal surface selection, surface blending, and laser scanning. Fig. 3 illustrates the `RViz` GUI and the real ABB robot.

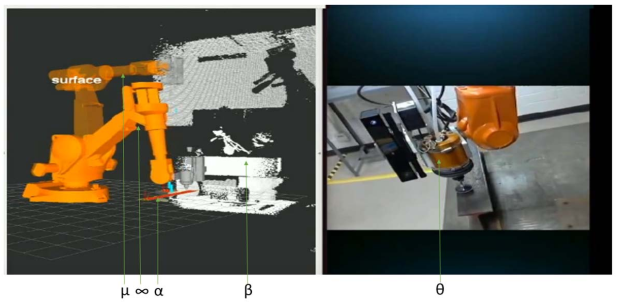


Figure 3: RViz display (left) and real ABB IRB 2400 robot at SwRI (right) [3]

V. Performance characterization

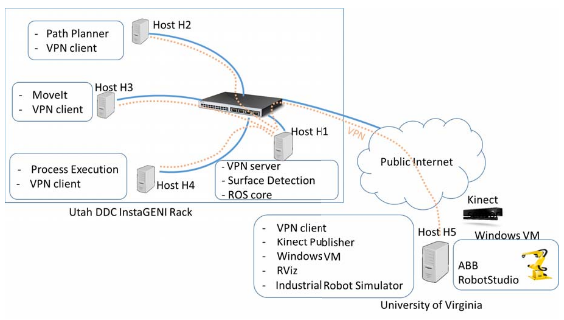
Section V-A describes our experimental testbeds. Section V-B describes the experimental steps. Section V-C presents and discusses the experimental results.

A. *Experimental setup: hardware and software*

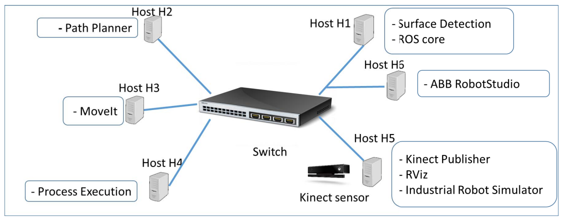
We used two experimental testbeds: (i) InstaGENI-UVA testbed: a wide-area network testbed, which includes hosts in an InstaGENI rack and a host at UVA, and (ii) UTD testbed: a local-area network testbed at UTD. For our experiments, we reserved a GENI slice consisting of four bare-metal (non-virtualized) hosts located in the U. Utah DDC InstaGENI rack.

Hardware: The InstaGENI-UVA testbed, shown in Fig. 4a, consists of 5 hosts, with H1 through H4 located at U. Utah, and H5 located at UVA. Each of the InstaGENI hosts has two 8-core CPUs (Intel Xeon CPU E5-2450) and 24 GB RAM, while the UVA host H5 has one 2-core CPU (Intel i5-3210m) and 12 GB RAM. Each InstaGENI host has four 1 Gbps Ethernet (1 GE) NICs, one of which is used as a *control-plane port* for remote user logins. The remaining three ports are referred to as *data-plane ports* as they are primarily used for experiments between two or more GENI hosts. Ubuntu 14.04 with kernel version 3.13 was run on all four GENI hosts. The UVA host has one 1 GE NIC, and runs Ubuntu 14.04 with kernel version 4.02. The Kinect sensor is attached to host H5.

The UTD testbed consists of 6 hosts, all of which are physically located in the same laboratory, and are all connected to the same Ethernet switch, as shown in Fig. 4b. Host H1 has a single six-core CPU (Intel Core i7-3930K@3.20GHz) and 32 GB RAM. Each of the hosts H2, H3, H4, and H5 has a single six-core CPU (Intel Core i7-4930K CPU@3.40GHz) and 32 GB RAM. H6 has a single four-core CPU (Intel Core i7-4770@3.4GHZ) and 8 GB RAM. The OS on hosts H1-H5 was Ubuntu 14.04 with kernel version 3.13, while host H6



(a) InstaGENI-UVA testbed



(b) UTD testbed

Figure 4: Experimental testbeds

ran 64-bit Windows 7 to support ABB RobotStudio. All host NICs and switch ports are 1 GE.

Godel software: Fig. 4a illustrates how the ROS nodes described in Section IV were distributed among the various hosts. The InstaGENI-UVA testbed emulated a scenario in which the computers used for the main Godel ROS nodes were located in a remote facility relative to the location of the industrial robot. UVA emulates the factory floor, and InstaGENI emulates a remote data center. Therefore the Kinect sensor is connected directly to the UVA host H5, and the ABB RobotStudio is run on a Windows VM on H5, while all the main Godel ROS nodes were run on InstaGENI hosts H1 to H4.

