SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

08 PJT

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Django 에서 알고리즘 구현 및 성능측정

챕터의 포인트

- 목표
- 테스트 & 성능 테스트 개념 (with. Locust)
- [실습] 정렬 알고리즘 성능 측정

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

목표

프로젝트 목표

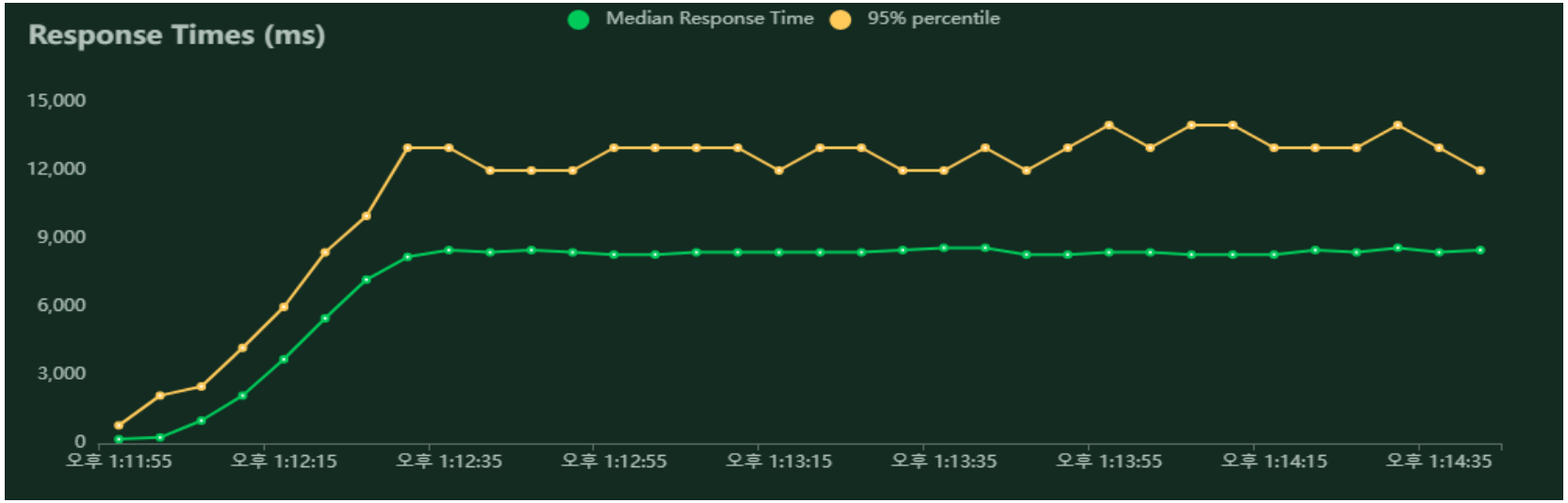
- 이번 프로젝트에서는

  Django 에서 요구사항에 따라 알고리즘을 구현해보고

  구현한 알고리즘의 성능을 측정합니다.

프로젝트 목표

- Locust 라는 툴을 이용하면 사진과 같이 성능 테스트를 할 수 있습니다.



진행 순서

1. 테스트 & 성능 테스트 개념
2. Locust 세팅
3. 정렬 알고리즘 성능 측정 실습

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

테스트

테스트란 ?

- 원하는 기능이 모두 구현되었는 지 확인하고, 숨겨져 있는 결함을 찾는 활동

- 여러가지 도구들을 활용하여

  버그를 찾아내고 신뢰성, 보안, 성능 등을 검증하는 중요한 단계

테스트란 ?

- 테스트의 종류는 너무 많다..!! (100가지가 넘습니다)

- 상황에 맞게 필요한 테스트를 진행해야 합니다.
  - 외우지 말고 검색을 통해 필요한 것만 찾아서 사용해야 합니다.

성능 테스트

- 핵심 적인 테스트 중 하나
- 특정 상황에서 시스템이 어느 정도 수준을 보이는가 혹은 어떻게 대처를 하는가를 테스트하는 과정
- 목적
  - 여러 테스트를 통해 성능 저하가 발생하는 요인을 발견하고 제거
  - 시장에 출시되기 전에 발생할 수 있는 위험과 개선사항을 파악
  - 안정적이고 신뢰할 수 있는 제품을 빠르게 만들기 위함

성능 테스트 예시

엄청 많은 사람들이

한 번에 요청을 보내면 ?

서비스

우리는 새벽에 쓸거야

우리는 30시간 동안 사용할거야

성능 테스트

- 성능 테스트의 종류도 많습니다.
- 그 중, 프로젝트에서는 핵심인 **부하 테스트**와 **스트레스 테스트**를 배웁니다.

부하 테스트 (Load Testing)

- 시스템에 임계점의 부하가 계속될 때 문제가 없는가 ?
- 목적 : 시스템의 신뢰도와 성능을 측정

임계점에 해당하는 인원이
30시간 동안 계속해서 사용

서비스

임계점: 사용자 혹은 요청이 점점 늘어나다가, 응답시간이 급격히 느려지는 시점

스트레스 테스트 (Stress Testing)

- 시스템에 과부하가 오면 어떻게 동작할까 ?
- 목적 : 장애 조치와 복구 절차가 효과적이고 효율적인 지 확인

어떻게 대처하는가

임계점 이상의 인원

서비스

임계점: 사용자 혹은 요청이 점점 늘어나다가, 응답시간이 급격히 느려지는 시점

부하 테스트 vs 스트레스 테스트

	부하 테스트(Load Testing)	스트레스 테스트(Stress Testing)
도메인	성능 테스트의 하위 집합	성능 테스트의 하위 집합
테스트 목적	전체 시스템의 성능 확인	중단점에서의 동작, 복구 가능성 확인
테스트 방법	임계점 까지의 가상 유저 수를 유지하며 모니터링	중단점 이상까지 가상 유저를 점진적으로 증가
테스트 대상	전체 시스템	식별된 트랜잭션에만 집중하여 테스트
테스트 완료 시기	예상 부하가 모두 적용된 경우	시스템 동작이 중단되었을 경우
결과	부하 분산 문제 최대 성능 시간 당 서버 처리량 및 응답 시간 최대 동시 사용자 수 등	안정성 복구 가능성

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

API 성능 테스트

Locust

- 오픈 소스 부하 테스트 도구
- 번역하면 메뚜기. 테스트 중 메뚜기 떼가 웹 사이트를 공격한다는 의미로 착안된 이름
- 내가 만든 서버에 수많은 사용자들이 동시에 들어올 때 어떤 일이 벌어 지는 지를 확인하는 부하 테스트를 할 수 있는 도구
- Locust 를 선택한 이유
  - 파이썬 언어로 테스트 시나리오를 간편하게 작성할 수 있습니다.
  - 결과를 웹에서 확인할 수 있는 UI를 지원합니다.

