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` і зберіганням значення лічильника в змінній.

Повний код застосунку:

```python
from flask import Flask
import threading

app = Flask(__name__)

counter = 0
lock = threading.Lock()

@app.route("/inc")
def inc():
    global counter
    with lock:
        c = counter
        counter = c + 1
```

```
        return str(counter)

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

Код що запускає сервер та застосунок на ньому:

```
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
import random

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

def worker(client_id):
    print(client_id, "worker started")
    for i in range(CALLS_PER_CLIENT):
        trying = True
        while trying:
            try:
                requests.get(f"{SERVER}/inc")
                trying = False
            except requests.exceptions.RequestException:
                print(client_id, "worker, request denied, i =",i)
                time.sleep(random.random())
    print(client_id, "worker end")

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)
```

```
32      print("MAIN: Threads created")
33      for t in threads:
34          t.join()
35      print("MAIN: Threads joined")
36      end = time.time()
37      elapsed = end - start
38      final_count = -1
39      i = 0
40      while final_count == -1 and i < 5:
41          try:
42              final_count = int(requests.get(f"{SERVER}/count").text)
43              print(final_count)
44          except:
45              final_count = -1
46              i+=1
47              time.sleep(1)
48
49      total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
50      throughput = total_calls / elapsed
51
52      print(f"Final count: {final_count}")
53      print(f"Total time:  {elapsed:.2f} s")
54      print(f"Throughput:  {throughput:.2f} requests/sec")
55
56  if __name__ == "__main__":
57      main()
58      input()
```

3 Запити з лічильником в оперативній пам'яті

Приклад результату роботи `client.py`:

```
Starting 10 clients x 10000 calls each...
0 worker started
1 worker started
2 worker started
3 worker started
4 worker started
5 worker started
6 worker started
7 worker started
8 worker started
9 worker started
MAIN: Threads created
7 worker, request denied, i = 6435
0 worker, request denied, i = 6588
7 worker, request denied, i = 6435
9 worker end
1 worker end
0 worker end
3 worker end
```

```
2 worker end
7 worker end
5 worker end
4 worker end
8 worker end
6 worker end
MAIN: Threads joined
100000
Final count: 100000
Total time:  173.60 s
Throughput:  576.02 requests/sec
```

Результати роботи системи з записом в оперативну пам'ять:

Клієнтів	Всього запитів	Лічильник	Час (s)	count/time (req/s)
1	10 000	10 000	11.81	846.85
2	20 000	20 000	23.35	856.70
5	50 000	50 000	67.23	743.67
10	100 000	100 000	173.60	576.02

4 Реалізація на PostgreSQL

В код застосунку додано функції під'єднання до бази даних та ініціалізації таблиці. Додано змінну яка показує застосунку де зберігати дані: на диску чи у оперативній пам'яті.

Повний код застосунку:

```python
from flask import Flask
import threading

app = Flask(__name__)
USE_DISK = True

DB_HOST = ***
DB_NAME = ***
DB_USER = ***
DB_PASS = ***
DB_PORT = 5432

if USE_DISK:
    def get_conn():
        return psycopg2.connect(
            host=DB_HOST,
            dbname=DB_NAME,
            user=DB_USER,
            password=DB_PASS,
            port=DB_PORT
        )
    def init_db():
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("""
```

```
            CREATE TABLE IF NOT EXISTS counter (
                id SERIAL PRIMARY KEY,
                value BIGINT NOT NULL
            );
        """)
        cur.execute("SELECT COUNT(*) FROM counter;")
        if cur.fetchone()[0] == 0:
            cur.execute("INSERT INTO counter (value) VALUES (0);")
        conn.commit()
        cur.close()
        conn.close()

    init_db()

    @app.route("/inc")
    def inc():
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("UPDATE counter SET value = value + 1 RETURNING value;")
        new_value = cur.fetchone()[0]
        conn.commit()
        cur.close()
        conn.close()
        return str(new_value)

    @app.route("/count")
    def count():
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("SELECT value FROM counter LIMIT 1;")
        current_value = cur.fetchone()[0]
        cur.close()
        conn.close()
        return str(current_value)
else:
    counter = 0
    lock = threading.Lock()

    @app.route("/inc")
    def inc():
        global counter
        with lock:
            c = counter
            counter = c + 1
            return str(counter)

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

5 Запити з лічильником на диску

Задля зменшення часу очікування (та зменшення шансів смерті старенького SSD) сумарна кількість запитів в усіх запусках рівна 10 000.

Клієнтів	Всього запитів	Лічильник	Час (s)	count/time (req/s)
1	10 000	10 000	392	25.48
2	10 000	10 000	238	42.00
5	10 000	10 000	115	86.50
10	10 000	10 000	115	86.33

6 Порівняння

Запис в базу даних в більш ніж 30 разів повільніший за оперативну пам'ять для одного клієнта, в 20 для двох, в 8.5 для п'яти і в 7 разів для десяти.

Згідно отриманих даних, можна зробити припущення, що лічильник в оперативній пам'яті впирається в швидкість роботи застосунку майже одразу, тож після 2 клієнтів, подальші зменшують швидкість обробки запитів.

Лічильник в базі даних впирається спочатку в швидкість формування запитів в БД на 1-2 клієнтах і впирається в швидкість запису самої БД на 5-10 клієнтах