* 과제 목표 (도출해야 할 결과)
* 단일프로세스 / 멀티프로세싱 / 멀티스레딩 통신부하 비교
* 코드 설명과 과제 해결 방법
* 멀티프로세싱 / 멀티스레딩 대상 함수 : 같은 동작을 수행하고, 이름만 다르다. 대상 클라이언트 소켓에 대하여 수신하고, 답장을 전송한 후 소켓을 닫는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 멀티프로세싱 수행 내용 : Accept 수행 후 받은 클라이언트들의 소켓을 리스트로 모은 뒤, 순회하며 각 소켓마다 프로세스를 생성한다. Exception이 발생한 경우, 각 프로세스들의 종료를 join()을 통해 기다린다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 멀티스레딩 수행 내용 : 클라이언트 소켓의 개별 처리에 프로세스 대신 스레드를 사용한다는 점 말고는, 전부 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* locustfile : 단순히 서버에 GET요청을 하기 위한 파일이며, locust의 수행 내용이다.

wait\_time이 1~5초 사이를 기다리도록 오버라이딩되고,

task 데코레이터를 통해 task의 내용이 직접 정의한 index함수와 같이 수행된다.

따라서 1~5초 사이의 간격을 두고 root 디렉토리에 대해 get을 수행하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 실행 결과 (공통적)

1. 성공 : 메시지를 잘 주고받음

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 실패 : Broken pipe (요청이 많을 경우 발생한 것으로 추정, multiprocessing에서 특히 많이 나타남)



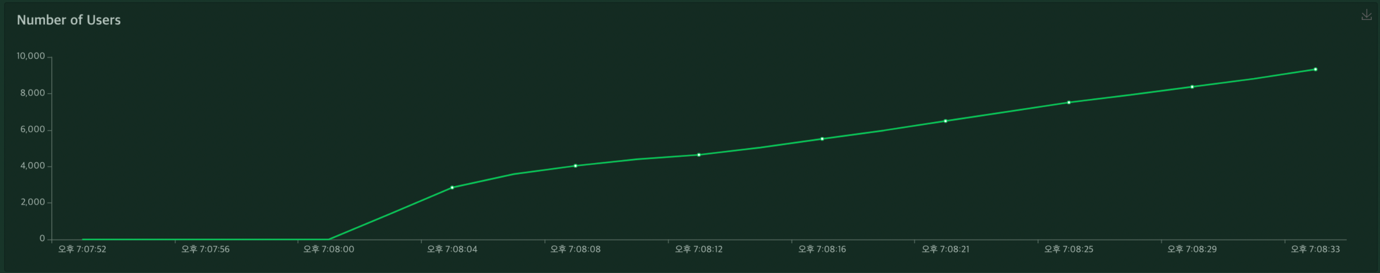
* Single Process vs Multi Process vs Multi Thread
* Total users : 10000, Hatch rate : 1000
* Single Process : RPS가 204.8이며, 18672번의 요청을 했을 시 failure가 발생한다. 그리고 평균 응답시간은 132ms이다.

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

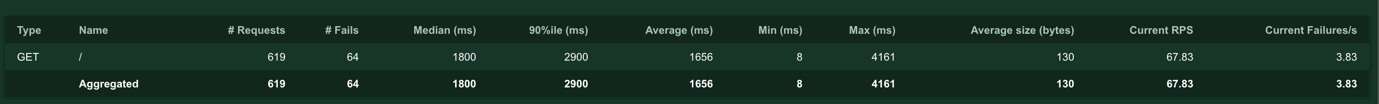
자동 생성된 설명

텍스트, 모니터, 스크린샷이(가) 표시된 사진

자동 생성된 설명



* Multi Process : RPS가 67.83이며, 619번의 요청을 했을 시 failure가 발생한다. 그리고 평균 응답시간은 1656ms이다.



텍스트, 모니터, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 레이저, 밤하늘이(가) 표시된 사진

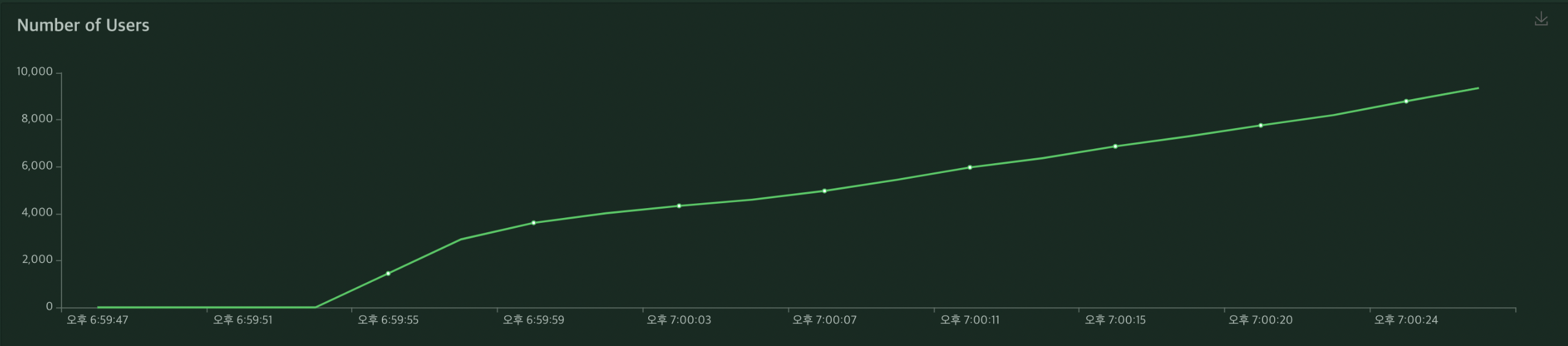
자동 생성된 설명

* Multi Thread : RPS가 320.4이며, 20522번의 요청을 했을 시 failure가 발생한다. 그리고 평균 응답시간은 88ms이다.



텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명



* Multi Thread를 쓰는 방법이 가장 효율적이다. 이는 Request per Second가 가장 높고 응답시간도 가장 빠르며, Failure 까지 걸리는 Request 수도 가장 적어 안정적이기 때문이다.
* 반면에 Multi Process를 쓰는 방법은 Failure가 많이 발생하여 매우 불안정하였기 때문에, 이는 적절하지 않다는 생각이 들었다.
* 과제 느낀 점

컴퓨터가 자꾸 테스트 중에 리셋돼서 힘들었다..