금융상품비교

영화추천서비스

Locust 사용법 - (1/10)

1. 테스트 스크립트 작성하기 (공식문서 참조)

공식 문서 코드

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

class QuickstartUser(HttpUser):
    wait_time = between(1, 5)

    @task
    def hello_world(self):
        self.client.get("/hello")
        self.client.get("/world")

    @task(3)
    def view_items(self):
        for item_id in range(10):
            self.client.get(f"/item?id={item_id}", name="/item")
            time.sleep(1)

    def on_start(self):
        self.client.post("/login", json={"username":"foo", "password":"bar"})
```

- **HttpUser**: HTTP 요청을 만드는 가상 유저
- **wait_time**: 작업 간 대기 시간
- **on_start()**: 가상 유저 생성 시 실행
- **@task**: 유저가 실행할 작업
- **@task(N)**: 가중치 (실행 확률)
  - N만큼 높은 확률로 작업을 수행
- **self.client.get**: HTTP GET 요청 전송

Locust 사용법 - (2/10)

2. Django 서버 실행 - 제공된 Django API 서버를 실행합니다.

```
$ cd performence_test

$ python -m venv venv

$ source venv/Scripts/activate

(venv) $ pip install -r requirements.txt

(venv) $ python manage.py makemigrations

(venv) $ python manage.py migrate

(venv) $ python manage.py runserver
```

Locust 사용법 - (2/10)

3. vscode 터미널 추가 & Locust 설치 및 실행

```
(venv) $ pip install locust

(venv) $ locust -f ./locust_test.py
```

금융상품비교

영화추천서비스

Locust 사용법 - (3/10)

4. Locust 정상 실행 시 터미널에 아래와 같이 접속할 수 있는 URL 이 출력됩니다.

