

컴퓨터 네트워크

-Server Stress Test2(7주차)-

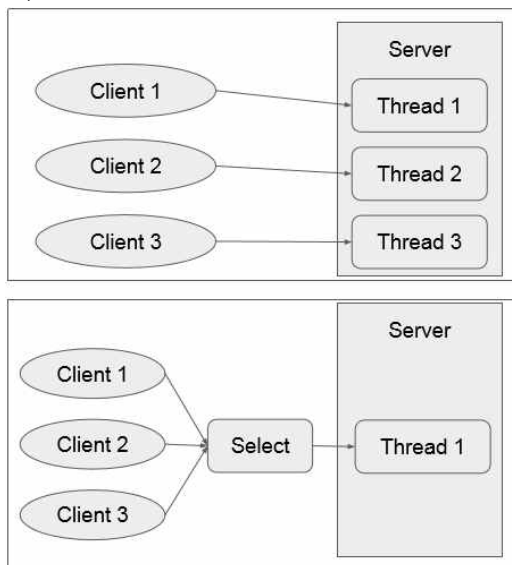
학번: 201404377

이름: 진승언

[과제목표]

이번과제는 저번과제에 이어서 locust를 이용한 Server Stress Test 즉 웹서버 구현 방법에 따른 성능비교를 하는 것이다. 저번 과제에 이어 추가로 join()의 위치를 수정하고 HTTP Server를 구현하는데 Select를 이용한 Multiplexing을 추가하였다. 결론적으로 총 구현 방법은 SingleProcess, MultiProcess, MultiThread, Select이다.

Select란 하나 이상의 file descriptor가 가 I/O에 대해서 'ready'상태가 될때 까지 기다리면서 여러 파일 디스크립터를 모니터 하는 것이다. Select Multiplexing(다중화)을 사용하여 select가 여러 socket descriptor를 관리하고 스레드, 프로세스와 달리 context switching이 발생하지 않아 효율적이다.



수용 가능한(locust 실행 후Fail이 뜰때까지의) 최대 RPS와 다중 클라이언트 환경에서 평균 응답시간에 초점을 맞추어 결과를 확인했다.

[과제 해결방법]

3. 싱글프로세스 서버 코드

```
import time
import os
import socket

def send_rcv(client_socket):
    data = client_socket.recv(1024)
    print("[client {}] {}".format(os.getpid(), data.decode()))
    response = "HTTP/1.1 200 OK\r\n"
    client_socket.send(response.encode('utf-8'))
    client_socket.send(data)
    client_socket.close()

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind(('', FLAGS.port))
    serversocket.listen()

    while True:
        (clientsocket, address) = serversocket.accept()
        print("accept client from", address)
        send_rcv(clientsocket)

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port',
                        type = int,
                        default = 8000,
                        help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

2. 멀티스레드 서버 코드

```
from threading import Thread
import socket
import os

def send_rcv(client_socket, address):
    data = client_socket.recv(1024)
    print("[client {}] {}".format(os.getpid(), data.decode()))
    msg = "HTTP/1.1 200 OK\r\n"
    client_socket.send(msg.encode('utf-8'))
    client_socket.send(data)
    client_socket.close()

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind(('', FLAGS.port))
    th = None
    print("listening...")
    serversocket.listen(5)
    client_socket = list()
    try:
        while True:
            client, address = serversocket.accept()
            print("accept client from", address)

            th = Thread(target=send_rcv, args=(client, address))
            th.start()

    except Exception:
        th.join()

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type = int, default=8000, help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

3. 멀티프로세스 서버 코드

```
from multiprocessing import Process
import socket
import os

def send_recv(client, address):
    msg = client.recv(1024)
    print("[client {}] {}".format(os.getpid(), msg.decode()))
    response = "HTTP/1.1 200 OK\r\n"
    client.send(response.encode('utf-8'))
    client.close()

def main(FLAGS):
    proc = None
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind('', FLAGS.port)
    serversocket.listen(5)
    clients = list()
    try:
        while True:
            client, address = serversocket.accept()
            print("accept client from", address)
            clients.append(client)
            proc = Process(target=send_recv, args = (client, address))
            proc.start()
    except Exception:
        proc.join()

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type = int, default=8000, help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

먼저 저번에 했던 멀티스레드 멀티프로세스 소켓서버 코드의 join() 위치를 수정하였다. 인터럽트 즉 예외가 발생하면 join()을 실행하게 수정하였다.

4. select 서버 코드