We had access to a real ABB IRB 2400 industrial robot located at SwRI. But it was unnecessary to schedule time to use this robot for testing and characterizing the performance of Godel software. The `ABB RobotStudio` was sufficient for this purpose. Therefore, we used the real robot only for demonstrations, such as the one we gave at the USIgnite 2016 Smart Community Week in Austin, TX, a video of which is available on our project Web site [17].

The `Kinect Publisher`, `RViz` and `Industrial Robot Simulator` ROS nodes were run on UVA host H5 as these nodes should be executed close to the sensors and robots. In our experiments, the human operator is assumed to be physically located on the factory floor next to the robots, and hence `RViz` was also executed at that site. Recall that `RViz` is used by the human operator to enter commands, and visualize computed paths for the robot arm using the `Industrial Robot Simulator` node before commands are sent to the real robot (or ABB RobotStudio).

The remaining ROS nodes, `Surface Detection`, `MoveIt`, `Path Planner`, and `Process Execution`, were each run on one of the four hosts H1 through H4 on the InstaGENI rack. Finally, we chose to run `RosCore` on host H1 along with `Surface Detection`, because `RosCore` does not require significant CPU capacity.

To quantify the effects of WAN vs. LAN communication, we used the same distribution of ROS nodes on the UTD testbed, with the exception that a separate host, Host H6, was used to run ABB RobotStudio.

Virtual Private Network (VPN): We used a VPN between the InstaGENI rack at U. Utah and host H5 at UVA because host H5 was using a private IP address and was located behind a NAT router, but the communication patterns are such that ROS nodes such as the `Surface Detection` node, which was executed on one of the InstaGENI hosts, initiates a TCP connection to the `Kinect Publisher` running on host H5. A VPN server was executed on InstaGENI U. Utah host H1, and a VPN client was run on UVA host H5. The VPN client first connected to the VPN server via the public IP address of the control-plane port of host H1. Next, the data-plane ports of hosts H1 through H4 on the InstaGENI rack were assigned private IP addresses on the same VPN. Thus communications between ROS nodes running on hosts H2, H3 and H4 to ROS nodes on H5 were routed via the VPN server on H1. Effectively a wide-area private-IP addressed VPN connected hosts H1-H4 at U. Utah to H5 at UVA. The round-trip time between hosts H5 and H1 is approximately 50 ms.

The performance of Godel suffered because of the processing delays incurred in this VPN as will be presented in Section V-C. To address this problem, we obtained a public IP address for host H5, and reran the experiments after tuning TCP-layer parameters for optimal performance on this wide-area 50-ms path. The performance improved significantly.

New resource monitoring ROS package: To measure Godel application resource usage statistics, we developed a ROS package named `Resource Monitor`, and ran the corresponding ROS node on all hosts. The `Resource Monitor` node accepts as input: (i) a list of names of ROS nodes to monitor, and (ii) a parameter specifying the frequency with which to collect and publish resource usage statistics. Specifically, the `Resource Monitor` collects CPU usage, memory consumption, and network bandwidth used by each process. For CPU and memory usage, the `Resource Monitor` uses `psutil` [18], which is executed in periodic intervals, where the period is specified by the input frequency parameter. While `psutil` does provide bandwidth usage, the information is reported on an aggregate basis, not on a per-process basis. Therefore, the `Resource Monitor` additionally invokes `nethogs` [19] to measure the network sending and receiving rates for each process (ROS node).

At the end of each period, the `Resource Monitor` node creates a message that includes CPU, memory, sending and receiving rates for each process. These messages are sent out in a ROS topic called `chatter`. A ROS utility tool, `rostopic echo`, was configured to subscribe to the `chatter` topic, and save the received messages to a file.

We developed a Python parser to extract measurements from the chatter messages, and used R to make graphs.

Modification of Godel application XML launch files: The ROS framework supports distributed applications, however, Godel was implemented to run all ROS nodes in one host. But the "machine" tag feature available for use in `roslaunch` XML files made it relatively easy to run Godel on a distributed set of hosts. The main launch file for Godel was modified to add a list of machine names with corresponding IP addresses. Also, our login credentials for each host (machine) was included in the main launch file. Next, we manually searched the tree of launch files called from the main launch file hierarchically, and identified all the launch files in which ROS nodes were initiated. In each of these launch files, we added the name of the machine on which the ROS node invoked by that launch file should be executed.

