ECSE429 Project Part C:

Non-Functional Testing of Rest API Report

Group 25

260722818 - Ege Odaci

Table of Contents

1) Summary of Deliverables

Our team is split up due to previous issues in the team. We were suggested to do the project seperately. I was assigned the CPU use.

For Part C of my ECSE429 Automation Project, I continued testing '**api todo list manager'** created by Alan Richardson. I have created scripts that does many post, delete and modify request to the todos, projects and categories of the given api. My task was to measure the CPU use during those api calls and present it as a graph against time as well as a number of instances

1.1) Activity Monitor

Activity Monitor

Mac equivalent of Task Manager
Displays resources in use in real-time
Can be used to track the cpu usage

My first approach was to use the activity monitor to track the cpu as i do many such as 10.000 post, delete, modift requests to the given api. Activity monitor shows you every process and how much cpu they are using. Also shows at the bottom that how many of those resources are used by the system and how many are by the user. Has a nice illustration of CPU Load graph.

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	% GPU	GPU Time	PID	Us
Python	70.2	18.77	2	2	0.0	0.00	77781	odaci
java	32.2	19.33	34	978	0.0	0.00	77779	odaci
kernel_task	12.6	10:43:51.09	212	405	0.0	0.00	0	root
mDNSResponder	12.5	7:49.05	5	957	0.0	0.00	248	_mdnsr
Activity Monitor	8.2	18:14.25	5	2	0.0	0.00	59386	odaci
configd	3.7	6:31.78	9	0	0.0	0.00	154	root
WindowServer	3.5	13:03:56.23	11	8	0.0	7:26:39.54	245	_windo
PyCharm	3.0	1:56.72	47	451	0.0	5.64	77465	odaci
symptomsd	2.4	4:47.04	2	0	0.0	0.00	247	_networ
hidd	1.6	2:14:49.82	5	0	0.0	0.00	204	_hidd
karabiner_observer	1.2	38:59.08	13	1	0.0	0.00	150	root
launchservicesd	0.8	13:43.51	4	0	0.0	0.00	182	root
sysmond	0.5	17:59.64	3	1	0.0	0.00	342	root
logd	0.5	4:35.25	3	0	0.0	0.00	161	root
Google Chrome	0.1	1:59.60	28	2	0.0	0.00	76370	odaci
karabiner_grabber	0.1	40:43.25	18	4	0.0	0.00	152	root
Dock	0.1	6:49.73	4	0	0.0	0.05	437	odaci
Google Chrome Helper (Renderer)	0.0	3.20	16	1	0.0	0.00	77663	odaci
Google Chrome Helper	0.0	34.22	9	4	0.0	0.00	76382	odaci
AdobeCRDaemon	0.0	4:35.98	4	2	0.0	0.00	3461	odaci
karabiner_console_user_server	0.0	1:07.64	12	3	0.0	0.00	58909	odaci
fseventsd	0.0	8:45.55	14	2	0.0	0.00	146	root

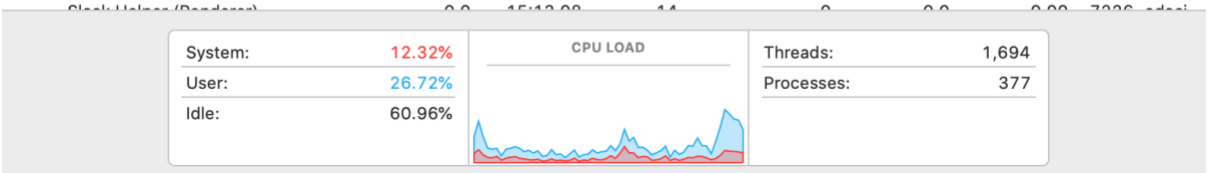

Figure-1 Activity Monitor

When i ran the requests, i observed the increase in the CPU load as it can be seen in the Figure-1. However when it comes to collecting this data, Mac didn't really have the tools.

One method to collect them was to use the code below.

```
top -l 10 -ocpu -R -F -s 1 >top.samples.txt
```

What this snipped does is that, it collects the data in the activity monitor every second for 10 seconds and saves it as text file. It provides text file but it has way too much information about every single process that is running on the system. It would require too much cleaning to be able to see the data we want. Thus i needed another way to handle this issue

1.2) Using Psutils

There is library called psutils that lets you track the system usage of your computer in the phyton code. I decided this can be better approach than the Activity monitor. I gathered cpu usage information during the code to be able to make the required graphs. More details can be found in the Structore of the Story Test Suite

```
a)psutil.cpu_percent(interval=1)

b)psutil.cpu_percent()
```

Above code snippit

- A) Returns the cpu usage percentage every 1 second.
- B) Returns the cpu usage since it is called last time.

2) Structure of Story Test Suite

This section explains the structure of our Story Test Suite in more detail.

Device Details

MacBook Pro (Retina, 13-inch, Early 2015)

Processor 2.7 GHz Dual-Core Intel Core i5

Memory 8 GB 1867 MHz DDR3

2.1) Performance Tests CPU vs Time

As previously mentioned, I used psutils to keep track of the cpu status. I have done 10.000 post, delete and modify requests to todos, projects and categories seperately. And tried to keep track of the cpu change.

