Проектування високонавантажених систем

Лабораторна робота 1

Торгало Ігнатій ФБ-51мп

1. Лічильник у пам’яті

Було створено веб-застосунок на FastAPI, який зберігає лічильник у змінній RAM. Для потокобезпечності використано threading.Lock().

server_memory.py

```python
from fastapi import FastAPI
import uvicorn
import threading

app = FastAPI()

counter = 0
lock = threading.Lock()

@app.get("/inc")
def inc():
    global counter
    with lock:
        counter += 1
        return {"status": "ok", "counter": counter}

@app.get("/count")
def get_count():
    return {"counter": counter}

if __name__ == "__main__":
    uvicorn.run("server_memory:app", host="0.0.0.0", port=8080, workers=4)
```

Далі реалізовано клієнтський застосунок для генерації навантаження.

client.py

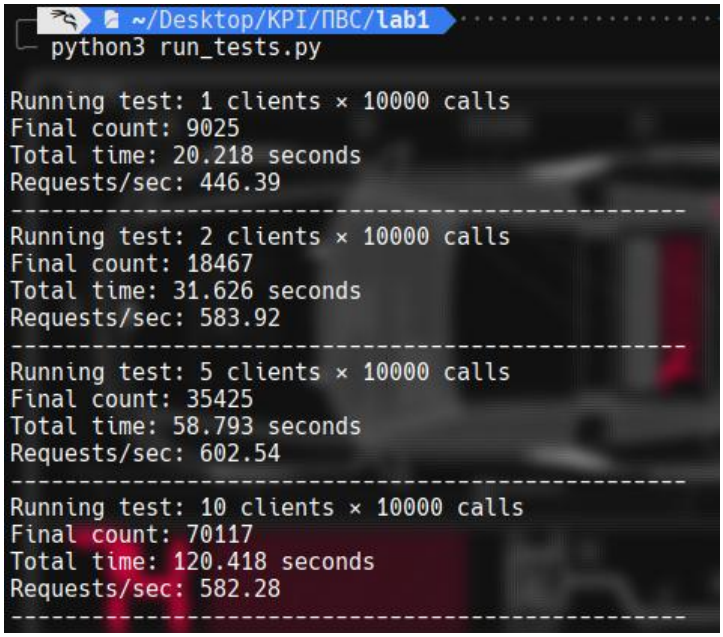
```python
import requests
import time
from concurrent.futures import ThreadPoolExecutor

SERVER = "http://localhost:8080"

def send_inc(n):
    for _ in range(n):
        requests.get(f"{SERVER}/inc")

def run_test(clients, calls_per_client=10000):
    print(f"Running test: {clients} clients × {calls_per_client} calls")

    start = time.time()

    with ThreadPoolExecutor(max_workers=clients) as pool:
        for _ in range(clients):
            pool.submit(send_inc, calls_per_client)

    end = time.time()
    total_time = end - start

    resp = requests.get(f"{SERVER}/count").json()["counter"]

    print(f"Final count: {resp}")
    print(f"Total time: {total_time:.3f} seconds")
    print(f"Requests/sec: {resp / total_time:.2f}")
    print("-" * 50)
```

Далі реалізовано код для тестування.

run_tests.py

```python
from client import run_test

run_test(1)

run_test(2)

run_test(5)

run_test(10)

```

Результати тестування (лічильник у пам’яті)



2. Лічильник у PostgreSQL

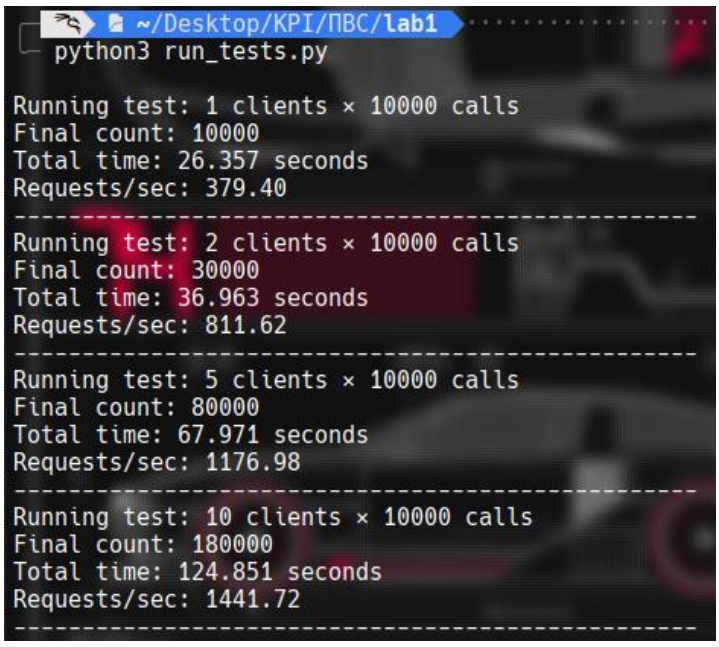
Було створено веб-застосунок на FastAPI, який зберігає лічильник у PostgreSQL.

server_db.py

```python
from fastapi import FastAPI
import uvicorn
import psycopg2
import threading

app = FastAPI()
lock = threading.Lock()

conn = psycopg2.connect(
    dbname="counterdb",
    user="counteruser",
    password="pass123",
    host="localhost",
    port=5432
)
conn.autocommit = True

@app.get("/inc")
def inc():
    with lock:
        cur = conn.cursor()
        cur.execute("UPDATE counter SET value = value + 1 WHERE id = 1;")
        cur.close()
    return {"status": "ok"}

@app.get("/count")
def get_count():
    cur = conn.cursor()
    cur.execute("SELECT value FROM counter WHERE id = 1;")
    result = cur.fetchone()[0]
    cur.close()
    return {"counter": result}

if __name__ == "__main__":
    uvicorn.run("server_db:app", host="0.0.0.0", port=8080, workers=4)
```

client.py та run_tests.py залишились без змін.

Результати тестування (лічильник у БД PostgreSQL)



3. Висновок

Під час тестування веб-застосунку з лічильником, що зберігається в пам’яті, було виявлено не лише обмежену продуктивність (близько 450–600 запитів за секунду), але й значні втрати інкрементів: фактичний результат виявився меншим за очікуваний на тисячі операцій. Це сталося тому, що сервер працював у кількох процесах, і кожен з них мав власний локальний лічильник, що призвело до розбіжностей і некоректної сумарної статистики.

На відміну від цього, реалізація лічильника у PostgreSQL показала нижчу швидкість для одного клієнта, проте значно кращу масштабованість — до 1400 запитів за секунду при 10 клієнтах. Головне — база даних забезпечила повну коректність: жоден інкремент не був втрачений.

Таким чином, зберігання в пам’яті підходить лише для простих, однопроцесних сценаріїв, тоді як PostgreSQL забезпечує набагато вищу надійність, узгодженість даних та стабільну роботу під паралельним навантаженням, що робить його кращим варіантом для реальних багатокористувацьких систем.