B. *Experimental execution*

It is simple to initiate the Godel application even in the distributed mode described in Section V-A. Only a single (main) launch file needs to be started. This launch file was executed on host H5 in both testbeds. The main launch file calls other launch files, which in turn instantiate ROS nodes on the hosts identified with the corresponding machine tags.

Once all the ROS nodes are running, the application execution starts when the experimenter (emulating the human operator on the factory floor) manually pushes a `SCAN` button on the `RViz` GUI asking the system to initiate the Kinect scanning of the robot work cell. An `RViz` parameter was set by the operator to execute scans at three positions of the work cell. The various ROS nodes, `RViz`, `Surface Detection`, `MoveIt`, and `Kinect Publisher`, execute their functions and communicate as needed, and effectively accomplish the task of detecting surface boundaries of the metal parts to be blended, and rendering these surfaces on the `RViz` GUI. At this point, the experimenter selects one of the rendered surfaces using the host-H5 mouse, and pushes the `GENERATE` button on the `RViz` GUI. This step leads to `RViz` and `Surface Detection` executing their corresponding tasks, and communicating with the `Path Planner` and `Process Execution` ROS nodes. When the path computation is completed, the `RViz` GUI enables the `SIMULATE` button, allowing the experimenter to push this button for the next step. In this step, commands are sent by the `Process Execution` ROS node to the `Industrial Robot Simulator`, which then offers the experimenter a visual rendering of the simulated path movement of the robot arm to accomplish surface blending. If the simulated robot arm movement meets the approval of the human operator (experimenter), the next button, `EXECUTE`, will be pushed causing the `Process Execution` ROS node to send commands to the `ABB RobotStudio`. There is no emulator for the laser scanner described in Section III, and hence this device was not used in our experiments.

Upon conclusion of the application execution, the file collected by the ROS utility tool `rostopic echo` was parsed and measurements for CPU and memory usage, sending and receiving rates for the main ROS nodes were computed for the duration of the experiment. These results are presented next.

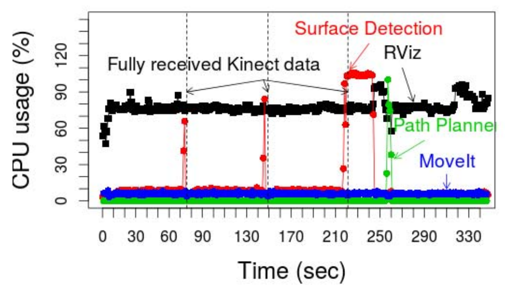
C. *Experimental results*

Fig. 5 shows our results for three different experiments with the Godel application: (i) across InstaGENI-UVA testbed configured with a wide-area VPN; (ii) across UTD testbed; and (iii) across InstaGENI-UVA testbed with an improved network path.

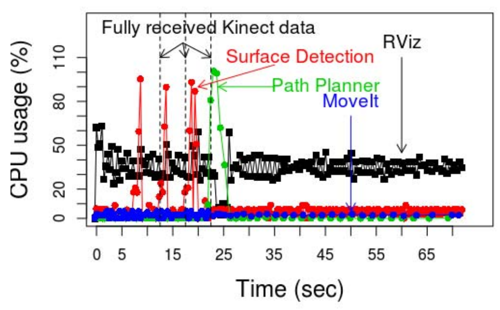
Figs. 5a and 5b show CPU usage for different ROS nodes across the InstaGENI-UVA testbed and UTD testbed, respectively, and Figs. 5c and 5d show the rate at which each ROS node receives data through one run of the distributed Godel application. While we also collected memory usage and network-traffic sending rates for each ROS node, these are not presented in this paper due to a lack of space. The sending rates are small, i.e., on the order of 100 kbps, and the memory usage is highest for the `RViz` node, and second highest for `Surface Detection`, reaching 300 MB. This memory usage is largely due to the Kinect sensor point cloud data from the three scans.

