PACSWG

Nima Mahmoudi

May 22, 2020

CONTENTS:

CHAPTER

ONE

INSTALLATION

Install using pip:

```
$ pip install pacswg
```

Upgrading:

```
pip install pacswg --upgrade
```

For installation in development mode:

```
git clone https://github.com/nimamahmoudi/pacswg
cd pacswg
# have docker installed before this line, it generates the README.rst file
source .travis/build.sh
pip install -e .
```

CHAPTER

TWO

USAGE

The following example shows how we can use this module with any library for making the requests:

```
import requests
import time
import pacswg
import pandas as pd

site_url = 'https://nima-dev.com/'

# the worker function should return a dict with each item having a single value
# return any value you want to keep track of in the dictionary
def worker_func():
    client_start_time = time.time() # current timestamp
    resp = requests.get(site_url)
    client_end_time = time.time() # current timestamp
    resp_len = len(resp.content)
    resp_millis = resp.elapsed.microseconds / 1000
    return {
        'resp_len': resp_len,
        'resp_millis': resp_millis,
        'client_start_time': client_start_time,
        'client_end_time': client_end_time,
    }

# Test the worker function
print(worker_func())

# Create the PACS Workload Generator
wg = pacswg.WorkloadGenerator(worker_func=worker_func, delay_func=lambda x: 1/x,
                          rps=3, worker_thread_count=100)

wg.start_workers()
wg.prepare_test()

timer = pacswg.TimerClass()

# reset the timer
timer.tic()
while timer.toc() < 10:
    wg.fire_wait()
wg.stop_workers()

# Get the results from the workers
res = wg.get_stats()
```

(continues on next page)

(continued from previous page)

```
print('Number of requests:', len(res))
df_res = pd.DataFrame(data=res)

# print the pandas dataframe
print(df_res.head())
```

Which results in the following output:

```
{'resp_len': 53020, 'resp_millis': 122.752, 'client_start_time': 1579129304.3390362,
↪'client_end_time': 1579129304.4707813}
Number of requests: 30
   resp_len  resp_millis  client_start_time  client_end_time
0     53020      119.270       1.579129e+09     1.579129e+09
1     53020      125.307       1.579129e+09     1.579129e+09
2     53020      120.665       1.579129e+09     1.579129e+09
3     53020      141.177       1.579129e+09     1.579129e+09
4     53020      132.713       1.579129e+09     1.579129e+09
```

CHAPTER

THREE

API REFERENCE

3.1 Workload Generator

class wg.**WorkerThread**(*parent*, *sleep_time=2*)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class wg.**WorkloadGenerator**(*worker_func*, *rps=0.16666666666666666*, *delay_func=None*, *worker_thread_count=10*, **args*, ***kwargs*)

WorkloadGenerator is the class responsible for generating the desired workload using the delay function provided, to achieve the target requests per second.

Returns an instance of the WorkloadGenerator class

Return type object

__init__ for WorkloadGenerator class.

Parameters

- **worker_func** (*function*) – the worker function that will be called by the worker threads, it shouldn't have any arguments and should return a dict.
- **rps** (*float, optional*) – desired requests per second to be achieved by the workload generator, defaults to 10/60
- **delay_func** (*function, optional*) – the function that generates a draw from inter-arrival time given rps as an argument, defaults to exponential distribution

- **worker_thread_count** (*int, optional*) – number of worker threads, defaults to 10

fire()
fire causes one of the worker threads to call worker_func once

fire_wait()
fire_wait fires a request, generates an inter-arrival delay using delay_finc, then waits for that amount of time.

get_stats()
get_stats gathers the values generated by calling the workload function throughout the test.

Returns stats

Return type array of dicts

prepare_test()
prepare_test resets the timer that will be used to time the requests.

reset_stats()
reset_stats resets the info gathered from worker threads.

Returns sucess

Return type boolean

set_rps(*new_rps*)
set_rps sets the number of requests per second that will be made by the workers.

Parameters **new_rps** (*float*) – the new rps

Returns success

Return type boolean

start_workers()
start_workers starts up the worker pool

stop_workers()
stop_workers stops all workers and waits until the threads are all shut down.

Returns success

Return type boolean

wg.**get_random_wait_time**(*rps*)
get_random_wait_time generates random exponential inter-arrival times corresponding to Poisson process.

Parameters **rps** (*float*) – rps or requests per second is the target number of requests per second

Returns a draw from the resulting exponential distribution for inter-arrival time

Return type float

3.2 Timer

class timer.**TimerClass**

TimerClass is an object that can help with timing different components of the execution.

Returns TimerClass object

Return type object

tic()

tic resets the starting time fo the timer. toc() will get seconds past since calling tic().

toc()

toc calculates the number of seconds passed since tic() has been called, or object has been created, whichever is more recent.

Returns elapsed time since time reference

Return type float

toc_print()

toc_print prints the value returned by toc_str

Returns number of seconds elapsed with 2 digits of precision.

Return type string

toc_str()

toc_str returns the time elapsed in string with 2 digits of precision. This is for user printing mainly.

Returns number of seconds elapsed with 2 digits of precision.

Return type string

timer.**get_time_in_secs**(*s*)

get_time_in_secs converts the string given to seconds, e.g. 1h is converted to 3600.

Parameters **s** (*string*) – input string of the format "Nu" where N is a number and u is a unit (s/m/h/d/w).

Returns Number of seconds

Return type float

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

w

INDEX