At first as i do the requests, i tracked the time and measured the CPU usage change every 1 second for 20 seconds. Then stored that information as an array, at the end of execution made a seperate graph for each different type a request. Graph for post requests on todos can be seen in Figure-2 below.

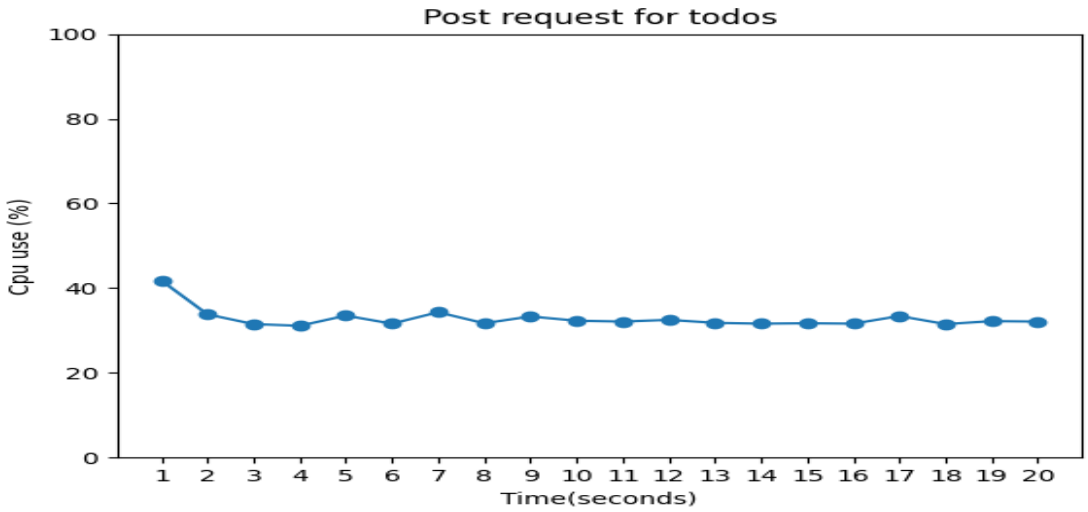

Figure-2

The rest of the graphs can be found in Graphs/TIME_CPU and can also be produced by running the code described in Section 3 of this report. In the above graph we see how many percentage the CPU is used by all the sources in my system. To be able to compare how much cpu is used when computer is idle, i got the same graph while not running the requests.

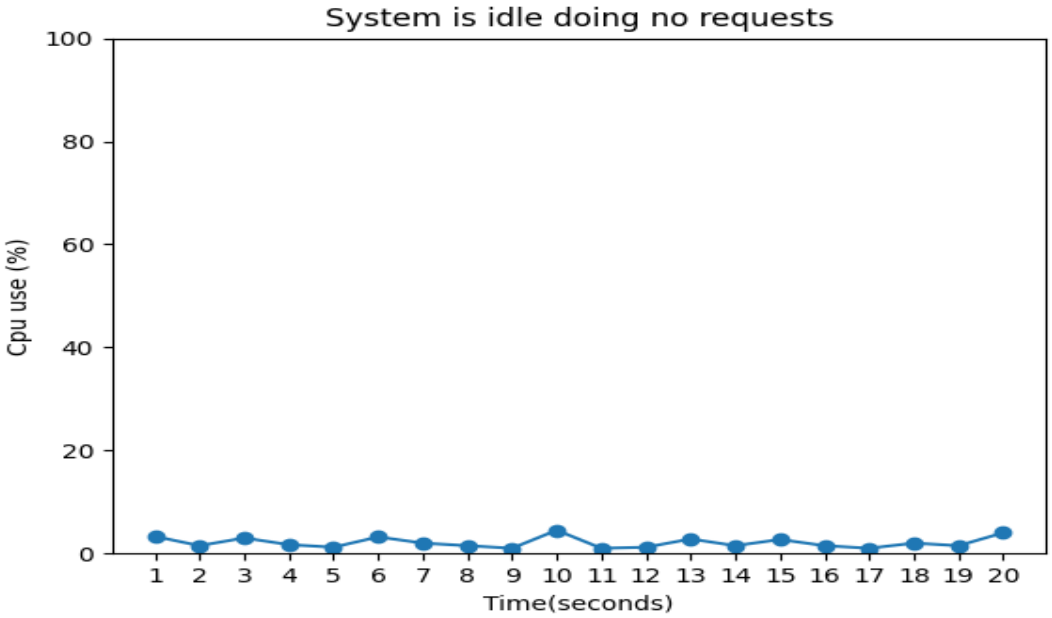

Figure-3

By comparing Figure-2 and Figure-3 we can see that post todo requests are consuming around 40% of the cpu. It is a normal amount since we are doing requests over and over for 10000 data entry into the api.

