Title

Computation Offloading for Smart Touristic Sites - COSMOS

Organization

Institute of Communication and Computer Systems – ICCS

Experiment Description

The COSMOS project develops a framework which enables the dynamic offloading of processing workloads from mobile devices to edge clouds to facilitate the deployment of smart touristic applications in crowded cultural areas, by capitalizing on the features provided by the 5GINFIRE functionalities.
COSMOS Objectives

- Dynamic computation offloading mechanism for the minimization of the energy consumption of the portable devices.

- Workload profiling for the computation of resource requirements of the VNF chain towards efficient VNF placement and scaling.

- User mobility estimation by using measurement of motion sensors. The estimated position is taken into account for the offloading decision.

- Performance evaluation (*i.e.,* measurement of provisioning and scaling timescales, micro-benchmarks to identify potential performance bottlenecks) with the VNFs (*e.g.,* object recognition) that are deployed in COSMOS.

The performance of the proposed framework is evaluated with the below experiment scenario:

The scenario addressed in this work capitalizes on the University of Bristol 5G Testbed MEC infrastructure within the Millennium Square in the center of Bristol, and users moving in the vicinity of the Square. Visitors of this crowded place will use Raspberry Pi's, equipped with cameras to take snapshots of a PoI to get useful sight information via an object recognition service. Google's TensorFlow deep learning framework, an image classification service where the "Inception v3" Deep Neural Network was selected for the specific use case and was retrained in order to classify images.
IoT devices are generally limited in terms of energy availability and processing power. Hence, to enhance energy saving according to MEC principles for IoT-enabled applications, mobile users connect via multiple Wireless Access Points located in the Millennium Square and offload their computing intensive requests to a cluster of Edge servers. This placement enables low-latency access to the servers, capable of serving the users' requests in an on-demand fashion as depicted in Figure 1.

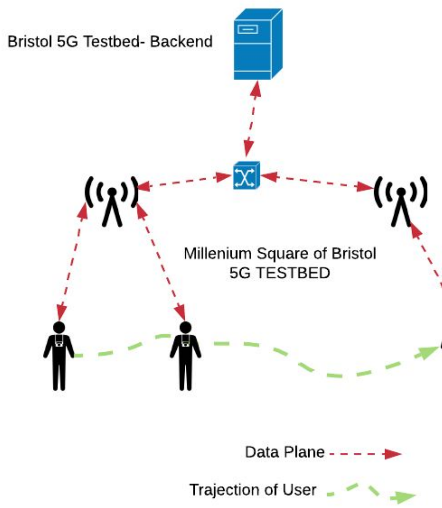

Figure 1 COSMOS use case overview

Cosmos Architecture

An overview on the system architecture is shown in Figure 2. The Bristol University testbed provides OSM Release 4 operating as an NFV management and orchestration tool that is connected with OpenStack, which acts as Virtual Infrastructure Manager (VIM) that controls the Virtual Deployment Units (VDUs). The system architecture of COSMOS follows a top-down design, meaning that there exists a VDU, namely; the Centralized Controller that dictates the decisions needed for the load balancing of the incoming workload, while at the bottom layer 3 VDU's namely; TensorFlow VDUs with different flavors, are deployed by the Bristol's OpenStack node. The proposed architecture is generally applicable in single-site MEC infrastructures and can be easily expanded towards Edge-to-Cloud or Edge- to-Edge collaboration.

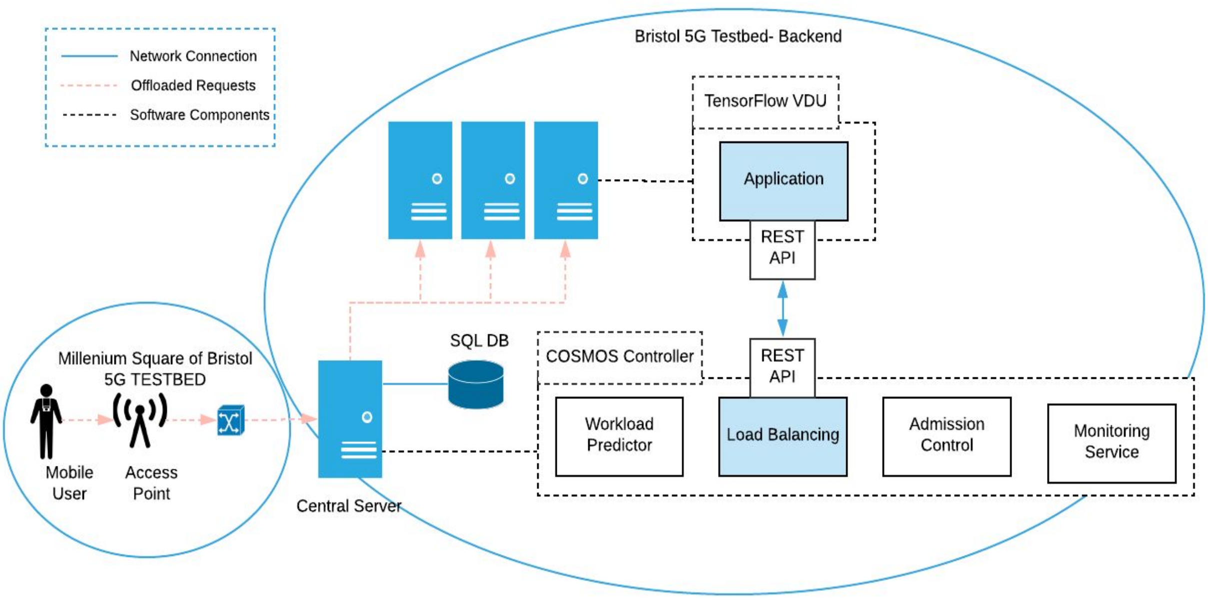

Figure 2 COSMOS architecture

VNF Description

The COSMOS experiment uses 4 VNFs:

1. **COSMOS Controller VNF**

This VNF enables the functionalities below:

- **Dynamic offloading mechanism** that is based on contextual information and aims to satisfy both the QoS requirements of the users and the infrastructure provider.
- An accurate **workload prediction methodology** which is necessary, in order to enable optimal load balancing and resource allocation.
- **A Load Balancing mechanism** for an intelligent dynamic optimal resource allocation.

2. **Three TensorFlow VNFs**

The TensorFlow-based object recognition services are deployed as a VNF, communicating with the COSMOS Controller via a REST-API present at each of the application VDU's and exploiting the internal connection elaborated in the Network Service Descriptor. The flavors for the three different TensorFlow VNFs are respectively:

- Flavor 1: 2 CPU cores, 2 GB of RAM and 10 GB Storage (Small)
- Flavor 2: 4 CPU cores, 4 GB of RAM and 20 GB Storage (Medium)
- Flavor 3: 8 CPU cores, 8 GB of RAM and 20 GB Storage (Big)

NS Description

Figure 3 represents COSMOS Network Service graph as it is depicted in the OSM dashboard. Each VNF interacts with two different planes, the management and the data one. The communication between them is conducted through a dedicated network interface assigned to that specific plane. This allows the latter to have its own separate network, which is solely used for the plane's purposes.

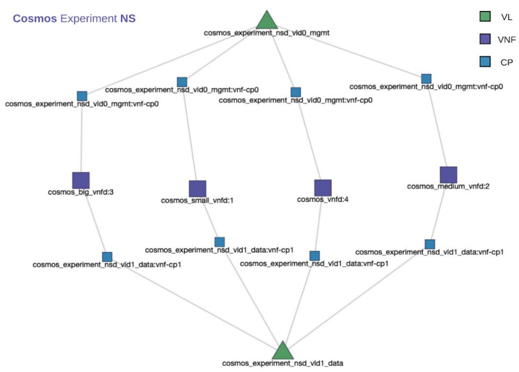

Figure 3 – COSMOS Network Service graph

Configuration for Experimentation

Experiment Deployment through 5GinFIRE portal. The network Service of figure 3 is instantiated. The VNF and NS Descriptors can be found here:
https://github.com/jdimol/cosmos_experiment

The COSMOS Controller's functionalities can be found here:
https://github.com/Dspatharakis/cosmos_project

The Object-recognition VNF functionalities can be found here:
https://github.com/ksgourom/cosmos_project/tree/master/cosmos_object_detection_api

To access the VNFs after the Deployment

Configuration to use TensorFlow VNFs:

```
Login to tensorflow vdu's
Username: cosmos
Pass: cosmos2019


Check the mtu (using ip link command) and change it to the appropriate: "ip link set dev ens3 mtu 1500"

Execution:
Run the following command to start the flask server that serves the application: nohup flask run
--host='XX.XX.XX.XX' --port=5000 --with-threads &
the output of the flask application is saved in nohup.txt and host = ip address of vdu

Your application server is ready to server offloaded requests for image recognition using tensorflow!
```

Configuration to use COSMOS controller VNF:

```
Your application server is ready to server offloaded requests for image recognition using tensorflow!


Login to COSMOS controller vdu's
Username: cosmos
Pass: cosmos2019

Check the mtu (using ip link command) and change it to the appropriate: "ip link set dev ens3 mtu 1500"


Run the following commands:
cd cosmos_project/app
vim config.py

Ensure that the ips of the config file indicate the ips of the tesnorflow vdu's. The first one is for the small
flavour, the second one for medium and the third one for big.

Run the following command:
cd ..

Activate virtual environment
source venv/bin/activate
```

```
export  FLASK_APP=microblog.py
flask run --host='XX.XX.XX.XX' --port=8000 --with-threads
where XX.XX.XX.XX = ip of COSMOS controller vdu

Run the following commands for database initiation:
flask db init
flask db migrate -m "table"
flask db upgrade

And then use curl command for initialization of the db

curl -X POST http://XX.XX.XX.XX/db
where XX.XX.XX.XX = ip of COSMOS controller vdu

Initiate redis with celery and run a celery worker:
redis-3.2.1/src/redis-server
./venv/bin/celery worker  -A app.celery --loglevel=INFO
./venv/bin/celery beat -A app.celery --schedule=/tmp/celerybeat-schedu --loglevel=INFO --pidfile=/tmp/celerybeat.pid

Now we are ready to run the flask server that serves the COSMOS Controller:
flask run --host='XX.XX.XX.XX' --port=8000 --with-threads

Now you are ready to run an example of offloading requests using
python3 user.py
```