First, we study Figs. 5a and 5c to understand the CPU resource and network rate requirements of the four Godel ROS nodes: (i) `Surface Detection`, (ii) `RViz`, (iii) `Path Planner`, and (iv) `MoveIt`. The `Surface Detection` and `RViz` ROS nodes are the big consumers of CPU time. The three spikes in CPU usage of `Surface Detection` correspond to reception of the point cloud data generated by the three scans of the Kinect sensor. After the reception of the last scanned data, starting from 221 s, the `Surface Detection` node executed for 23 secs to process the image data from the three scans, and to identify the metal surfaces that were laid out on the robot's work cell. After the surfaces were detected, and the experimenter pushed the `GENERATE` button, the `Path Planner` consumed more CPU resources, reaching up to 100% for about five secs, to compute the path for the surface blending tool. The `RViz` node then consumed about 80% of a CPU core while rendering the computed path on the `Industrial Robot Simulator` for the operator to visually check for correctness.

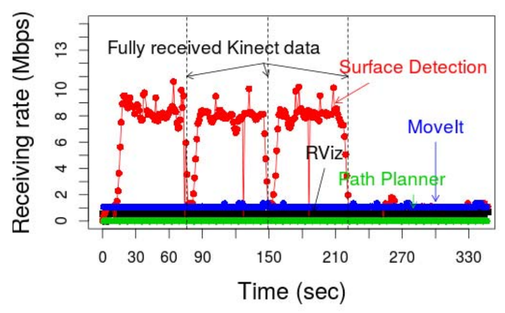
Second, we observe an alignment between data reception rates of `Surface Detection` in Fig. 5c and CPU usage of this ROS node in Fig. 5a. The Kinect sensor collected data using FullHD resolution (1920×1080). Each scanned data file has 66 MB of data. The scanned data is sent by the `Kinect Publisher` to `Surface Detection`. During reception of each of these three data files (one from each scan point), CPU



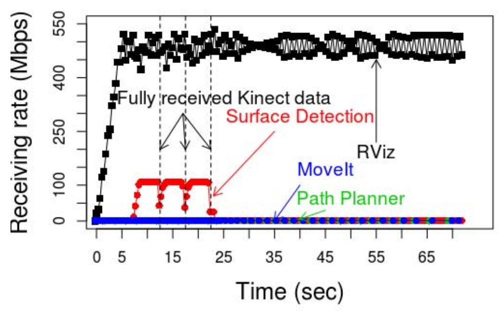
(a) InstaGENI-UVA; CPU usage; VPN



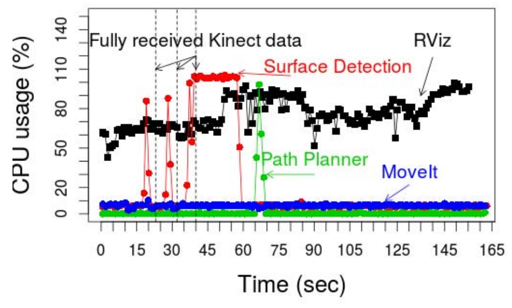
(b) UTD; CPU usage



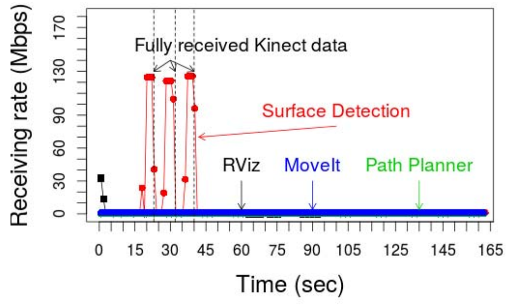
(c) InstaGENI-UVA; Receiving rate; VPN



(d) UTD; Receiving rate



(e) InstaGENI-UVA; CPU usage; Improved network path



(f) InstaGENI-UVA; Receiving rate; Improved network path

Figure 5: Experimental results

usage by `Surface Detection` spiked, and in some cases, this ROS node used more than 1 CPU core.

Third, we compare the data reception rate in the InstaGENI-UVA testbed and the UTD testbed (see Figs. 5c and 5d). The data reception rate is an order of magnitude higher on the UTD testbed. The `Surface Detection` node received data at approximately 120 Mbps in the UTD testbed, but at about 12 Mbps on the InstaGENI-UVA testbed. Correspondingly, the completion time for one run of Godel was 341 sec on the WAN testbed, but only 72 sec on the local testbed.

Fourth, a deliberate difference was introduced in the experiment execution on these two testbeds. Specifically, `RViz` was configured to receive the sensor data transmitted by `Kinect Publisher`, while this option was disabled in the WAN experiment. To enable network traffic measurement between `RViz` and the `Kinect Publisher`, we moved `RViz` to host H2 in UTD. Fig. 5d shows `RViz` receiving data at a high rate of 500 Mbps throughout the experiment (since `Kinect Publisher` keeps sending sensed data).