```
(venv)
GGR@DESKTOP-K5GQ6M2 MINGW64 ~/Desktop/ssafy-python-9th-pjt/pjt08 (master)
$ locust -f ./locust_test.py
[2023-02-08 16:05:10,980] DESKTOP-K5GQ6M2/INFO/locust.main: Starting web interface at http://0.0.0.0:8089 (accepting connections from all network interfaces)
[2023-02-08 16:05:10,996] DESKTOP-K5GQ6M2/INFO/locust.main: Starting Locust 2.14.2
[2023-02-08 16:09:07,111] DESKTOP-K5GQ6M2/INFO/locust.runners: Ramping to 500 users at a rate of 10.00 per second
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
```

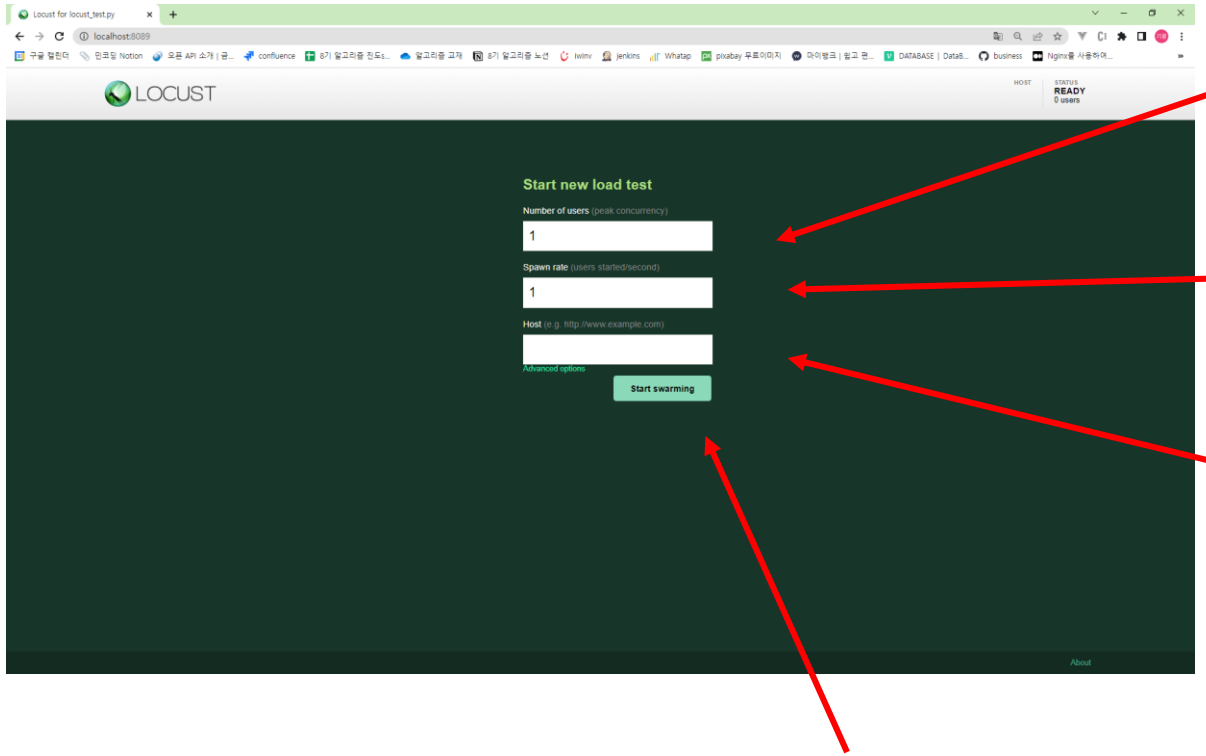
- http://localhost:8089 로 접속하면 Web 화면을 볼 수 있습니다.
  - [주의사항] 콘솔에서 출력되는 http://0.0.0.0:8089 로 접속하면 에러가 납니다!

금융상품비교

영화추천서비스

Locust 사용법 - (4/10)

5. 웹 실행 화면 (http://localhost:8089 접속)



- Number of users
  - 생성할 총 가상유저 수
- Spawn rate
  - 동시에 접속하는 유저 수
- Host
  - 서버 주소(Django 서버)
  - ex) http://localhost:8000/

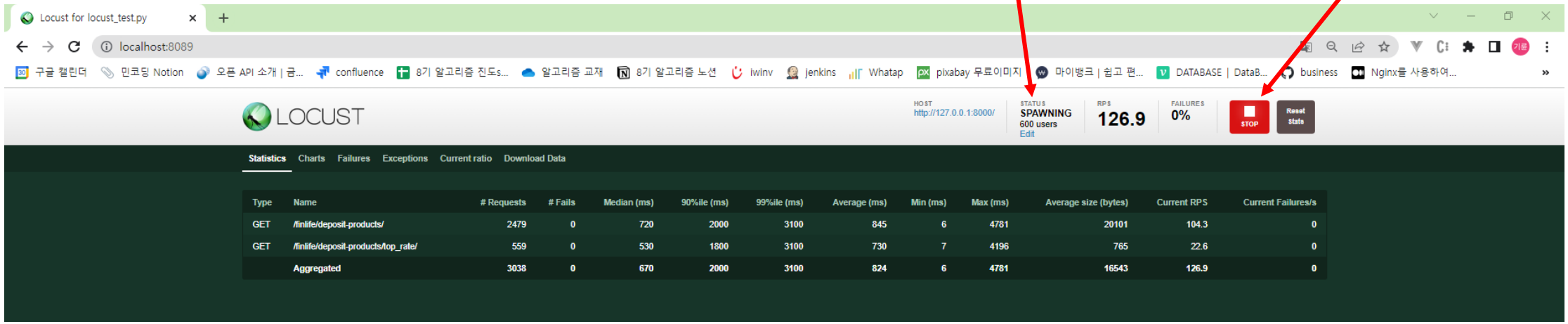
클릭 시 가상 유저에 등록된 작업을 수행합니다.

Locust 사용법 - (5/10)

6. 웹 실행 화면 - Statistics 탭

접속 유저 수
edit: 유저 수 설정 수정 가능

작업 중지

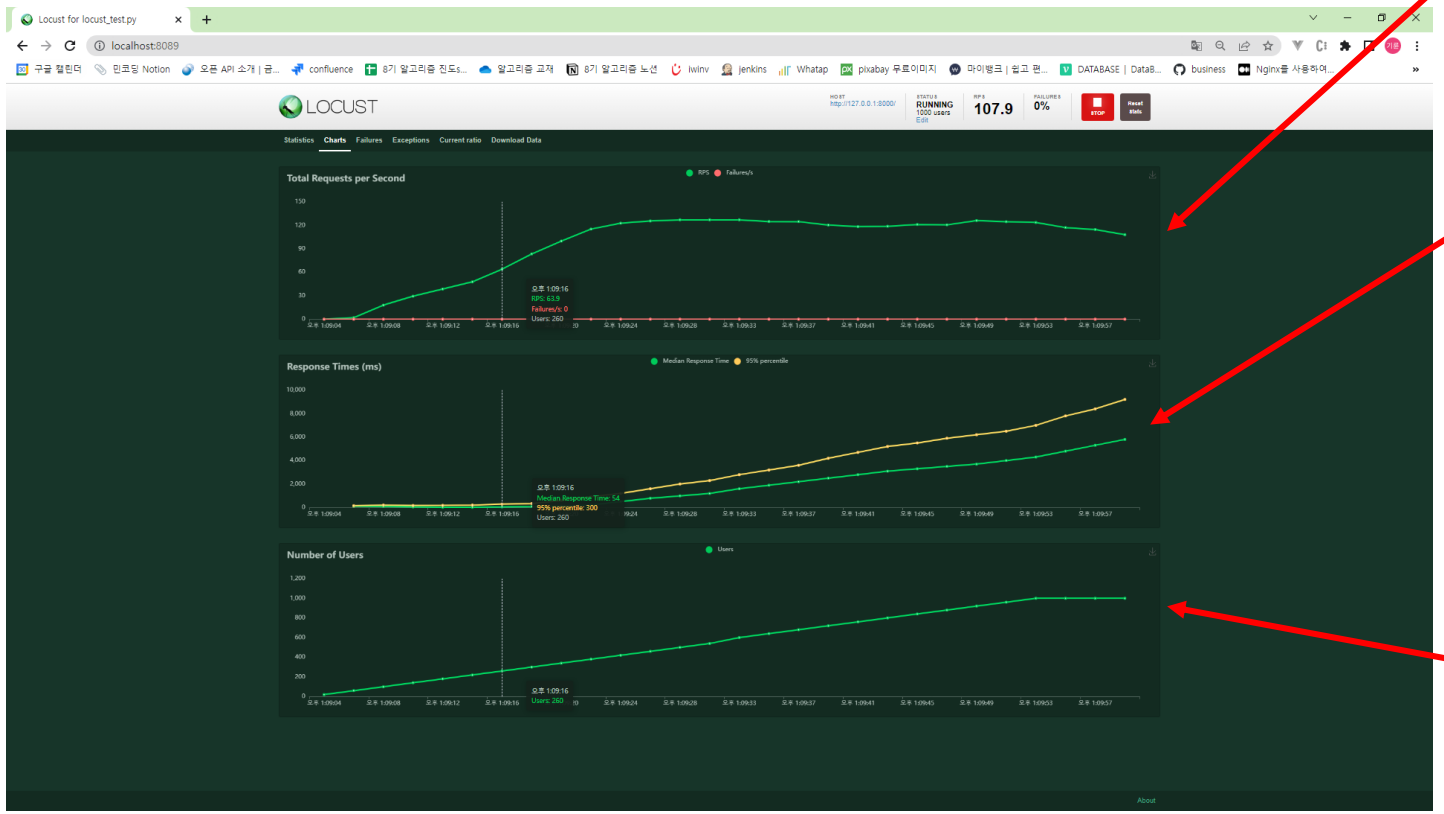


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/finlife/deposit-products/	2479	0	720	2000	3100	845	6	4781	20101	104.3	0
