Introduction

The problem in this case can be divided into two different phases. First the program needs to figure out and learn the maximum number of requests allowed and the length of the time frame when these requests can be made. The rate limit could range from e.g. 10 requests in 60 seconds to 100 requests in 600 seconds. While these above-mentioned rate limits support the same frequency for both limits (1 request/6 second), the 100/600 rate limit enables to make all the 100 requests immediately when the 10/60 allows to make maximum of 10 requests immediately.

The second phase for the program is to actually limit the number of outgoing requests according to the rate limit learned earlier. The program will allow the user to make simple API calls, but it automatically limits the requests if the learned API rate limit is reached.

Learning Algorithm

The first thing to figure out is the maximum number of requests that can be made in a narrow space of time. This will establish the baseline for the maximum allowed requests during an unknown period of time. This can be done by rapidly making queries for the server and checking the server output for any signs of rate limiting on the server side. The behaviour in this case can be either server not responding at all to inbound requests, server responding slowly to requests, or server responding with HTTP status code 429. Before the first request, a time stamp needs to be saved so we can use it as a reference point later to figure out the length of the rate limit time frame.

After making all the allowed requests and figuring out the maximum number of requests, the time frame needs to be determined. This can be done by utilizing exponential backoff algorithm to make API requests at a decreasing rate or using a constant rate to possibly get more accurate results. At some point the server will respond again normally and by comparing the time stamp of the first request and the time of the first successful request after waiting, the time frame length can be determined. Exponential backoff algorithm limits the number of unnecessary API requests and is the basic method of trying to resume communications if the server fails to respond. The maximum wait time per request when using exponential backoff is 63 seconds in this program.

Rate Limiter

The actual rate limiter algorithm utilizes an idea of a token bucket. The bucket is a counter variable and it holds tokens which represent the number of requests that can be sent at any given time. If the bucket only has one token, then only one request can be made until another token frees up and increases the bucket counter by one. Individual tokens return to the bucket after the rate limit time frame length has passed since the time the request was originally sent.

Running the program

The program is written in Python3, so a suitable interpreter is required, and program uses *requests* and *time* python libraries to provide functionality for making simple HTTP request and measuring time.

Execution of the program starts by entering the desired web API address and selecting if user wants to use exponential backoff for reconnecting. After this the program starts to make API call independently to figure out the API rate limit. When the program has learned the previously unknown rate limit it prints it and stays waiting for user input.

Now the user can start making API calls to the server by just pressing enter key. The program will tell you if the request was successful or not and tells you to wait if the maximum number of requests has been reached. If the rate limit changes and the program starts to sends bad requests to the server, the learning algorithm can be run again by typing ‘refresh’ to provide new rate limit. User can exit the program by typing ‘exit’ to the command prompt.

Possible Improvements

In this form the program doesn’t feature any concurrency so exceeding the rate limit will only allow the user to wait for new tokens to send new requests. By making the requesting and sending of the request different processes, it could enable to user to que requests that would be sent by another process according to the server rate limit. Also, if multiple processes make outbound requests some jitter could be added to the exponential backoff algorithm in order to avoid possible collisions between requests coming from different processes.

The same behaviour as in this program could be easily accomplished by using a premade library like *Celery* or *ratelimit* for python. Also, some APIs directly tell you after HTTP status code 429 the time user needs to wait before the server is ready to receive more queries. Like other rate limiting libraries, also this algorithm could be used as a function decorator to make integration to existing code easier.

For some reason, the results given by the test API are at times bit irregular and so the time frame length gets wrong results sometimes. Occasionally the test API responds back with HTTP code 200 earlier than expected even if the maximum number of requests has already been reached. So, in order to introduce better results to the system, the learning algorithm could be run multiple times and the most promising result could be used as the rate limit.

A clear problem when working with this kind of algorithm is the fact that the user might need to wait for extended periods of time if the server time frames lengths are really long. Unfortunately, if the time frame happens to be rather long and the server doesn’t tell you the wait time, only way to find out the time frame length is to wait.