Demo: Managing Sensing Resources at the Edge using Cloud OSes

Lirim Osmani,* Ashwin Rao,* Samu Varjonen,* Eemil Lagerspetz,* Hannu Flinck,† and Sasu Tarkoma*
*University of Helsinki, †Nokia Bell Labs

I. Introduction

Environmental sensing is an important use case for edge computing and mobile networks. Specifically, edge computing approaches and mobile networks are expected to be used by sensing applications to collect data from fixed environment monitoring stations along with the sensors mounted on mobile sensing platforms. These mobile sensing platforms are in turn expected to leverage micro-servers such as Raspberry Pis for collecting the data, performing some initial computation, and disseminating their results for further processing.

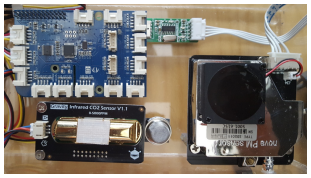
A mobile sensing platform is also envisioned to host a wide range of applications, each requiring access to a subset of available sensors. For instance, sensors deployed on buses may be used for monitoring the air quality, the temperature, and the sound levels [1]. Consequently, there is a need to support multi-tenancy and support applications with different life-cycles. We believe that Cloud OSes such as OpenStack[1] can be used to meet these requirements.

We posit that Cloud OSes can be extended to manage the sensing resources along with the computing resources at the network edge. In this demonstration we show that OpenStack can be used to manage containers on Rasperry Pis. This setup enables a) micro-servers at the network edge to support multi-tenancy like their counterparts in data centers, and b) Cloud OSes to manage the resources at the network edge along with the resources in data centers. Specifically, such a *hybrid cloud* setup can be useful in augmenting the computational resources in private and public clouds with their counterparts in the network edge.

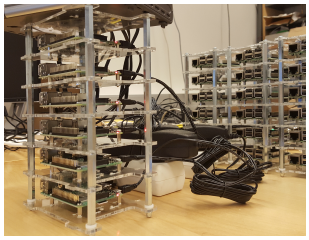
Along with the setup with OpenStack, we also have a prototype setup of a mobile sensing platform in which air quality sensors are connected to a Raspberry Pi. For instance, we have a) the Grove optical dust sensor,[2] b) the Sainsmart MQ131[3], and c) DF Robotics SEN0159[4] connected to a Raspberry Pi using GPIO. An example of the setup is shown in Figure 1(a). We plan to integrate these sensors with the current setup of the Pi cluster, as discussed in section III.

II. System Description

As shown in Figure 2(a), our setup consists of i) a high-end server cluster which contains five Dell PowerEdge M610 servers with two quad core Intel Xeon E5540 processors, and ii) a Raspberry Pi cluster with three Raspberry Pi 3B model. We use Ubuntu 16.04 as the base OS in these clusters.



(a) Our mobile sensing platform.



(b) Rack of Raspberry Pis.

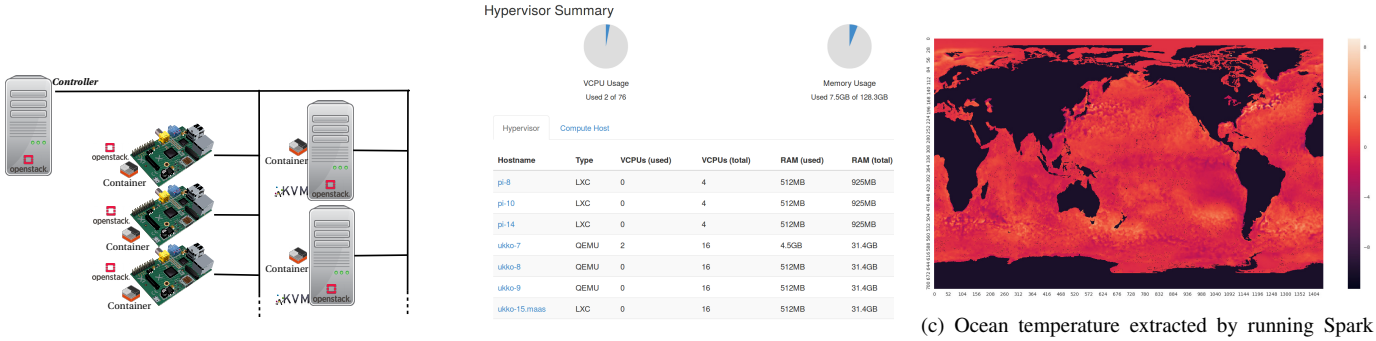
Fig. 1. **Devices used by our system.** *Our mobile sensing platform consists of air quality sensors connected to a Raspberry Pi. We also have a cluster of Raspberry Pis which are managed using OpenStack.*

We use OpenStack Newton release to manage this cluster. Specifically, we configure one of the high-end servers as an OpenStack controller with the Neutron services running on a Raspberry Pi unit. All the other nodes are configured as OpenStack compute nodes.