GET	/finlife/deposit-products/top_rate/	559	0	530	1800	3100	730	7	4196	765	22.6	0
	Aggregated	3038	0	670	2000	3100	824	6	4781	16543	126.9	0

- 각 URL 에 대한 **요청 수, 실패 수, 각 기준에 대한 응답 시간, 평균 응답 크기, RPS 등** 다양한 통계 내용을 확인할 수 있습니다.
- 전체 분석은 터미널에서 터미널 종료(Ctrl + C) 입력 또는 Download Data 탭의 Download Report 클릭 시 확인할 수 있습니다.

Locust 사용법 - (6/10)

7. 웹 실행 화면 - Charts 탭



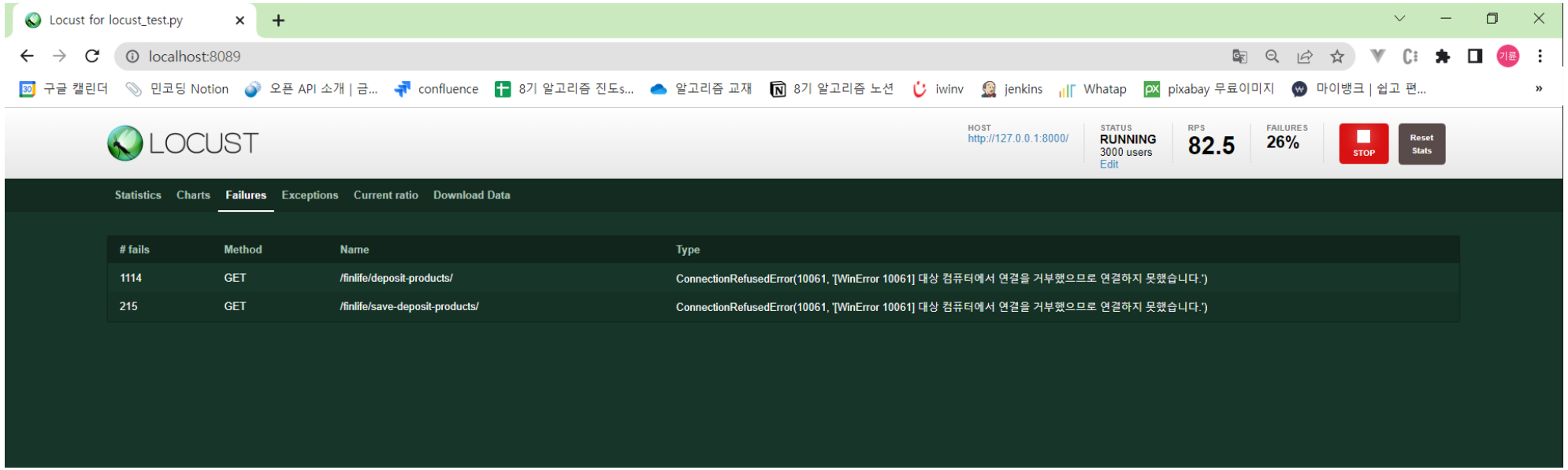
- 가로는 모두 시간을 의미합니다.
- Total Requests per Second
  - 초록선: 초당 요청 수(RPS)
  - 빨간선: 초당 실패한 요청 수
- Response Times(ms)
  - 각 응답에 대한 평균 응답 시간
  - 노란선: (95% Percentile)
    - 95% 응답이 해당 시간 내에 처리되었다.
  - 초록선: (Median)
    - 응답 시간의 중앙값
- Number of Users
  - 동시에 요청을 보내는 유저 수

금융상품비교

영화추천서비스

Locust 사용법 - (7/10)

8. 웹 실행 화면 - Failures 탭



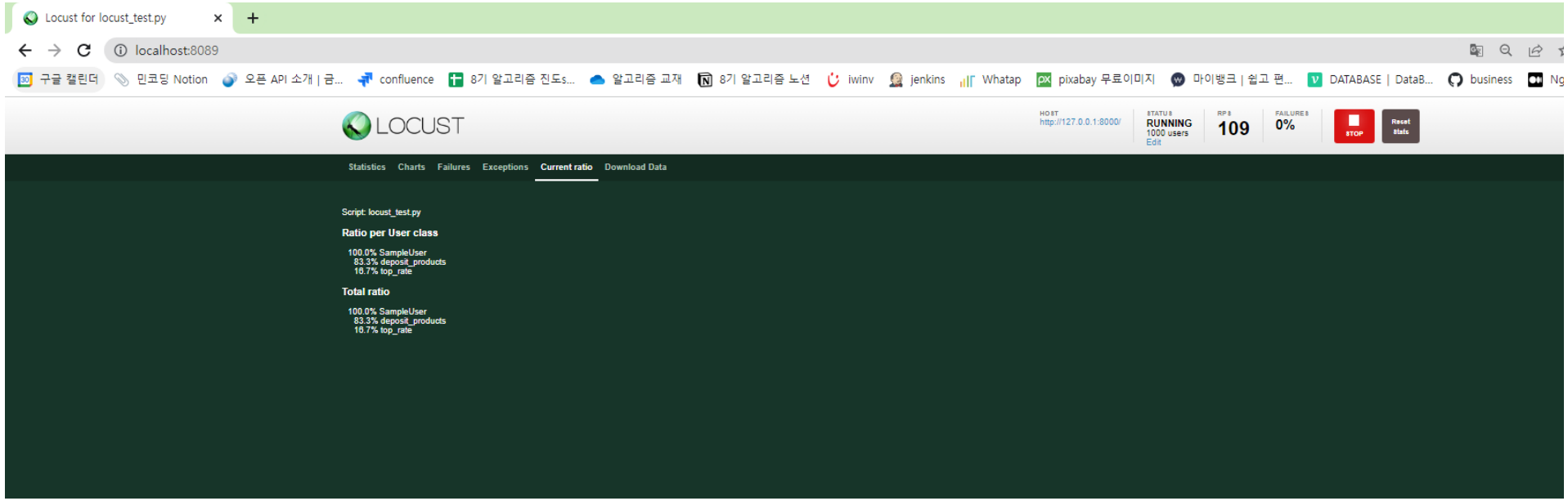
- 실패한 요청에 대한 정보와 실패 원인이 출력됩니다.
  - ex) 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.

금융상품비교

영화추천서비스

Locust 사용법 - (8/10)

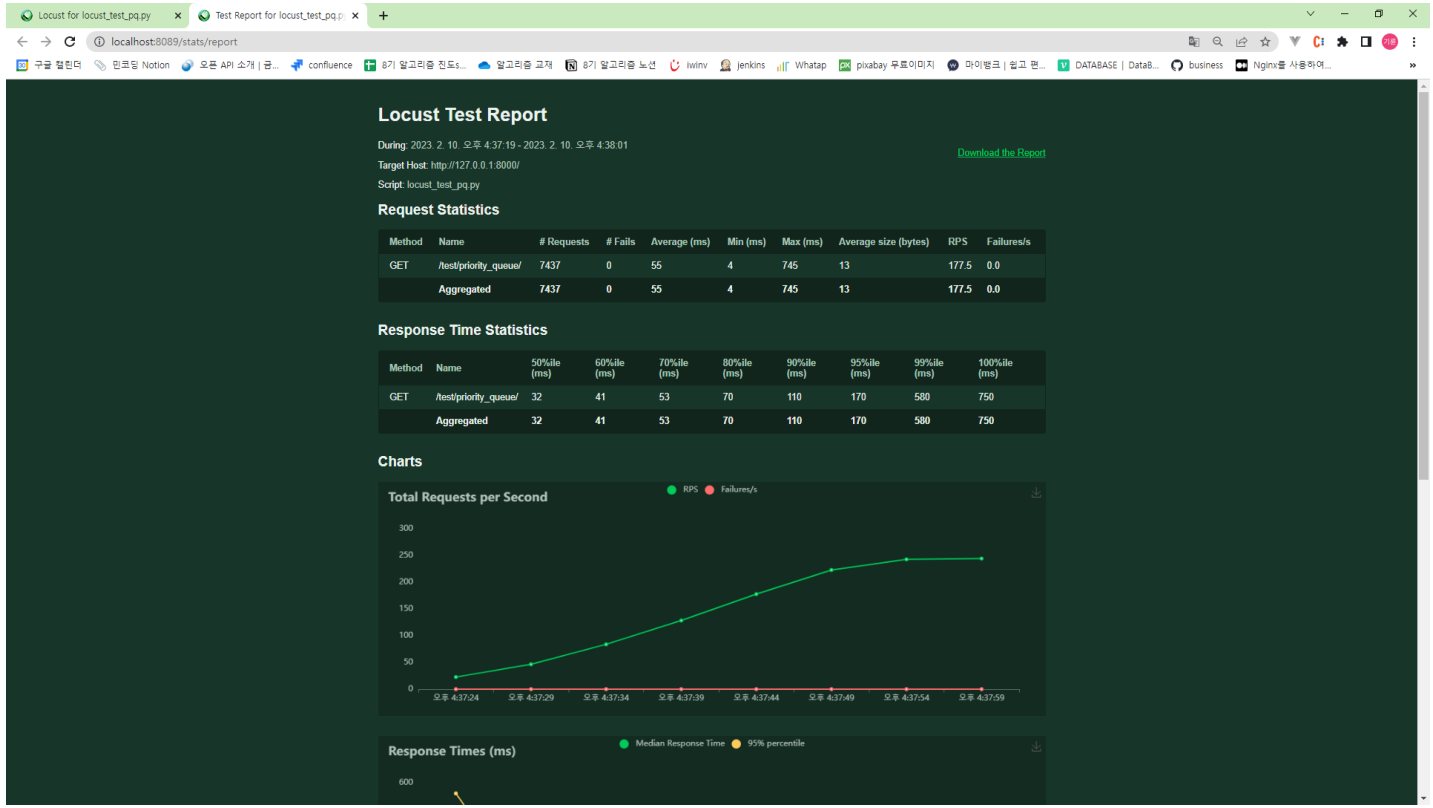
9. 웹 실행 화면 - Current ratio 탭



- 현재 작업이 수행된 비율을 출력합니다.

Locust 사용법 - (9/10)

10. 웹 실행 화면 - 결과 화면(Download Data -> Download Report)



Locust 사용법 - (10/10)

11. 콘솔 종료 화면