2.2) Performance Tests CPU vs Instances

For the second part, I had to test the Cpu against the number of instances. Since cpu usage can not be measured accurately every time an instance created, because it can't be measured at a unit time thus it should be measured in a period of time. Every instance is

created so fast thus it would be misguiding. My approach was to measure the cpu usage when every 500 request is created and store them as an array. At the end, using that array graph of cpu usage vs instances are created.

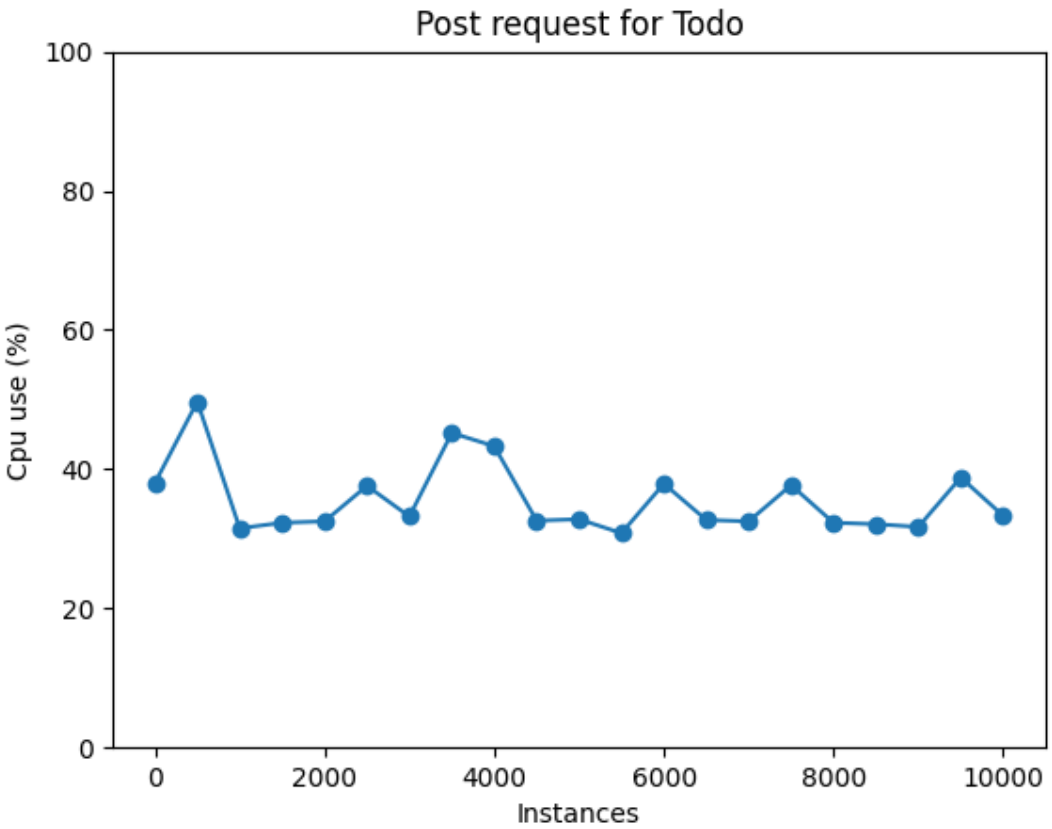

Figure-4

Cpu usage for post request on todo can be seen in the Figure-4. Rest of the Cpu Use vs Instances graphs can be found in Graphs/Instances_CPU folder and also can be created by running the program. We again see in Figure-4 that it is consuming respectable amount of cpu for every instance created.

3) Source Code Repository

Our source code repository consists of 4 phyton files.

- collect_cpu_data.py
  - It does post requests on the system for 10.000 times, to be run along with graph_time_cpu.py
- graph_time_cpu.py
  - It measures the cpu usage against time on the system. To be run together with collect_cpu_data.py in order to measure cpu usage as post requests are done. Then it plots Cpu Usage vs Time graph
- graph_cpu_instance.py

    - It measures the cpu usage against number of instances. It takes a cpu usage data for every 500 instance and plots the graph Cpu Usage vs Time.
- test_performance.py
    - It is a pytest test tool, it tests the api requests done on the given api.

Our repository also includes:

- ReadMe file (***ReadMe.txt***) that explains how to run the tests and program.
- Graphs folder that contains graphs produced by our program.
    - Instances_CPU
        - Has graphs for Cpu Usage vs Instances for all post requests.
    - Time_CPU
        - Has graphs for Cpu Usage vs Time for all post requests

4) Recommendations Improving CPU Test

At this part of the report, i will summarize the challenges faced and possible improvements for the better accuracy of measuring cpu usage.

- As i mentioned that i tried to get data from activity manager, i couldn't get what i tried to get. It maybe improved writing scripts that cleans text file leaving only the usage of Python thread and somehow converting it to .csv file. By doing the same for computer time in that text file then theoretically we can get cpu usage for phyton vs time graph.
- Tried to run collect_cpu_data.py and graph_time_cpu as two seperate process using multiprocess library in phyton but due to measuring cpu every 1 second, they didn't run at the same time. It can probably be fixed by more code implementation. I fixed the solution by running them as seperate process through different terminals.
- Finally, graphs can be improved to get much detailed information.