Finally, consider Figs. 5e and 5f. These graphs show the significant improvement that was possible when the VPN was removed. With a public IP address assigned to UVA host H5, we re-ran the Godel application on the InstaGENI-UVA testbed. In this run, importantly, we changed the default settings of TCP-layer parameters such as sending and receiving buffer sizes to values larger than the Bandwidth-Delay Product (BDP), and used HTCP [20], which offers quicker recovery from packet losses. Fig. 5f shows that the receiving rate of `Surface Detection` increased to 120 Mbps, which is the same as the receiving rate on the UTD testbed.

However, the total completion time on the WAN testbed was still 163 s. The difference with the 72-s completion time achieved on the UTD testbed is attributed to the following: (i) live sensing with the Kinect sensor was used, not stored ROS bag files; (ii) there were four manual inputs; and (iii) the `ABB RobotStudio` was executed in a separate Windows host H6 at UTD, but on a VM in the same host H5, which also ran the `RViz`, `Kinect Publisher`, and `Industrial Robot Simulator` on a Linux VM, at UVA. The difference in `ABB RobotStudio` execution speeds between UTD and UVA was significant. Even though the point cloud data size was the same (66 MB) and network rate was the same (120 Mbps), it took 40 sec to complete this phase of the application in UVA but only 22 sec in UTD. Similarly, the blending movement took 60 s vs. 38.5 s. While the physical environment for Kinect sensing was set up to be as similar as possible at UVA and UTD, and the point cloud data size was the same, the processing times by `Surface Detection` were significantly different (22 s in InstaGENI-UVa testbed vs. 1.5 s in UTD testbed). In other words, after improving the WAN path, the impact of the differences between the CPU power of the hosts at UVA and UTD became apparent.

VI. Summary and Conclusions

This paper demonstrated the feasibility of a distributed execution of an industrial robotics surface blending application. High CPU usage was observed for two ROS nodes of this application: `Surface Detection` and `RViz`. While the latter needs to be located with the operator, who typically is at the same location as the robot (though this could change), the former was successfully executed on a remote cloud computer. Also `Path Planner`, which computes robot arm trajectories, uses significant CPU power. In addition, an important finding was that a high-speed WAN connection, with TCP tuning, is required between the robot location and the datacenter.

VII. Acknowledgment

This work was supported by UVA NSF grants ACI-1340910, CNS-1405171, and CNS-1531065, and UTD NSF grant CNS-1531039.

References

[1] Y. Chen, Z. Du, and M. Garca-Acosta, "Robot as a service in cloud computing," in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, June 2010, pp. 151–158.

[2] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE Network*, vol. 26, no. 3, pp. 21–28, May 2012.

[3] Scan-N-Plan for Robotic Blending Milestone 3. http://rosindustrial.org/news/2015/8/19/scan-n-plan-for-robotic-blending-milestone-3.

[4] Scan-N-Plan^{TM}. http://rosindustrial.org/scan-n-plan/.

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[6] ROS-Industrial. [Online]. Available: http://rosindustrial.org/

[7] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014.

[8] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, April 2015.

[9] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, "Robot web tools: Efficient messaging for cloud robotics," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, Sept 2015, pp. 4530–4537.

[10] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots," in *Robotics and Automation (ICRA), 2010 IEEE Intl Conf. on*, May 2010, pp. 3084–3089.

[11] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 4263–4270.

[12] B. Liu, Y. Chen, E. Blasch, K. Pham, D. Shen, and G. Chen, "A holistic cloud-enabled robotics system for real-time video tracking application," in *Future Information Technology*. Springer, 2014, pp. 455–468.

[13] ROS-I Descartes Planner. [Online]. Available: https://github.com/ros-industrial-consortium/descartes

[14] Create a MoveIt Package for an Industrial Robot. [Online]. Available: http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot

[15] Rapid reference manual. http://rab.ict.pwr.wroc.pl/irb1400/datasys_rev1.pdf.

[16] RobotStudio. [Online]. Available: http://new.abb.com/products/robotics/robotstudio

[17] Godel Demonstration at the USIgnite 2016 Smart Community Week, Austin, TX, June 13-15. https://pages.shanti.virginia.edu/HSN/software/robotic/.

[18] psutil: a cross-platform library for retrieving information on running processes. [Online]. Available: https://github.com/giampaolo/psutil

[19] nethogs. [Online]. Available: https://github.com/raboof/nethogs

[20] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Protocols for Fast Long Distance Networks Workshop (PFLDnet)*, Feb. 16-17, 2004.