Проєктування високонавантаженних систем. Лабораторна 1, звіт

Михайло Голуб

17 листопада 2025 р.

1 Завдання лабораторної роботи

1.1 Загальне

Порівняти throughput (пропускна здатність) Веб-застосунку в залежності від навантаження та способу зберігання даних (в пам'яті та БД). Створити Веб-застосунок який містить два ендпоінта (приймає два запити)

- /inc - інкрементує внутрішній каунтер
- /count - повертає значення каунтера

Наприклад, якщо

`http://localhost:8080/inc` був викликаний 10 разів, то

`http://localhost:8080/count` має повернути 10.

Наступним кроком необхідно створити HTTP-клієнта, який зможе робити задану кількість викликів до Веб-застосунку, а також заміряти час витрачений на здійснення цих викликів.

У якості клієнта може бути утиліта curl (чи будь-яка інша утиліта командного рядка), скрипт на Python (чи будь-якій іншій мові програмування)

У подальших завданнях треба буде в залежності від одночасної кількості клієнтів визначати яку кількість запитів в секунду обробляє Веб-застосунок. Це обраховується як - count/time, де time - кількість часу (секунд) яка була сумарна витрачена на здійснення всіх запитів

1.2 Частина 1

Завдання:

1. Один клієнт робить послідовно 10K викликів, кінцеве значення count = 10K - порахувати кількість запитів в секунду
2. Два клієнта роблять одночасно по 10K викликів кожен, кінцеве значення count = 20K - порахувати кількість запитів в секунду
3. 5 клієнтів роблять одночасно по 10K викликів кожен, кінцеве значення count = 50K - порахувати кількість запитів в секунду

4. 10 клієнтів роблять одночасно по 10K викликів кожен, кінцеве значення count = 100K - порахувати кількість запитів в секунду

Будьте уважні:

- Веб-застосунок має підртимувати багатопоточність (це не завжди так за замовченням)
- Клієнти мають запускатись паралельно та одночасно генерувати запити
- Каунтер на вашому Веб-застосунку має бути потоко-безпечним та забезпечувати відсутність lost-update

1.3 Частина 2

Перенести каунтер з оперативної пам'яті до БД (наприклад PostgreSQL). Тепер при кожному виклику /inc має інкрементуватись значення, яке зберігається у таблиці БД. При виклику /count кінцеве значення має зчитуватись з БД.

Зробить аналогічні вимірювання для 1) - 4) з попередної частини.

Порівняйте та проаналізуйте результати.

1.4 Вимоги до звіту та реалізації

- Мова реалізації будь-яка
- Має бути надано код програми/скрипта та результати виконання

2 Реалізація лічильника в оперативній пам'яті

2.1 Сервер

Сервер реалізовано на `Python` з використанням модуля `Flask`.

Оголошуються застосунок, лічильник та семафор (lock). Визначаються ендпоінти які через семафор змінюють або читають лічильник.

Сам застосунок є багатопотоковим, але однопроцессовим.

Повний код застосунку:

```
from flask import Flask
import threading

app = Flask(__name__)

counter = [0]
lock = threading.Lock()

@app.route("/inc")
def inc():
    global counter
```

```
    with lock:
        counter[0] += 1

@app.route("/count")
def count():
    with lock:
        return str(counter[0])
```

Повний код сервера:

```
from waitress import serve
from app import app

if __name__ == "__main__":
    serve(app, host="0.0.0.0", port=8080, threads=5)
```

2.2 Клієнт

Клієнт містить дві функції – `worker` та `main`. `worker` послідовно робить вказану кількість запитів і зупиняється. `main` запам'ятовує час початку, запускає необхідну кількість потоків які виконують `worker`, очікує на завершення цих потоків, робить запит на ендпоінт `/count`, і виводить в консоль отриманий `count` та `count/time`

Повний код:

```
import requests
import threading
import time

SERVER = "http://127.0.0.1:8080"
CALLS_PER_CLIENT = 10_000
NUM_CLIENTS = 1

def worker(client_id):
    for i in range(CALLS_PER_CLIENT):
        try:
            requests.get(f"{SERVER}/inc")
        except requests.exceptions.RequestException:
            pass

def main():
    print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
    calls each...")
    start = time.time()

    threads = []
    for i in range(NUM_CLIENTS):
        t = threading.Thread(target=worker, args=(i,))
        t.start()
        threads.append(t)

    for t in threads:
        t.join()

    end = time.time()
    elapsed = end - start
```

```
31
32      try:
33          final_count = int(requests.get(f"{SERVER}/count").text)
34      except:
35          final_count = -1
36
37      total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
38      throughput = total_calls / elapsed
39
40      print(f"Final count: {final_count}")
41      print(f"Total time:  {elapsed:.2f} s")
42      print(f"Throughput:  {throughput:.2f} requests/sec")
43
44  if __name__ == "__main__":
45      main()
46      input()
```

3 Запити з лічильником в оперативній пам'яті

Приклад результату роботи `client.py`:

```
Starting 1 clients x 10000 calls each...
Final count: 10000
Total time:  19.81 s
Throughput:  504.72 requests/sec
```

Результати роботи системи:

Клієнтів	Всього запитів	Лічильник	Час (s)	count/time (req/s)
1	10 000	10 000	11.18	894.58
2	20 000	20 000	15.88	1259.47
5	50 000	50 000	34.70	1440.96
10	100 000	98 010	209.64	477.00