```
import socket
import select
import os

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind(('', FLAGS.port))
    serversocket.listen()
    print("listening ...." )
    readable = [serversocket]
    writable = []
    error = []

    while True:
        try:
            r_list, w_list, e_list = select.select(readable, writable, error)
            for read_desc in r_list:
                if read_desc == serversocket:
                    client_socket, client_addr = serversocket.accept()
                    r_list.append(client_socket)
                    print("from {}".format(client_addr))
                else:
                    data = read_desc.recv(1024)
                    if data:
                        response = "HTTP/1.1 200 OK\r\n"
                        read_desc.send(response.encode('utf-8'))
                    else:
                        read_desc.close()
                        r_list.remove(read_desc)
            except Exception as err:
                print("EXCEPTION !!!", err)


if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type = int, default=8000, help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

select를 사용한 서버의 코드이다. 클라이언트가 소켓을 select가 관리한다. r_list(리스트)에 클라이언트 소켓들이 저장이 된다. 그리고 처리한 클라이언트 소켓은 리스트에서 제거된다.

다음은 locust 실행결과이다.

들어가기에 앞서 Failure는 다음과 같이 동일합니다. (멀티스레드의 Failure)

 LOCUST

HOST
http://172.30.1.36:2345

STATUS
STOPPED
[New test](#)

RPS
535.1

FAILURES
2%


Statistics Charts Failures Exceptions Download Data

# fails	Method	Name	Type
181	GET	/	ConnectionError(MaxRetryError("HTTPConnectionPool(host='172.30.1.36', port=2345): Max retries exceeded with url: / (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x...>: Failed to establish a new connection: [Errno 24] Too many open files.))",))

1. SingleProcess

충남대학교 포탈

Locust

 LOCUST

HOST
http://172.30.1.36:2345

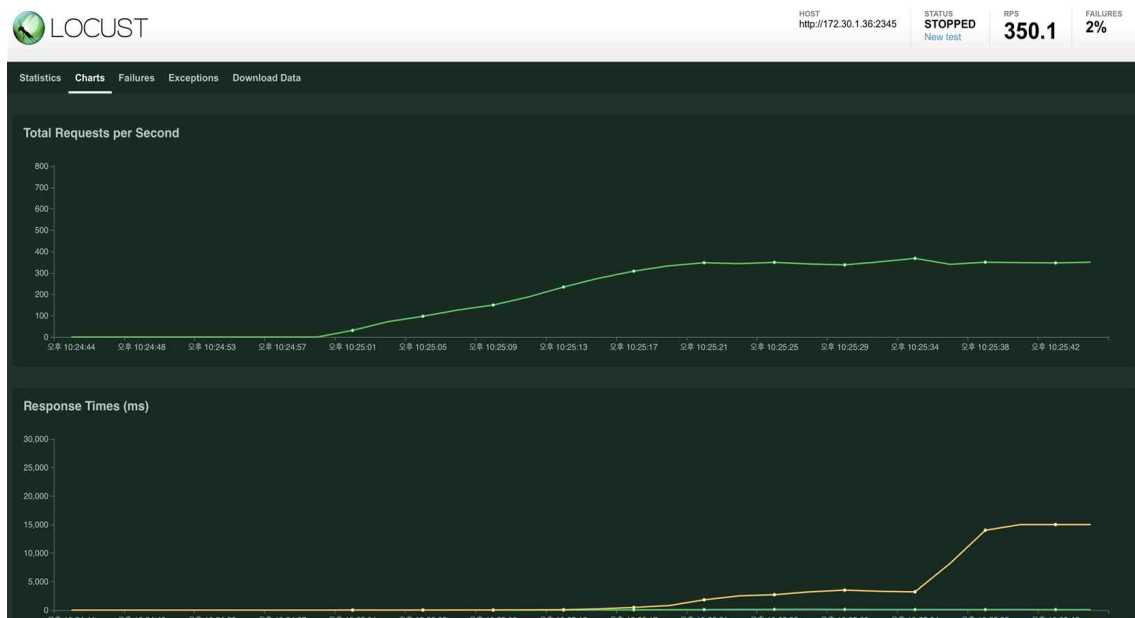
STATUS
STOPPED
[New test](#)

