API Rate Limit

Version 1.0

Project Team :

- **Kush Singh**
- **Lokesh Chhabra**
- **Gangesh Dhar**

Table of Contents

API Rate Limiter

DOCUMENTATION

What is an API Rate Limiter ?

In our understanding API rate limiter is a program which runs on the server and executes before every API request to the server. The user requests to any of available API's and this program checks if the current request is within the permissible limits or not for the user. If yes then the server allows the request otherwise blocks the request.

Approach/Overview:

We are using socket programming for communication between server and client. We run each incoming client on a separate thread. This makes it possible to have multiple clients all in at once .

We have defined a Bucket class which has the core functionality of the rate limiter. The class is initialised with default values of click_limit and refresh _time. The bucket class contains different functions which performs different functionality of rate limiter. The reduce() function is the one which contains the major logic for validation of the user request for an API against its permitted values and return **TRUE** if valid or **FALSE** if invalid.

In our implementation of rate limiter, the limits for each user corresponding to an API are set by default to (10, 60) which corresponds to **click_limits** and **refresh_time** (in seconds) respectively. **click_limit** is the maximum number of requests that are allowed within refresh_ time. After the **refresh_time** runs out click_limit is again set to max. We have also added the functionality of changing these limits during the execution of the program.

The server is running and open for connections, can handle multiple clients. A client gets connected and sends its user_id. The server on receiving the id responds in a predefined way. If it is a normal user the execution proceeds. The user requests for an API, the server checks if the request is valid for the user and proceeds according to the results. If it's a super user , halts any further new connections as well as any API request by already connected user, this happens as limit details are being changed by the super user.

Data Storage Format

We have used a **file** to store dictionary (hash map) variable containing information about rate limit of user+API . The names, click limits and time left data are saved in this file in the case when the server stopped, when this happens the data stored in the file. When the server restarts, all the previous data is retrieved from this file.

On the server we have used a dictionary to store the data. Dictionary is an implementation of unordered hash map on python offering O(1) look up time for its keys . The key of this ***usrLog*** dictionary is the ***user_id***, which is unique. The value stored against this key will be a list containing 5 ***Bucket*** objects from ***ratelimObject***.py and an integer at the last position.

The integer will represent if user (represented by the key) is logged in or not. 0 representing not logged in and 1 representing logged in. the Objects will be the objects of ***Bucket*** class and the number of objects in a list will be equal to the number of API's present. Where each object represents (user,API) combination rate limit. Whenever a request is received from the user the reduce() function of the corresponding object is called to verify the request.

To store the state of server we could have also used a database but we didn't because that would have been slower and we not using any query so there was no need of a database. Also the size of the data to be stored is not going to be huge.

Redis was an alternative which could have been explored for storing user details instead of dictionary but the scope of this project didn't require an a distributed in memory database.

rateLimObject.py

This contains the main functionality of our program.

A ***class*** is named ***Bucket*** is defined which on initialisation takes two parameters as input (via constructor).

- The ***max_clicks*** representing maximum request count
- ***refresh_time*** representing the time span/window for which rate limit is maintained, and after that everything resets.

Those two input values in the constructor serve as the default limit values for a ***(user, API)*** combination. Each new user when requests for an API, the default values will be set against its entry. In this implementation the limit is set on ***(user,API)*** combination. So every unique combination can have different rate set against it.

This ***Bucket*** contains a function ***reduce()*** which is called every time a user requests for an API. What this function does is that is checks the called object for its validation. Meaning when a user requests an API this function checks that the request that user made was valid or not according to his rate limits. This functions returns or ***TRUE*** or ***FALSE***.

Returns false when ***max_clicks*** value reaches zero before ***refresh_time*** window. Otherwise return true for every case.

Server.py

This file acts as a server in our program which represents an actual web server. We have user socket programming approach to show client server relation. A server socket is binded to the port 5555 (used by us). This port will behave as a server port for incoming connections.The server is capable of handling multiple clients at the same time, this we have achieved using threading. Multi-threading is used to create client server connection on different threads. Once server is running, it is open for connection and is n listening mode. Each new connection a server makes a new thread is generated to handle that connection.

For each connection following things will occur after successful connection.

