Distributed Operating System

HW.2: Microwebservices

Student Name: 1. Asem Jamil Ishtayah

2. Hussien Abu Amer

Supervisor: Dr. Samer Arandi

Introduction:

On this part (2) we will learn and dealing in real world with replication and consistency, and this part will build on previous part, that we built on it a small micro-web-services project. And we will deal with a docker container (O.S level virtualization) and put all requirements for each server on a single image of docker to make the running of any server from anywhere more simple.

Requirements:

1. VMware: to operate gusts operating systems that will operate servers separately.
2. Gusts OS: ubuntu-18.04.3.
3. Micro-web framework: We use FLASK for Python language.
4. Database: We use SQLite3.
5. Text editor: Visual Studio Code.
6. Github account: https://github.com/AsemIsht/DOS_HW-ASEM
7. Docker hub account: https://hub.docker.com/r/asemisht/flaskapp
8. Previous part of project 😁.

Design:

1. In-memory cache:
   We use in-memory cache, to save the lookup requests. And to reuse them without recalling the category server.
2. Cache replacement policy: we use LRU (least recently used) to store new requests if the cache is full.
3. Load balancing algorithm, to distribute the requests on the category and order replicas.
4. Cache consistency - server push techniques: when write occurs on the DB, we must invalidate the corresponding data (with id) on cache.
5. Replicas sync: when write occurs on any replica, must sync occurs on others.

How to run the program:

I will describe how to run any server on any machine with ubuntu-18.04.3 OS.

Each server has source code file that has extend of (.py) indicate to Python text. This file includes all of necessary code for the micro-webservices related to the specific server.

* Open Terminal, and insert:

```
$ python3 -V            ➔ output: Python 3.6.6 # Python3 Version
```

install a virtual environment: insert:

```
$ sudo apt install python3-venv
```

Change the directory to work directory (server directory where python code exists) by insert `cd` command.

run the following command to create your new virtual environment:

```
$ python3 -m venv venv → #not needed because the venv directory is existed.
```

The command above creates a directory called `venv`, which contains a copy of the Python binary

To start using this virtual environment, you need to activate it by running the `activiate` script:

```
$ source venv/bin/activate
```

Installing Flask:

```
(venv) $ sudo apt install python-pip ➔ # at first install pip
```

```
(venv) $ pip install Flask
```

```
(venv) $ pip sudo apt install python3-flask
```

Installing requests library:

```
(venv) $ pip install requests
```

Run the hello.py code:

```
(venv) $ export FLASK_APP=hello.py
```

```
(venv) $ export FLASK_ENV=development     # mode as debugger (developing)
```

```
(venv) $ flask run –host=0.0.0.0    # run the server on the network to be
                                    accessed from any machine on the network
```

To be more honest, what I wrote before is sufficient to run the project, but I want to note that I found the best way to run the project is to delete the `venv` directory (delete manually), and reinstall it again on your machine. (yellow line on the page above).

Output generated:

The same of previous part, but what will be noticed is the speed of the requests.

The average response time:

Avg. response time with cache:

I make 10 times several requests and calculate the avg.

```
0.000004529953002929687 5
0.000005960464477539062 5
0.000004768371582031 25
0.000005960464477539062 5
0.000005245208740234375
0.000009059906005859375
0.000009298324584960938
0.000008821487426757812
0.000006675720214843 75
0.000005006790161132 8125

average: 0.000006532669067382 81 second !
```

Note: These values when all requests (cache hit)!

Avg. response time without cache (previous part):

```
0.0101604461669921888
0.00495338439941406255
0.00653338432312011710
0.00735688209533691410
0.00652337074279785165
0.00578594207763671910
0.00541710853576660165
0.00726890563964843755
0.00788235664367675810
0.00607824325561523410


average: 0.006796002388000485 second !
```

We can now imagine how important cache is

Overhead of cache consistency operations:

Cache consistency will occur each write on any replica. (consistency with replicas and with in-memory cache): it will send put requests to (1. Replica 2. In-memory)

The sequence will be:
Frontend → Order → Catalog → In-memory consistency → replica consistency

But on previous it was only:
Frontend → Order → Catalog

If we record time on frontend, we will approx. be feeling that the double time need!

With replication consistency time need:

```
0.03938698768615723
0.0290830135345459
0.03517723083496094
0.03637886047363281
0.030886173248291016
0.05513739585876465
0.024342060089111328
0.024616479873657227
0.028575897216796875
0.03619122505187988

average: 0.0339775323867798 second !
```

Without replication consistency time need:

```
0.028792858123779297
0.018616914749145508
0.014951944351196289
0.014739513397216797
0.01744222640991211
0.021662235260009766
0.02418994903564453
0.0224263668060302 73
0.018498659133911133
0.014993906021118164


average: 0.019631457328796 3 second !
```

Latency of a subsequent request that sees a cache miss:

```
0.009247064590454102
0.005703926086425781
0.007290124893188476 6
0.005868673324584961
0.006117582321166992
0.00583195686340332
0.005526304244995117
0.006717681884765625
0.006094694137573242
0.00516963005065918

average: 0.006356763839721 68 second !
```

Summary table:

Lookup with cache	Lookup without cache	Buy with replication	Buy without replication	Lookup cache miss
4.52995E-06	0.010160446	0.039386988	0.028792858	0.009247065
5.96046E-06	0.004953384	0.029083014	0.018616915	0.005703926
4.76837E-06	0.006533384	0.035177231	0.014951944	0.007290125
5.96046E-06	0.007356882	0.03637886	0.014739513	0.005868673
5.24521E-06	0.006523371	0.030886173	0.017442226	0.006117582
9.05991E-06	0.005785942	0.055137396	0.021662235	0.005831957
9.29832E-06	0.005417109	0.02434206	0.024189949	0.005526304
8.82149E-06	0.007268906	0.02461648	0.022426367	0.006717682
6.67572E-06	0.007882357	0.028575897	0.018498659	0.006094694
5.00679E-06	0.006078243	0.036191225	0.014993906	0.00516963
6.53267E-06	0.006796002	0.033977532	0.019631457	0.006356764

We calculate finally:

Lookup (with_cache/without_cache)*100% = 0.096%

Buy (without_replication /with_replication)*100% = 57.78%

And we note that Lookup cahce miss and Lookup without cache approx. the same!

All comments that needed was written on the code. And I I tried to make everything clear.

Thank you 😍