Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

Проектування високонавантажених систем
Лабораторна робота No1
Web-counter

Виконав:
Студент 4-го курсу
групи ФІ-21
Климентьєв Максим
Перевірив:
____

Зміст

1 Код реалізації

```
from threading import Lock
import sqlite3 as sql
import os

path_to_root = os.getcwd()
if "Lab1" not in path_to_root:
    path_to_root += "\\Lab1"
if "Data" not in path_to_root:
    path_to_root += "\\Data\\"


class Counter:
    def __init__(self):
        self._lock = Lock()

    def inc_count(self):
        raise NotImplementedError

    def get_count(self):
        raise NotImplementedError


class CounterMem(Counter):
    def __init__(self):
        super().__init__()
        self.value = 0
        # self._lock = Lock()

    def inc_count(self):
        with self._lock:
            self.value += 1
            return self.value

    def get_count(self):
        with self._lock:
            return self.value


class CounterDB(Counter):
    def __init__(self, db_path: str = path_to_root + "\\data.db"):
        super().__init__()
        self.db_path = db_path
        self.conn = sql.connect(db_path, check_same_thread=False)
        cur = self.conn.cursor()
        cur.execute("""
            CREATE TABLE IF NOT EXISTS counters (
                id INTEGER PRIMARY KEY,
                value INTEGER NOT NULL
            )
        """)
        cur.execute("INSERT OR IGNORE INTO counters (id, value) VALUES
   (1, 0)")
        cur.execute("UPDATE counters SET value = 0")
        self.conn.commit()

    def inc_count(self):
        with self._lock:
            cur = self.conn.cursor()
            cur.execute("UPDATE counters SET value = value + 1")
            self.conn.commit()
            cur.execute("SELECT value FROM counters")
            return cur.fetchone()[0]

    def get_count(self):
        with self._lock:
            cur = self.conn.cursor()
            cur.execute("SELECT value FROM counters")
            return cur.fetchone()[0]
```

Counter.py

```
from fastapi import FastAPI
import uvicorn

from Counter import CounterMem, CounterDB


class WebCounterApp:
    def __init__(self, counter: str = 'mem'):
        wrong_name = NameError("Wrong Name. Expected one from ['mem', '
   memory', 'db', 'database']")
        self.app = FastAPI(title="Web Counter")
        self.counter = CounterMem() if counter.lower() == 'mem' or
   counter.lower() == 'memory' else CounterDB() if counter.lower() == '
   db' or counter.lower() == 'database' else wrong_name
        if self.counter is wrong_name:
            raise self.counter
        self._setup_routes()

    def _setup_routes(self):
        @self.app.get("/inc")
        def inc_count():
            value = self.counter.inc_count()
            return value
        @self.app.get("/cnt")
        def get_count():
            value = self.counter.get_count()
            return value

    def run(self, host: str = "localhost", port: int = 8000, log_level:
   str = None):
        uvicorn.run(self.app, host=host, port=port, log_level=log_level
   )


if __name__ == "__main__":
    web_app = WebCounterApp('db')
    web_app.run()
```

WebCounter.py

```
import asyncio
import aiohttp
import time


class HttpClient:
    def __init__(self, base_url: str, concurrency: int = 100):
        self.base_url = base_url.rstrip('/')
        self.concurrency = concurrency

    async def _fetch(self, session: aiohttp.ClientSession, endpoint:
   str):
        async with session.get(f"{self.base_url}/{endpoint}") as resp:
            return await resp.text()

    async def _worker(self, session, endpoint, num_requests):
        tasks = [self._fetch(session, endpoint) for _ in range(
   num_requests)]
        await asyncio.gather(*tasks)

    async def run_load_test(self, endpoint: str, total_requests: int,
   name: str = "Memory"):
        start_time = time.perf_counter()
        async with aiohttp.ClientSession() as session:
            per_worker = total_requests // self.concurrency
            tasks = [
                self._worker(session, endpoint, per_worker)
                for _ in range(self.concurrency)
            ]
            await asyncio.gather(*tasks)

            final_value = await self._fetch(session, "cnt")
        duration = time.perf_counter() - start_time
        throughput = total_requests / duration
        print(f"[Name] {name}")
        print(f"[Clients|Requests] {per_worker} requests each for {self
   .concurrency} clients")
        print(f"[Sent] {total_requests} requests in {duration:.2f}
   seconds")
        print(f"[Count] {final_value}")
        print(f"[Throughput]: {throughput:.2f} requests/sec")
        print()
        return throughput


if __name__ == "__main__":
    client = HttpClient("http://localhost:8000", concurrency=200)
    asyncio.run(client.run_load_test("inc", total_requests=10_000))
```

Client.py

```
import time

import multiprocessing
import asyncio

from WebCounter import WebCounterApp
from Client import HttpClient

def run_server(counter_type, port):
    app = WebCounterApp(counter_type)
    app.run(port=port, log_level="critical")

if __name__ == "__main__":
    clients_amounts = [1, 2, 5, 10]
    requests_amount = 10_000


    for clients_amount in clients_amounts:
        total_requests = requests_amount * clients_amount
        # -------------------------------------------------- #

        p1 = multiprocessing.Process(target=run_server, args=('mem',
   8000))
        p1.start()
        time.sleep(1)

        client = HttpClient("http://localhost:8000", concurrency=
   clients_amount)
        asyncio.run(client.run_load_test("inc", total_requests=
   total_requests, name="Memory"))

        p1.terminate()
        p1.join()

        # -------------------------------------------------- #

        p2 = multiprocessing.Process(target=run_server, args=('db',
   8000))
        p2.start()

        time.sleep(1)


        client = HttpClient("http://localhost:8000", concurrency=
   clients_amount)
        asyncio.run(client.run_load_test("inc", total_requests=
   total_requests, name="DataBase"))

        p2.terminate()
        p2.join()
```

main.py

2 Результати

Один клієнт

```
[Name] Memory
[Clients|Requests] 10000 requests each for 1 clients
[Sent] 10000 requests in 3.29 seconds
[Count] 10000
[Throughput]: 3040.52 requests/sec

[Name] DataBase
[Clients|Requests] 10000 requests each for 1 clients
[Sent] 10000 requests in 10.75 seconds
[Count] 10000
[Throughput]: 929.95 requests/sec
```

Два клієнти

```
[Name] Memory
[Clients|Requests] 10000 requests each for 2 clients
[Sent] 20000 requests in 6.67 seconds
[Count] 20000
[Throughput]: 2996.28 requests/sec

[Name] DataBase
[Clients|Requests] 10000 requests each for 2 clients
[Sent] 20000 requests in 22.13 seconds
[Count] 20000
[Throughput]: 903.66 requests/sec
```

П’ять клієнтів

```
[Name] Memory
[Clients|Requests] 10000 requests each for 5 clients
[Sent] 50000 requests in 17.83 seconds
[Count] 50000
[Throughput]: 2804.70 requests/sec

[Name] DataBase
[Clients|Requests] 10000 requests each for 5 clients
[Sent] 50000 requests in 57.78 seconds
[Count] 50000
[Throughput]: 865.40 requests/sec
```

Десять клієнтів

```
[Name] Memory
[Clients|Requests] 10000 requests each for 10 clients
[Sent] 100000 requests in 39.65 seconds
[Count] 100000
[Throughput]: 2521.88 requests/sec

[Name] DataBase
[Clients|Requests] 10000 requests each for 10 clients
[Sent] 100000 requests in 117.59 seconds
[Count] 100000
[Throughput]: 850.39 requests/sec
```