

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

```
1 from fastapi import FastAPI
2 import uvicorn
3 import threading
4
5 app = FastAPI()
6
7 counter = 0
8 lock = threading.Lock()
9
10 @app.get("/inc")
11 def inc():
12     global counter
13     with lock:
14         counter += 1
15         return {"status": "ok", "counter": counter}
16
17 @app.get("/count")
18 def get_count():
19     return {"counter": counter}
20
21 if __name__ == "__main__":
22     uvicorn.run("server_memory:app", host="0.0.0.0", port=8080, workers=4)
```

Далі реалізовано клієнтський застосунок для генерації навантаження.

client.py

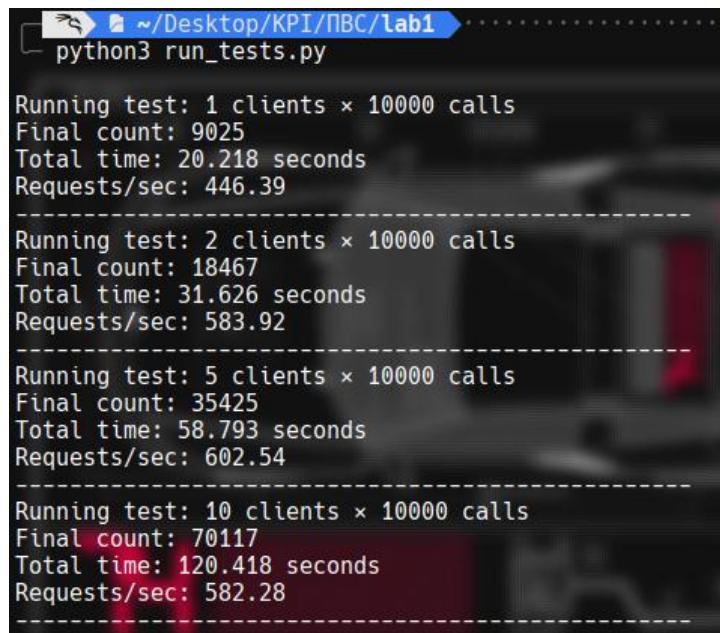
```
1 import requests
2 import time
3 from concurrent.futures import ThreadPoolExecutor
4
5 SERVER = "http://localhost:8080"
6
7 def send_inc(n):
8     for _ in range(n):
9         requests.get(f"{SERVER}/inc")
10
11 def run_test(clients, calls_per_client=10000):
12     print(f"Running test: {clients} clients x {calls_per_client} calls")
13
14     start = time.time()
15
16     with ThreadPoolExecutor(max_workers=clients) as pool:
17         for _ in range(clients):
18             pool.submit(send_inc, calls_per_client)
19
20     end = time.time()
21     total_time = end - start
22
23     resp = requests.get(f"{SERVER}/count").json()["counter"]
24
25     print(f"Final count: {resp}")
26     print(f"Total time: {total_time:.3f} seconds")
27     print(f"Requests/sec: {resp / total_time:.2f}")
28     print("-" * 50)
```

Далі реалізовано код для тестування.

run_tests.py

```
1 from client import run_test
2 |
3 run_test(1)
4
5 run_test(2)
6
7 run_test(5)
8
9 run_test(10)
10
```

Результати тестування (лічильник у пам'яті)



```
python3 run_tests.py

Running test: 1 clients × 10000 calls
Final count: 9025
Total time: 20.218 seconds
Requests/sec: 446.39

Running test: 2 clients × 10000 calls
Final count: 18467
Total time: 31.626 seconds
Requests/sec: 583.92

Running test: 5 clients × 10000 calls
Final count: 35425
Total time: 58.793 seconds
Requests/sec: 602.54

Running test: 10 clients × 10000 calls
Final count: 70117
Total time: 120.418 seconds
Requests/sec: 582.28
```

2. Лічильник у PostgreSQL

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

server_db.py

```

1 from fastapi import FastAPI
2 import uvicorn
3 import psycopg2
4 import threading
5
6 app = FastAPI()
7 lock = threading.Lock()
8
9 conn = psycopg2.connect(
10     dbname="counterdb",
11     user="counteruser",
12     password="pass123",
13     host="localhost",
14     port=5432
15 )
16 conn.autocommit = True
17
18 @app.get("/inc")
19 def inc():
20     with lock:
21         cur = conn.cursor()
22         cur.execute("UPDATE counter SET value = value + 1 WHERE id = 1;")
23         cur.close()
24     return {"status": "ok"}
25
26 @app.get("/count")
27 def get_count():
28     cur = conn.cursor()
29     cur.execute("SELECT value FROM counter WHERE id = 1;")
30     result = cur.fetchone()[0]
31     cur.close()
32     return {"counter": result}
33
34 if __name__ == "__main__":
35     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)

```

python3 run_tests.py

Running test: 1 clients × 10000 calls
Final count: 10000
Total time: 26.357 seconds
Requests/sec: 379.40
-----
Running test: 2 clients × 10000 calls
Final count: 30000
Total time: 36.963 seconds
Requests/sec: 811.62
-----
Running test: 5 clients × 10000 calls
Final count: 80000
Total time: 67.971 seconds
Requests/sec: 1176.98
-----
Running test: 10 clients × 10000 calls
Final count: 180000
Total time: 124.851 seconds
Requests/sec: 1441.72
-----
```

3. ВИСНОВОК

Під час тестування веб-застосунку з лічильником, що зберігається в пам'яті, було виявлено не лише обмежену продуктивність (блізько 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.