The high-end servers used as compute nodes were configured to use either KVM to spawn VMs, or LXC to spawn containers. In contrast, the Raspberry Pis currently have limited or no support for instantiating VMs spawned using KVM and QEMU. We therefore use the Raspberry Pis to spawn containers. Specifically, we configure OpenStack to use the nova-compute-lxc agent for managing containers spawned using LXC. An OpenStack compute node uses two interfaces, one for communicating with the controller, and the other for providing network connectivity for the VMs. The Raspberry Pi 3B has an embedded 100 Mbit Ethernet interface, and a Wi-Fi interface. However, OpenStack is unable to use the Wi-Fi interface. One reason for this behavior is that OpenStack is designed to manage servers in data centers who primarily use wired technologies such as Ethernet. We therefore add a USB Ethernet adapter on each Raspberry Pi.

In Figure 2(b), we present an example screenshot from the OpenStack admin console. We can see that the controller was able to include the Raspberry Pis in our cluster as compute nodes. In Figure 2(c), we present the results obtained by running Apache Spark in our containers to extract the average ocean temperature at specific locations from the NOAA data.[5] We run Apache Spark inside containers spawned on Raspberry

[1]https://www.openstack.org/
[2]http://wiki.seeedstudio.com/Grove-Dust_Sensor/
[3]https://www.sainsmart.com/products/mq-131-gas-sensor-ozone-module
[4]https://www.dfrobot.com/product-1023.html
[5]https://www.esrl.noaa.gov/psd/cgibin/db_search/DBListFiles.pl?did=132&tid=67170&vid=2421



(a) Network Topology.

(b) Screenshot of OpenStack console.

(c) Ocean temperature extracted by running Spark on our cluster.

Fig. 2. **System overview.** *We create an OpenStack Cloud with our cluster of Raspberry Pis and some high-end servers. We then launch a job which requests resources from our cloud for running Apache Spark.*

TABLE I
RESPONSE TIMES FOR LAUNCHING AND DELETING VIRTUAL INSTANCES

	Response Time (seconds)								
	KVM (Server)			LXC (Server)			LXC (Pi)		
Action	min	max	avg	min	max	avg	min	max	avg
nova.boot-server	29.53	61.66	39.39	16.77	23.41	19.02	19.71	21.76	20.45
nova.delete-server	5.06	5.69	5.28	12.29	17.19	14.94	7.94	10.75	9.89

Pi to exemplify that the containers running in the Raspberry Pis can be used to perform numerical analysis of data.

In Table I we compare the time required to spawn a VM using KVM on our high-end server, a container on our high-end server using LXC, and a container on a Raspberry Pi using LXC. We used an image of Ubuntu 16.04 for this evaluation, and the values presented are the minimum, maximum, and average values obtained across 10 iterations. The *nova.boot-server* action spawns the VM/container, while the *nova.delete-server* action deletes the VM/container. We observe that time required to spawn and delete containers on a Raspberry Pi is larger than the time required to perform those actions on a high-end server. However, it is still less than the time required to spawn and delete VMs instantiated using KVM on high-end servers. These preliminary results are promising and motivate us to continue building this platform.

Specifically, we plan to manage our sensing platform using this setup. In the following, we present our research agenda based on this setup.

III. RESEARCH AGENDA

Initially we plan to use and extend our current setup with additional Raspberry Pi units, and then address the following open questions.

A. Managing sensing resources in Cloud OSes

Currently the resources made available to containers are limited to CPU, memory, disk space, and networking interfaces. We plan to extend the list of resources with the list of sensors which are to be made available to the containers. This requires a) the OpenStack modules running on the micro-servers to be able to identify the set of sensors connected, export this set to the OpenStack controller, and instantiate containers with a subset of these sensors, and b) the OpenStack controller to be able to compose containers with a subset of these sensors. Furthermore, there may be instances where multiple containers would like to have access to a given sensor. In this context, we plan to extend OpenStack using the insights of the Android architecture which exposes sensors on mobile devices to applications.[6] For instance, OpenStack can either a) add a sensor into the container's name space and make it available solely to a container, or b) create virtual copies of the sensors distribute these virtual copies to the containers, or c) create a proxy for the sensor and the allows containers to request data from these proxies. We plan to explore the viability of each of these approaches.

B. Networking

The micro-servers are expected to use a range of communication technologies to communicate with each other and the OpenStack controller. For instance, micro-servers running on the same mobile station may either use Wi-Fi or Ethernet to communicate with each other, and use cellular technologies to communicate with the OpenStack controller. We have an LTE testbed configured in our lab, and we plan to experiment with running the cloud control plane over the cellular fabric.

C. Resource Management in Multi-clouds

In our environment we are already using heterogeneous hardware and we have configured the OpenStack scheduler to recognize the different CPU architectures accordingly when spawning instances. Our next step is to include resources from public clouds (Amazon EC2, Microsoft Azure, and Google Cloud Platform) and create a multi-cloud solution. In this respect we already have started experimenting with the libcloud API[7] as library for interacting with many of the popular cloud service providers using a unified API.

REFERENCES

[1] N. Castell, F. R. Dauge, P. Schneider, M. Vogt, U. Lerner, B. Fishbain, D. Broday, and A. Bartonova, "Can commercial low-cost sensor platforms contribute to air quality monitoring and exposure estimates?" *Environment international*, vol. 99, pp. 293–302, 2017.

[6] https://source.android.com/devices/sensors/sensor-stack

[7] https://libcloud.apache.org/