컴퓨터 네트워크 4주차 과제 보고서

-locust를 이용한 서버 부하 테스트-

201702022 배시현

1) 과제 목표

이번 과제는 저번 Socket을 이용한 TCP/IP 통신을 멀티프로세스와 멀티쓰레드로 서버를 구성하고 구성한 서버에 locust를 이용하여 생성한 웹서버의 부하를 테스트하는 과제이다.

RPS (Request Per Second) : 초당 클라이언트의 요청 수이다.

2) 코드 설명

Multiprocess.py

```python
from multiprocessing import Process
import socket
import time
import os

def multiprocess_sr(csocket, addr):
    data = csocket.recv(1024)
    print(f"[Client {addr} Info] {data.decode()}")
    res = "HTTP/1.1 200 OK\nContent-Type: text/html\n\n"
    csocket.send(res.encode('utf-8'))
    csocket.send(data)
    csocket.close()


def main(port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('',port))
    server_socket.listen()
    req_clients = list()

    try:
        while True:
            csocket, address = server_socket.accept()
          #  req_clients.append(csocket)
            proc = Process(target=multiprocess_sr, args=(csocket, address))
            proc.start()
    except Exception:
        proc.join()

if __name__ == "__main__":
    port = 8890
    main(port)
```

Multiprocess_sr의 경우 singleprocess에서 주어진 코드를 적당히 바꾸어서 넣었다.

Try 다음 부분에서는 과제자료에 설명되어 있는 것처럼 accept를 먼저 해서 클라이언트의 접속을 허가한뒤 접속한 클라이언트를 리스트에 넣는 부분이 있었는데 이부분을 구현하여 실행하게 되면 버퍼사이즈를 벗어나서 Too many file open에러가 발생하거나 Device or Resource busy가 계속 발생하여서 제거 하였다. 그 다음 멀티프로세스를 실행하는 방법을 이용하여 구현했다.

Multithread.py

```python
from threading import Thread
import socket
import time

def multithread_sr(csocket, addr):
    data = csocket.recv(1024)
    print(f"Client {addr} Info] {data.decode()}")
    res = "HTTP/1.1 200 OK\nContent-Type: text/html\n\n"
    csocket.send(res.encode('utf-8'))
    csocket.send(data)
    csocket.close()

def main(port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('', port))
    server_socket.listen()
    req_clients = list()

    try:
        while True:
            (csocket, address) = server_socket.accept()
       #    req_clients.append(csocket)
            th = Thread(target=multithread_sr, args=(csocket,address))
            th.start()
    except:
        th.join()

if __name__ == "__main__":
    port = 8891
    main(port)
```

이 경우 멀티프로세스와 코드에서 약간 다르기 때문에 설명을 생략하고 바로 분석결과로 들어가겠다.

Locustfile.py

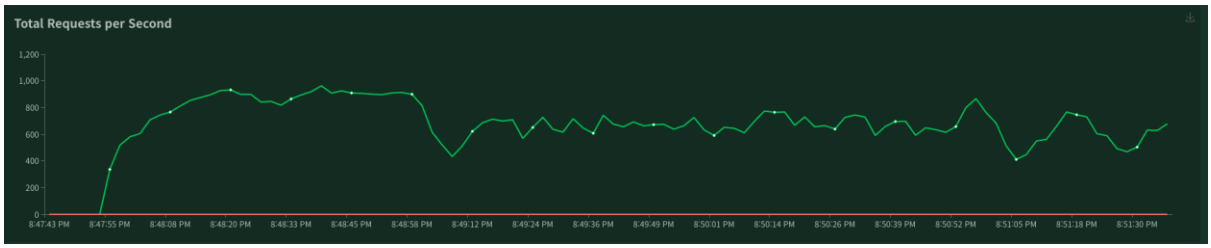
```python
import time
from locust import HttpUser, task, between, TaskSet

class QuickstartUser(HttpUser):
    wait_time = between(5,15)


    @task
    def Mytask(self):
        response = self.client.get("/")
~
~
~
```

Locust 파일 경우 많은 검색과 문서를 봤지만 이번 과제에서 원하는 것은 client의 요청이 많을경우라고 생각하여서 별다른 일을 하지 않는 파일로 생성하였다.

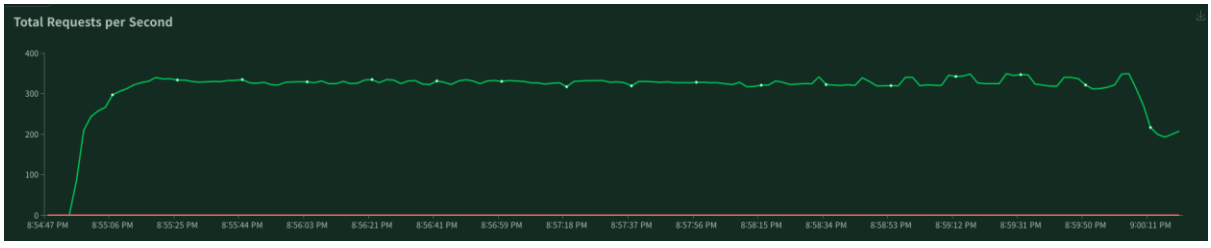
3) 분석

싱글프로세스와 멀티프로세스의 경우 아무리 기다리거나 유저수를 증가시켜도 failure가 발생하지 않았다. 그래서 요청한 클라이언트 수만큼 올려갔을 때 stop을 눌러서 분석하였다.