RPS
350.1

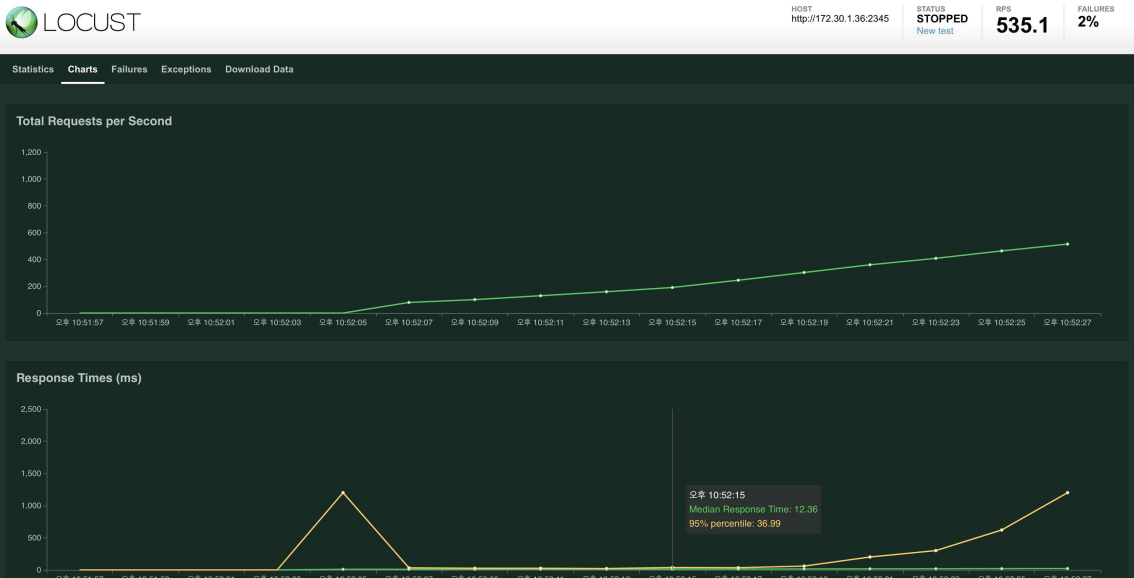
FAILURES
2%

Statistics Charts Failures Exceptions Download Data

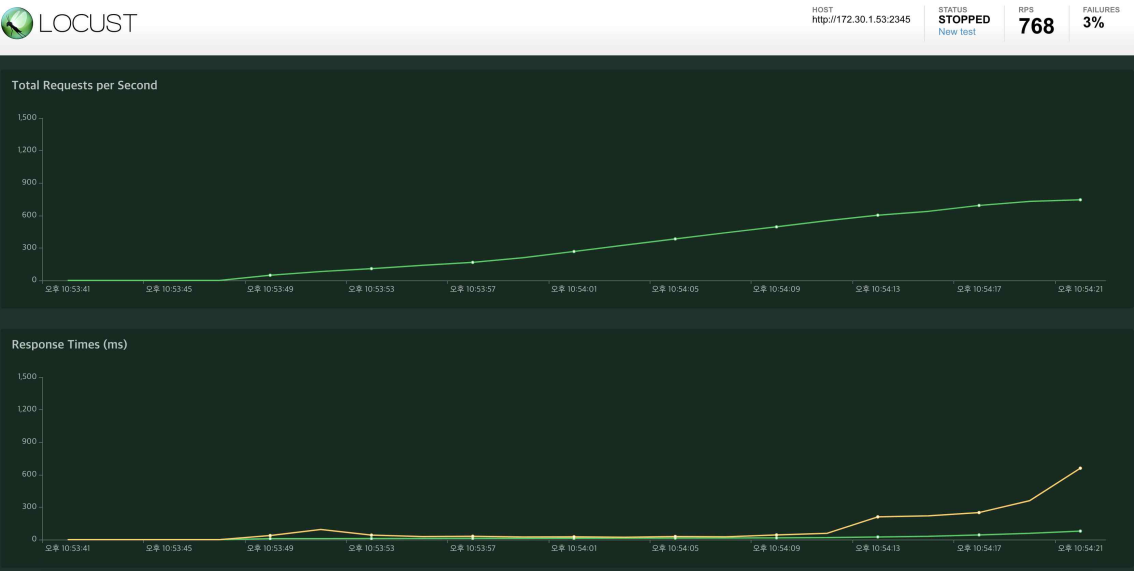
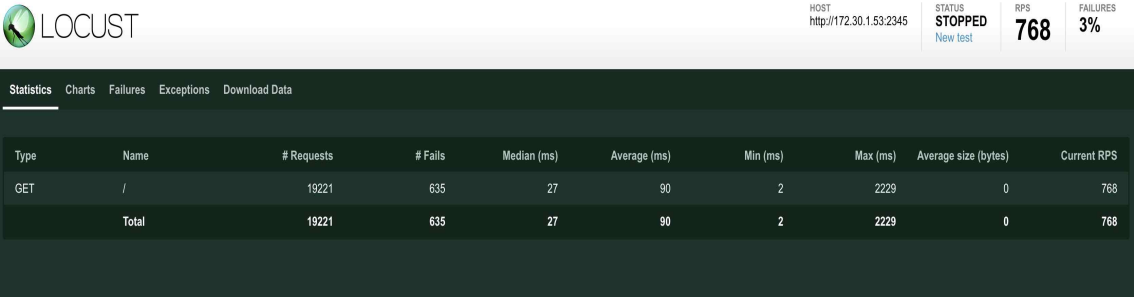
Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	14362	293	80	748	2	22823	0	350.1
	Total	14362	293	80	748	2	22823	0	350.1