1. Client sending its ***user_id.*** Users are of two types, ***superuser*** and ***normal user.***

2. Before executing anything , server checks if superuser is logged in or not. If yes then the connection is refused otherwise the user is logged in.

3. The server then checks if the current user is already logged in or not. If current user is already present then again disconnect. Otherwise proceed.

4. The user then sends a request for an API. If the user is an old user then a call is made to the ***reduce()*** function of the ***Bucket*** class for validation. If true is return then the request is granted otherwise rejected.

5. If it was a new user coming for very first time then, the limits of this user for using the API's is first initialised to default and then the ***reduce()*** function is called, and the execution proceeds like in the other case.

What is 'superuser' ?

'superuser' is a special user which we have created so that we can change rate limits of any user and its corresponding rate limit without making changes in the source code. Whenever a super user logs in, all the subsequent login request are rejected and the users which were already in established connection with the user cannot make any new API request until and unless the super user leaves. But every already online request will be completed.

Other Files

1. client.py:

➔ This file contains the logic of client connecting and communicating with the server. It is used to represent user in our program.

2. DummyAPI1.py
3. DummyAPI2.py
4. DummyAPI3.py
5. API4.py
6. API5.py

➔ Files 2-4 make calls to a dummy API requesting random names and places while 5-6 calls an actual working API. The main functions of these files is to make the representation of an API better and to make the project function as a project.

Assumptions

1. User_id is unique, ie, no 2 users will have same id.
2. Only 5 API are considered. No provision for increasing this count without changing the source code.
3. No security checks are present.
4. Not all use cases have been covered in this.
5. Server is impersonated by a socket using socket programming.
6. Clients are impersonated by the connections made to the server.
7. User_id and API request is send to the server as string from the client. URL is not used.
8. No API has been created. Only a few dummy API's have been used along with some API request to an actual API to represent API's.

Running the Program

Tested on Ubuntu 16.04

- Open first terminal at the location where project files are located.
- Run the ***server.py*** file using command **python2** ***Server.py*** in the terminal located. Terminal now acts as output screen for our server.
- Now run the ***client.py*** file using command **python2** ***client.py*** in the terminal window. This represents one user.
- Open 3rd terminal and run same command in previous point to open 2nd client. Open as many clients in different terminal windows as you wish.
- On server window, for each successfully established connection there will be an output in the server terminal, displaying the IP and the port no. of the connected client.
- On the client terminal , we will enter a ***user_id.*** This can be any string .

If **user_id** is not '***superuser' :***

- On receiving the ***user_id***, server terminal window displays the ***user_id***.
- The client terminal window welcomes the user and asks for API ID (between 1 and 5 in our case).
- Number 1-3 correspond to dummy API which request for random number and names, while 4-5 are APIs for weather and currency exchange. We will look to set limits at these APIs
- The user enters the API request and it is sent to the server.
- The server checks the request and determines if request for user + API is valid or not. If its valid gives the API access to the user otherwise a message is displayed on the client terminal: ***"limit exceeded!!!!"***
- If API access is given to the user, at the client terminal the ***clicks_left*** and time till next ***refresh_time*** is displayed. After which an input is required by the API, to which a response is displayed at the client terminal.

  If client is a ***superuser:***

- On receiving the ***user_id***, server terminal window displays the ***user_id***.
- The client terminal window welcomes the ***superuser*** and asks "Do you wish to change rate limit(y/n) : "

If ***y*** is entered:

- The ***superuser*** has to enter the name of the ***user_id*** whose limits he has to change, and it is send to the server.
- The server checks the ***user_id*** whether it is valid or not. If its valid then ***"valid name entered*** " is displayed and superuser has to enter API id.
- After that enter the new limit values of ***click_limit*** and ***refresh_limit***.
- After that again the message is displayed stating "Do you wish to change rate limit(y/n) : ".

If ***n*** is entered

- Connection is terminated.

Added functionality for key storage

- In case of a situation where you have to switch off server, the keys will be saved in a file ***key_dict*** in same directory.
- Before closing server, make sure all other client programs are closed either manually or forcefully.
- To close server press ***CTRL + C*** on server terminal.
- All keys are saved and will be loaded next time server starts.