(증거 사진 첨부)

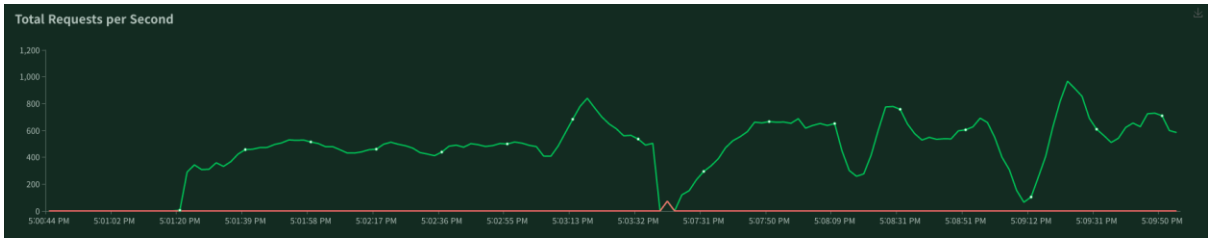


Single process 4분간 17만개의 요청을 받았으나 failure가 발생하지 않음



Multi process 6분간 10만개의 요청을 받았으나 failure가 발생하지 않음

(Single process)

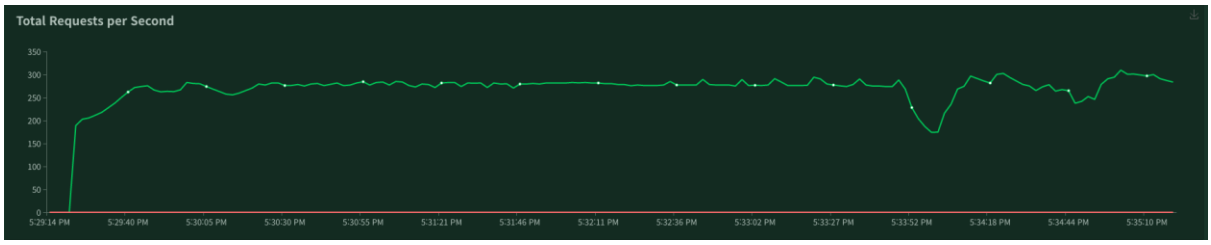


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
reOffice Writer		88503	3714	4200	11000	4778	3	19970	139	644.4	187.8
	Aggregated	**88503**	**3714**	**4200**	**11000**	**4778**	**3**	**19970**	**139**	**644.4**	**187.8**

Client 수를 10000으로 설정하였고 hatch rate를 1000으로 설정하였다. 올라가있는 fails는 Stop하고 cleaning 상태에 발생하였다.

Failure이 발생하지 않았고 총 요청 88503번에 RPS가 644.4이며, 평균응답속도는 4778ms이다.

(Multiprocess)

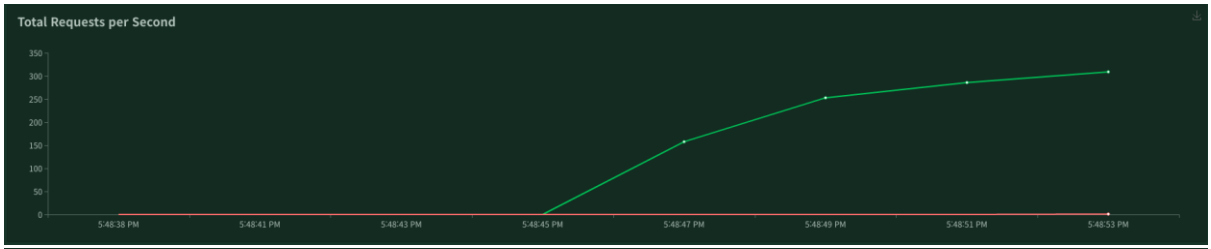


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	98675	0	11000	26000	12426	8	32061	145	273.8	0
	Aggregated	**98675**	**0**	**11000**	**26000**	**12426**	**8**	**32061**	**145**	**273.8**	**0**

Client 수를 10000으로 설정하였고 hatch rate를 1000으로 설정하였다.

Failure이 발생하지 않았고 총 요청은 98675에 RPS가 273.8.이며, 평균응답속도는 12425ms이다.

(Multithread)



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
reOffice Writer		3725	143	350	1600	667	1	8122	139	322.44	6.56
	Aggregated	**3725**	**143**	**350**	**1600**	**667**	**1**	**8122**	**139**	**322.44**	**6.56**

Client 수를 10000으로 설정하였고 hatch rate를 1000으로 설정하였다.

RPS가 322.44이며, 3725번의 요청을 했을 시 failure가 발생한다. 그리고 평균 응답시간은 667ms 이다.

4) 분석평가

결론적으로 봤을 때 일단 failure이 발생하게 되면 신뢰성이 떨어지기 때문에 Multithread 방식을 이용하는 것은 좋지 않다고 판단하였다. 나머지 두개를 비교해볼 때 Singleprocess의 경우가 Multiprocess보다 RPS가 높고, 평균응답속도가 빠르기 때문에 가장 효율적이라고 생각되는 부분은 SingleProcess이다.

효율적인 방법 순　SingleProcess > Multiprocess > MultiThread(신뢰성 문제)

하지만 상식적인 생각으로는 뭔가 잘못된 것 같다. 하나의 프로세스만 사용하는데 여러 개의 프로세스를 사용하는 서버보다 빠른거는 이상하다.