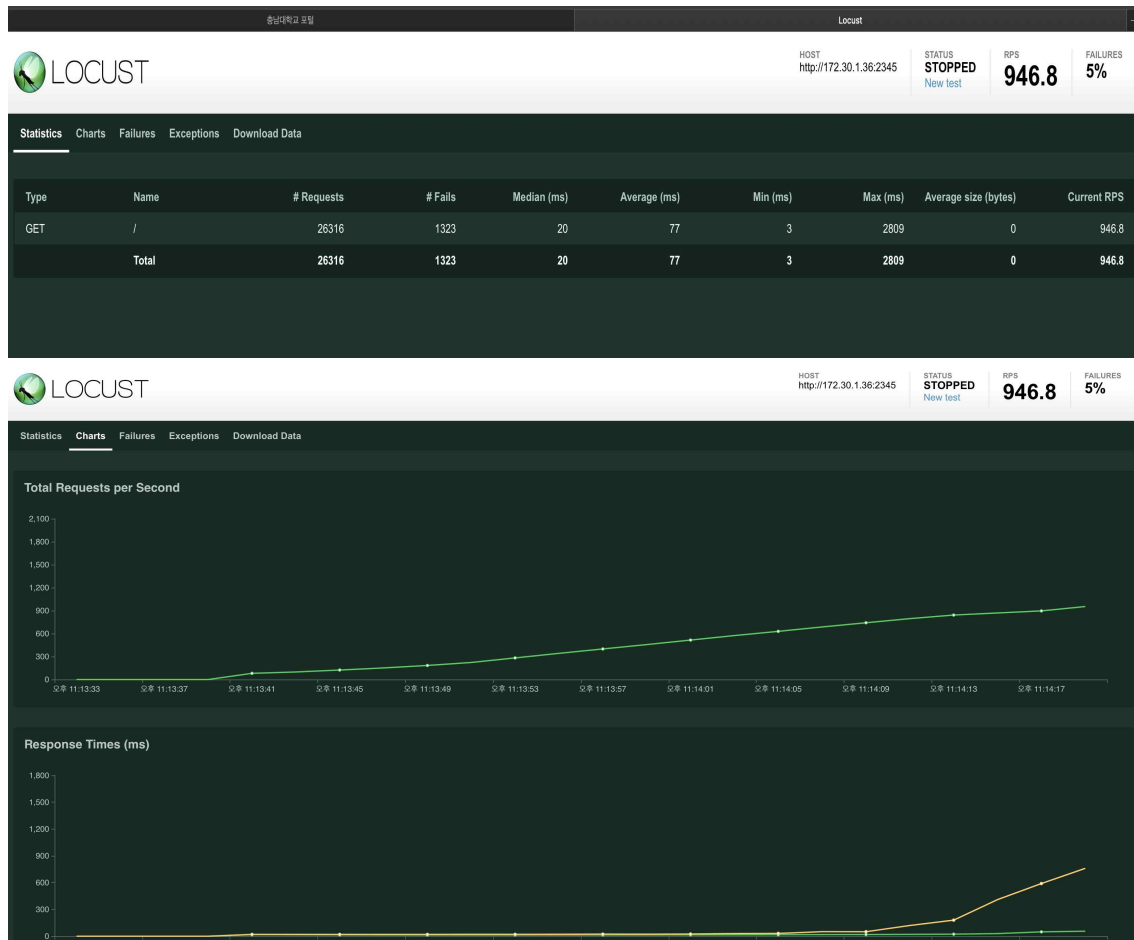
2. MultiThread



3. MultiProcess



4. Select



[비교]

Hatch Rate 100 , 10000 Requests	Request	Request Per Second	Average Response time
Single Process	14362	350.1	748
Multi Process	19221	768	90
Multi Thread	10237	535.1	119
Select	26316	946.8	77

RPS는 Select > Multi Process > Multi Thread > Single Process 순이고
평균응답속도를 빠른 순으로 적으면 다음과 같다.

Select, Multi Process, Multi Thread, Single Process

전체적으로 Select가 성능이 우수하다는 것을 알 수 있었다.