```
Traceback (most recent call last):
  File "C:\Users\GGR\Desktop\ssafy-python-9th-pjt\pjt08\venv\lib\site-packages\gevent\_ffi\loop.py", line 270, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
2023-02-09T02:53:20Z
[2023-02-09 11:53:20,975] DESKTOP-K5GQ6M2/INFO/locust.main: Shutting down (exit code 1)
Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
GET      /finlife/deposit-products/                                                     1397  221(15.82%) |   5051     189   13341   4400 |   51.79        8.19
GET      /finlife/save-deposit-products/                                                  41   38(92.68%) |   2700    2040   12215   2100 |    1.52        1.41
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                                                     1438  259(18.01%) |   4984     189   13341   4300 |   53.31        9.60

Response time percentiles (approximated)
Type     Name                                                                             50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
GET      /finlife/deposit-products/                                                      4400   6300   7500   8000   9300  10000  11000  11000  13000  13000  13000   1397
GET      /finlife/save-deposit-products/                                                 2100   2100   2200   2200   2200   7200  12000  12000  12000  12000  12000     41
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
         Aggregated                                                                      4300   6300   7400   8000   9300  10000  11000  11000  13000  13000  13000   1438

Error report
# occurrences      Error
------------------|---------------------------------------------------------------------------------------------------------------------------------------------
38                 GET /finlife/save-deposit-products/: ConnectionRefusedError(10061, '[WinError 10061] 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.')
221                GET /finlife/deposit-products/: ConnectionRefusedError(10061, '[WinError 10061] 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.')
------------------|---------------------------------------------------------------------------------------------------------------------------------------------
```

RPS 관련 통계

응답 시간 관련 통계

에러 관련 내용

- 콘솔에서 Locust 종료 (Ctrl + C)
  - 위와 같이 전체 요청에 대한 분석을 콘솔에서 확인할 수 있습니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

정렬 알고리즘 테스트

테스트 주의사항

- 오늘 테스트는 정석적인 방법과는 거리가 있습니다.
  - 정석: 서버에 배포된 API 또는 프로그램에 부하 테스트를 해야 합니다.
- 하지만, 현재는 PC 에서 작동 중인 서버로 요청을 보내는 것
  - PC 의 성능에 따라 결과가 매우 달라집니다.
  - 현재 서버가 작동 중인 PC 에서 테스트를 진행하므로, 테스트 중 다른 조작을 하지 말아야 합니다!

정렬 알고리즘 구현하기

- 대상

  1. 파이썬 내장 정렬함수 - O(N logN)
  2. 버블 정렬 - O(n^2)
  3. 우선순위 큐 - 삽입: O(logN), 삭제: O(logN)

- 시나리오1. 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 의 배열을 만들어 가장 큰 값 찾기
- 시나리오2. (10배) 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 의 배열을 만들어 가장 큰 값 찾기

정렬 알고리즘 구현하기

- **가상 환경 설정**

```
# performence_test 폴더에서 진행
$ source venv/Scripts/activate
```

- **테스트용 Django 프로젝트 및 앱 생성**

```
(venv) $ django-admin startproject mypjt .
```

```
(venv) $ python manage.py startapp test
```

정렬 알고리즘 구현하기

- 각 정렬 알고리즘에 요청을 보낼 수 있도록 코드 작성

test/urls.py

```python
from django.urls import path
from . import views

urlpatterns = [
    path('normal_sort/', views.normal_sort),
    path('priority_queue/', views.priority_queue),
    path('bubble_sort/', views.bubble_sort),
]
```

test/views.py

```python
from django.http import JsonResponse
import random

def bubble_sort(request):
    li = []
    for i in range(1000):
        li.append(random.choice(range(1, 5000)))
    for i in range(len(li) - 1, 0, -1):
        for j in range(i):
            if li[j] < li[j + 1]:
                li[j], li[j + 1] = li[j + 1], li[j]
    context = {
      'top': li[0]
    }
    return JsonResponse(context)

# Create your views here.
def normal_sort(request):
    li = []
    for i in range(1000):
        li.append(random.choice(range(1, 5000)))
    li.sort(reverse=True)
    context = {
        'top': li[0]
    }
    return JsonResponse(context)

from queue import PriorityQueue

def priority_queue(request):
    pq = PriorityQueue()
    for i in range(1000):
        pq.put(-random.choice(range(1, 5000)))
    context = {
        'top': -pq.get()
    }
    return JsonResponse(context)
```

금융상품비교

영화추천서비스

정렬 알고리즘 구현하기

- **테스트 스크립트 작성하기**

locust_test.py

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

class SampleUser(HttpUser):
    wait_time = between(1, 3)

    def on_start(self):
        print('test start')

    @task
    def normal_sort(self):
        self.client.get("test/normal_sort/")

    @task
    def priority_queue(self):
        self.client.get("test/priority_queue/")

    @task
    def bubble_sort(self):
        self.client.get("test/bubble_sort/")
```

테스트 시나리오

1. 모든 Task 를 주석 처리 합니다.
   - **task 를 하나씩만 주석을 풀어 활성화 시킵니다.**
2. Locust 를 실행합니다.

```
(venv) $ locust -f locust_test.py
```

3. 결과를 웹에서 확인합니다.

   http://localhost:8089 접속

4. 테스트가 끝난 task 를 주석처리 후 다음에 테스트 할 task 주석을 풀어 활성화 시켜 줍니다.

