I. Artifact description

We present the reproducibility artifact of our experimental validation. The source code is available in our GitHub repository[1].

A. Overview

- Matching game framework. See Section I-B;
- Dataset. See Section I-C;
- Software dependencies. See Section I-D;
- Run-time environment: `Visual Studio Code IDE 1.65.2`;
- Installation. See Section I-E;
- Testbed. See Section I-F.

B. Matching game framework

Matching game is a Python-based framework[2] for solving matching game based models. We used YAML files containing device and microservice preference lists (DPL.yaml and MPL.yaml), and the schedules of microservices on devices (matching-testbed.yaml).

C. Data sets

Two datasets for training neural network and machine learning models:

- Case study I: German traffic sign benchmark[3];
- Case study II: Amazon reviews for sentiment analysis[4].

D. Software dependencies

1) Model:

- `Python libraries` networkx, operator, numpy, yaml, json.

2) Integrating C^3-Match to `Kubernetes` scheduler:

- `Docker v20.10.12`[5] on all devices and instances;
- Orchestrating by `Kubernetes v1.21`;
- Measuring bandwidth and latency between the local Edge devices: `kube-latency`[6];
- Monitoring local Edge cluster by the `Prometheus operator v0.45.0`[7];
- `Python script "scheduling.py"` collects the monitoring information by `Prometheus Python API`[8] from the local `Kubernetes` Edge cluster, and then utilizes the `Python client library v17.17 for Kubernetes`[9] to execute the customized C^3-Match scheduler and deploy the application pods on the appropriate devices;
- Transmitting data through asynchronous message queue platforms: 1) a Kubernetes-based `KubeMQ v2.2.10`[10] for the local Edge cluster, and 2) `ZeroMQ v22.3.0`[11], to receive data by the Cloud and Fog instances.
- `Ping` and `iPerf3` tools to benchmark the latency, maximum achievable bandwidth, and effective downlink throughput between instances.

3) Case studies:

We Dockerized all the microservices with `Docker v19-20`[12].

a) Python libraries: ffmpeg, scikit-learn, numpy, tensorflow, keras, matplotlib, opencv-python, pandas.

b) Tensorflow API: Docker image `nvcr.io/nvidia/l4t-tensorflow r32.5.0-tf2.3-py3`[13] on Nvidia devices.

E. Installation

- Matching library by `package installer for Python 3.9` (`pip3.9`);
- Docker Engine on Ubuntu[14], Kubernetes on Ubuntu[15], and Kubernetes on (vanilla) Raspbian Lite[16];
- NVIDIA Jetson Linux[17] and Raspberry Pi OS[18].

F. Testbed

- Three on-demand instances from the Exoscale provider[19], hosted in the data center of the $A1$ network operator in Sofia, Frankfurt, and Vienna: `large` with four virtual cores and 8 GB of memory; `medium` with two virtual cores and 4 GB of memory; `small` with two virtual cores and 2 GB of memory.
- A private Cloud at the university campus and `medium` Edge instances, managed by an `OpenStack v13.0` and `Ceph v12.2` with support for block and S3-compatible object storage;
- Five NVIDIA Jetson Nano (NJN) running `Linux for Tegra (L4T)` OS, 10 Raspberry Pi-3 model B+ (RPi3) and 30 RPi4 running `Raspberry Pi OS`.
- Testbed components are interconnected by a managed layer-3 HP Aruba with 48 1 Gbit/s ports, 3.8 µs latency and aggregate throughput of 104 Gbit/s.

[1] https://github.com/anonymousuni/c3-match

[2] https://github.com/daffidwilde/matching

[3] https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[4] https://www.kaggle.com/bittlingmayer/amazonreviews

[5] https://www.docker.com/

[6] https://github.com/simonswine/kube-latency

[7] https://github.com/prometheus-operator/prometheus-operator

[8] https://pypi.org/project/prometheus-api-client/

[9] https://github.com/kubernetes-client/python

[10] https://github.com/kubemq-io/kubemq-community/releases/tag/v2.2.10

[11] https://pypi.org/project/pyzmq/

[12] https://www.docker.com/

[13] https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-tensorflow

[14] https://docs.docker.com/engine/install/ubuntu/

[15] https://phoenixnap.com/kb/install-kubernetes-on-ubuntu

[16] https://github.com/alexellis/k8s-on-raspbian/blob/master/GUIDE.md

[17] https://developer.nvidia.com/embedded/linux-tegra

[18] https://www.raspberrypi.com/software/

[19] https://www.exoscale.com/compute/