5. 위 과정을 반복하며 결과를 확인합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

예시 - 시나리오 1

정렬 알고리즘 테스트 - 시나리오1

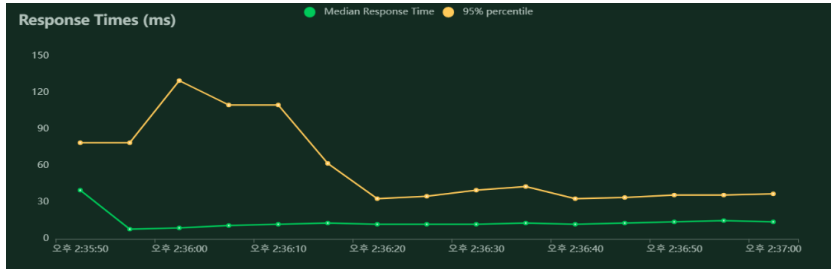
Confidential

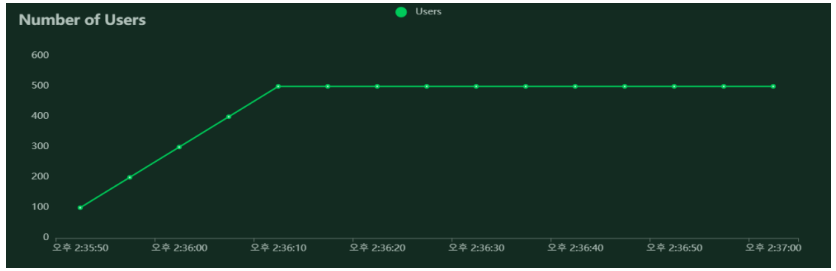
금융상품비교

영화추천서비스

테스트 결과 - Python Built-in Sort

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 20





Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/normal_sort/	16739	0	20	2	667	13	212.0	0.0
	Aggregated	**16739**	**0**	**20**	**2**	**667**	**13**	**212.0**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/normal_sort/	13	16	19	23	34	50	120	670
	Aggregated	**13**	**16**	**19**	**23**	**34**	**50**	**120**	**670**

- 평균 RPS: 212.0
- 응답 시간: 모든 응답이 0.6초 이내
- 시작 할 땐 병목이 잠깐 발생하지만 곧 해결됩니다.
- **결론: 사용자 수가 늘어날 때를 제외하고 매우 안정적이다.**

정렬 알고리즘 테스트 - 시나리오1

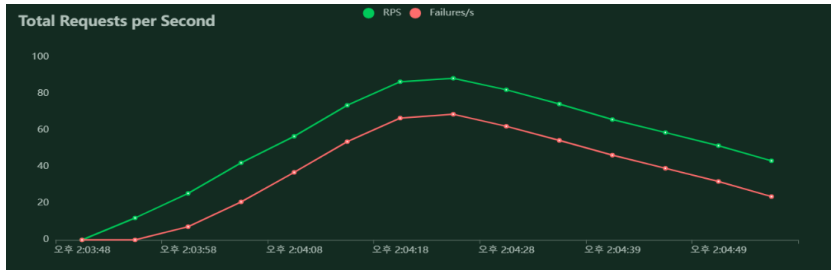
Confidential

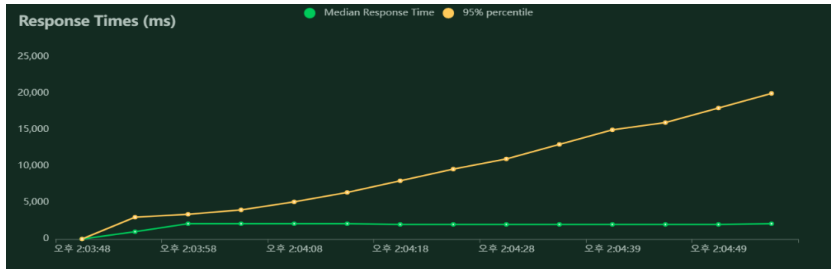
금융상품비교

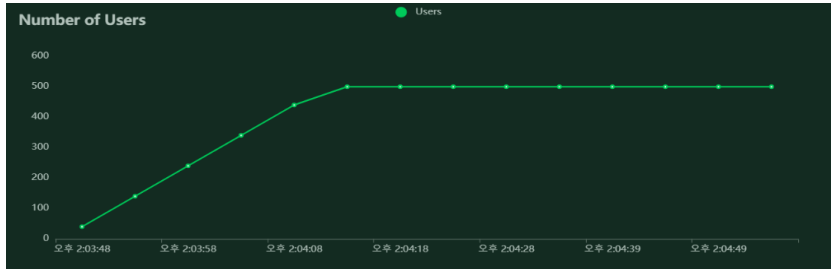
영화추천서비스

테스트 결과 - 버블 정렬

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 20







Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/bubble_sort/	3988	2665	3917	122	20908	4	59.3	39.7
	Aggregated	**3988**	**2665**	**3917**	**122**	**20908**	**4**	**59.3**	**39.7**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/bubble_sort/	2000	2100	2100	3800	11000	16000	19000	21000
	Aggregated	**2000**	**2100**	**2100**	**3800**	**11000**	**16000**	**19000**	**21000**

- 평균 RPS: 59.3
- 응답 시간: 중간: 0.2초 / 최대: 21초
- 응답 시간에서 중간 값과 95% percentile 의 차이가 커집니다.
- **결론: 요청이 늘어나면서 병목 현상이 바로 발생합니다.**
  - 이로 인해 실패가 점점 늘어납니다.

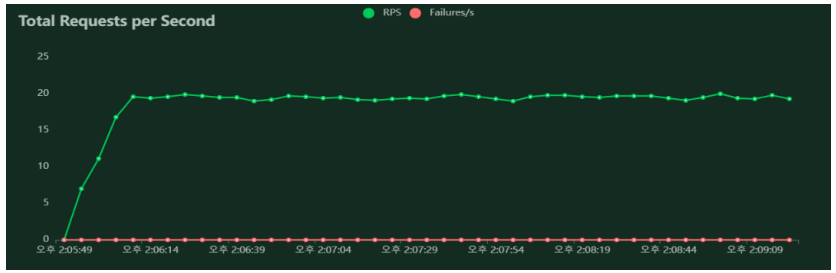
서버가 감당하기 힘든 알고리즘이다!

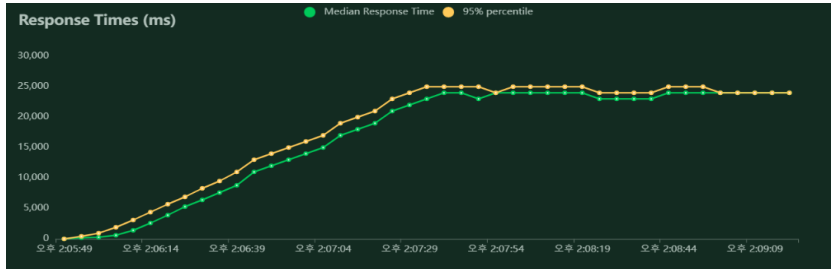
정렬 알고리즘 테스트 - 시나리오1

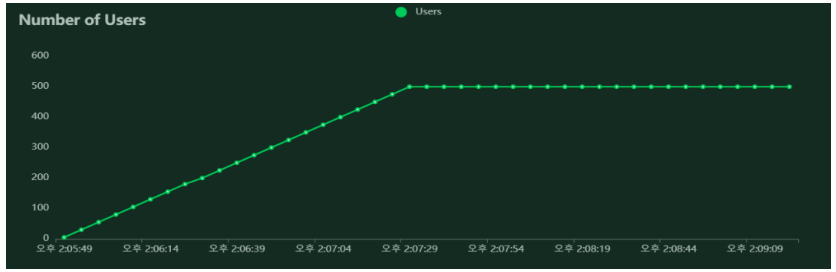
Confidential

테스트 결과 - 버블 정렬

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 5 (서버가 감당할 수준)







Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/bubble_sort/	4131	0	16644	50	24920	13	19.2	0.0
	Aggregated	**4131**	**0**	**16644**	**50**	**24920**	**13**	**19.2**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/bubble_sort/	23000	23000	23000	24000	24000	24000	25000	25000
	Aggregated	**23000**	**23000**	**23000**	**24000**	**24000**	**24000**	**25000**	**25000**

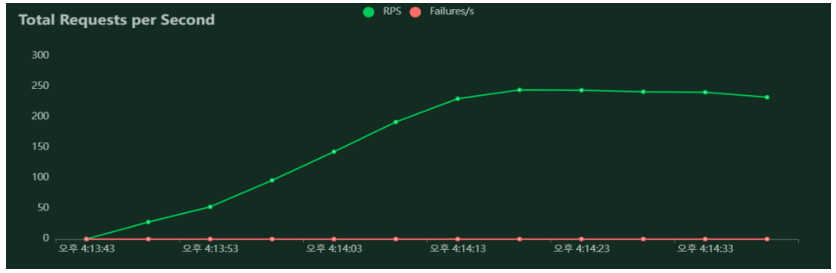
- 평균 RPS: 19.2
- 응답 시간: 모든 요청이 2.5초 이내
- 최고/최악일 때 응답 차이가 거의 없습니다.
- **결론: 유저 수와 응답 시간이 비슷하게 올라간다는 점과 RPS 나 응답 시간이 더 늘어나지 않고 일정하다는 점에서 결과는 이상적이나, 응답 시간(2.5초)이 오래 걸립니다.**

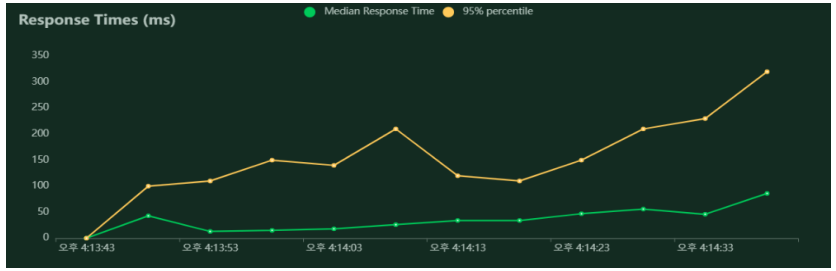
정렬 알고리즘 테스트 - 시나리오1

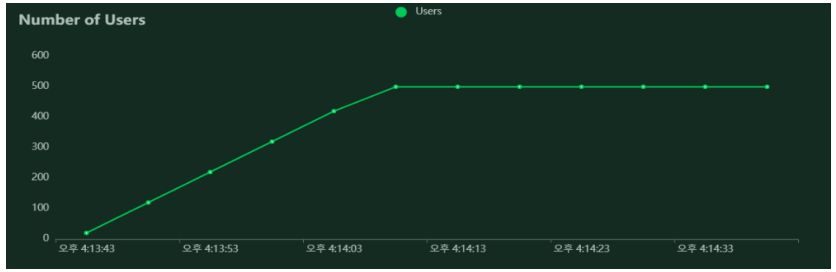
Confidential

테스트 결과 - 우선순위 큐

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 5 (서버가 감당할 수준)







Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/priority_queue/	25581	0	57	3	1116	13	219.5	0.0
	Aggregated	**25581**	**0**	**57**	**3**	**1116**	**13**	**219.5**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/priority_queue/	36	46	58	76	110	160	510	1100
	Aggregated	**36**	**46**	**58**	**76**	**110**	**160**	**510**	**1100**

- 평균 RPS: 219.5
- 응답 시간: 모든 응답이 1.1초 이내
- 응답 시간에서 중간값과 95% percentile 의 차이가 커집니다.
- **결론: 구현한 알고리즘에서 병목 현상이 발생하며,**
  언젠가 서버에 과부하가 온다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

예시 - 시나리오 2

정렬 알고리즘 테스트 - 시나리오2

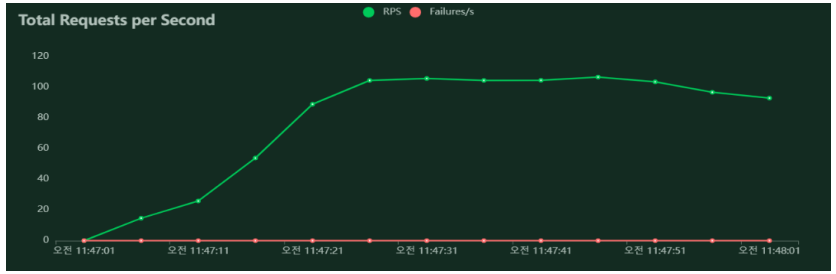
Confidential

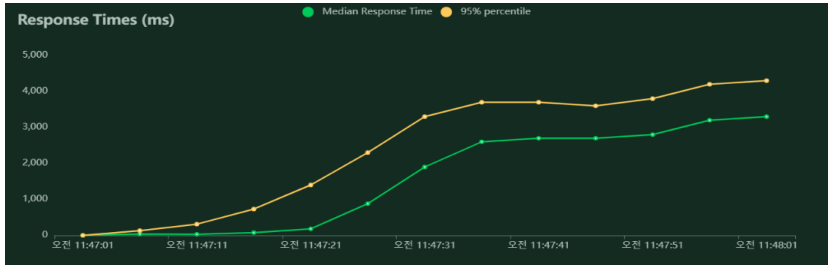
금융상품비교

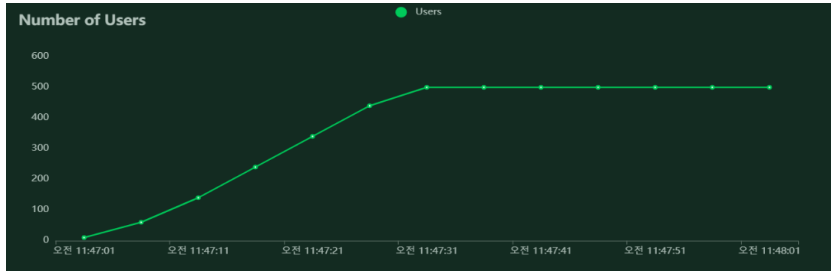
영화추천서비스

테스트 결과 - Python Built-in Sort

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20







Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/normal_sort/	5795	0	2119	9	8486	14	90.7	0.0
	Aggregated	**5795**	**0**	**2119**	**9**	**8486**	**14**	**90.7**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/normal_sort/	2400	2700	3000	3300	3600	3800	4300	8500
	Aggregated	**2400**	**2700**	**3000**	**3300**	**3600**	**3800**	**4300**	**8500**

- 평균 RPS: 90.7 (시나리오1: 212)
- 응답 시간: 중간: 0.2초 / 최대: 8.5초
- RPS 가 확연히 감소하였습니다.
- 응답 시간에서 중간 값과 95% percentile 의 차이가 변동이 없습니다.
- **결론: 서버의 응답 시간이 점점 늘어납니다.**

  즉, 갈수록 서버가 느려지는 현상(RPS 감소)가 발생할 것입니다.

테스트 결과 - 버블 정렬

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20
- 동시 사용자와 동시 접속자 수에 관계 없이 버블 정렬은 결과를 나타내지 못하였습니다.
  - 원인: 서버 시간 초과
- 응답에 너무 많은 시간이 걸리는 알고리즘은 테스트 불가능!

정렬 알고리즘 테스트 - 시나리오2

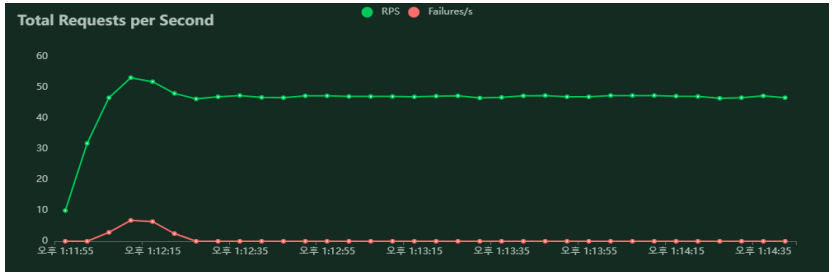
Confidential

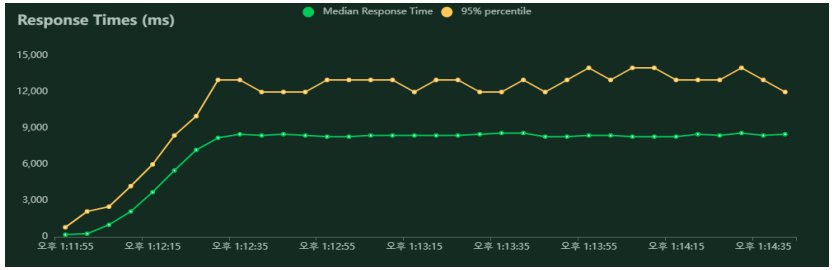
금융상품비교

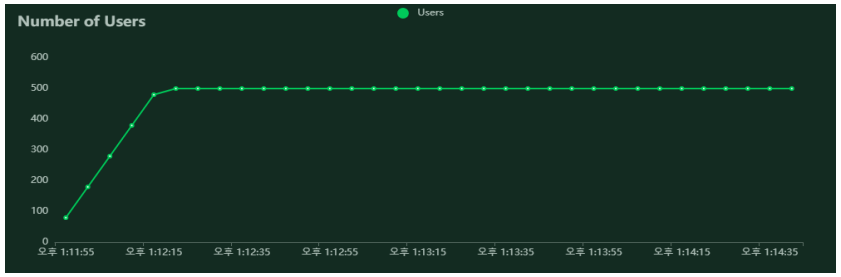
영화추천서비스

테스트 결과 - 우선순위 큐

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20







Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/priority_queue/	8065	93	7639	21	21536	13	47.0	0.5
	Aggregated	**8065**	**93**	**7639**	**21**	**21536**	**13**	**47.0**	**0.5**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/priority_queue/	8100	8500	9000	9700	11000	12000	15000	22000
	Aggregated	**8100**	**8500**	**9000**	**9700**	**11000**	**12000**	**15000**	**22000**

- 평균 RPS: 47.0 (시나리오1: 219.5)
- 응답 시간: 중간: 8.1초 / 최대: 15초
- RPS 가 확연히 감소하였습니다.
- 응답 시간에서 중간 값과 95% percentile 의 차이가 거의 일정합니다.
- **결론: 전체 응답 시간은 유지되는 것으로 보아**
  안정적인 것으로 보이지만,
  응답 시간 자체가 너무 느려 개선이 필요합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

테스트 결론

결론

- 직접 구현한 우선순위 큐보다 파이썬의 내장 함수가 안정적이고 빠릅니다.
  - 병목 현상이 발생하지 않음
  - 응답 시간이 최고/최악 모두 빠르다
- 알고리즘에 따라 서버 성능이 크게 좌우될 수 있습니다.
- 테스트 결과가 보여주는 내용은 작성한 결론 외에도 수 많은 정보를 내포하고 있습니다!
  - 추가적으로 많은 학습이 필요합니다.

테스트 결론

금융상품비교

영화추천서비스

요약

- 테스트 및 성능 테스트의 개념을 알아보고, Locust 를 활용한 부하 테스트를 해보았습니다.
- 스트레스 테스트 등의 다른 테스트는 명확한 목표를 정하고 진행해야 합니다.
  - 개선사항의 방향을 찾기 위해 테스트를 진행합니다.
  - 예시) 우리 서버는 반드시 0.8초 이내에 모든 응답을 주어야 한다.
- 여러 번 테스트를 해보아야 정확한 결과를 받을 수 있습니다.
  - 최대 부하 지점(임계점)의 부하를 지속하여 서버를 테스트한다 == 부하 테스트
  - 과부하가 오는 시점(중단점)을 찾아 지속적 혹은 반복적으로 서버를 테스트한다 == 스트레스 테스트

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

도전 과제

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

금융 상품 비교 앱
PJT 08

관통 Ver1 - PJT08 도전 과제

- 프로젝트명: 알고리즘 구현 및 성능 측정
- 목표
  - Django Rest Framework 를 활용하여 요구사항에 맞는 결과를 반환하도록 구성
  - 요구사항에 맞는 알고리즘 구현 및 성능 측정
- 특징
  - Pandas 라이브러리를 활용한 데이터 처리
  - Locust 를 활용한 알고리즘 성능 측정

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

영화 추천 서비스
PJT 08

관통 Ver2 - PJT08 도전 과제

- 프로젝트명: 비동기 통신을 이용한 웹 사이트 구현
- 목표
  - Django 에서 비동기 통신 활용 이해
  - 영화 추천 알고리즘 설계
- 특징
  - 제공된 영화 데이터를 DB에 저장하여 활용
  - 최종 프로젝트의 핵심인 추천 알고리즘을 생각해볼 수 있는 좋은 기회