

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

Проектування високонавантажених систем

Лабораторна робота №2
MassiveCounterIncrement

Виконав:

Студент 4-го курсу
групи ФІ-21
Климент'єв Максим
Перевірив:

Зміст

1	Код реалізації	3
2	Результати	8

1 Код реалізації

PostgreUpdates.py

```
import time
from Updates import LostUpdate, OptimisticConcurrency, InplaceUpdate, RowLevelLocking, SerializableUpdate

if __name__ == "__main__":
    database_params = {
        "host": "database-lab3",
        "port": 5432,
        "dbname": "user_counter_db",
        "user": "postgres",
        "password": "postgres"
    }

    user_id = 1
    threads = 10
    increments_per_thread = 10_000
    expected = threads * increments_per_thread

    variants = {
        "Lost-update": LostUpdate,
        "Serializable update": SerializableUpdate,
        "In-place update": InplaceUpdate,
        "Row-level locking": RowLevelLocking,
        "Optimistic concurrency control": OptimisticConcurrency,
    }

    for name, variant in variants.items():
        print(f"[INFO] Running: {name}")
        # print(f"[INFO] Connecting to PostgreSQL...")
        var = variant(user_id, threads, increments_per_thread, database_params, serializable=(name == "Serializable update"))
        # print(f"[INFO] Starting time evaluation...")
        start_total = time.perf_counter()
        final = var.run()
        total_time = time.perf_counter() - start_total
        percent = (final / expected) * 100
        print(f"[RESULTS] Received: {final} ({percent:.2f}% from expected {expected})")
        print(f"[RESULTS] Elapsed: {total_time:.2f} sec")
        print("-" * 60)
```

BaseUpdate.py

```
import time
from concurrent.futures import ThreadPoolExecutor, as_completed

from Updates.Utilities import Utilities

class BaseUpdate:
    def __init__(self, user_id: int, threads: int, increments: int, database_params: str, serializable: bool = False):
        self.user_id = user_id
        self.threads = threads
        self.increments = increments
        self.utilities = Utilities(database_params, serializable)
        self.utilities.ensure_table
        self.utilities.ensure_user

    def run_threads(self, target):
        futures = []
        # start = time.perf_counter()
        with ThreadPoolExecutor(max_workers=self.threads) as ex:
            for t in range(self.threads):
                futures.append(ex.submit(target, t))
            for f in as_completed(futures):
                exc = f.exception()
                if exc:
                    raise exc
        # elapsed = time.perf_counter() - start
        # return elapsed

    # @property
    def run(self):
```

```

        self.utilities.reset_counter(self.user_id)
        # elapsed = self.run_threads(self.worker)
        self.run_threads(self.worker)
        final = self.utilities.read_counter(self.user_id)[0]
        return final

```

Utilities.py

```

import psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_READ_COMMITTED, ISOLATION_LEVEL_SERIALIZABLE

class Utilities:
    def __init__(self, database_params: dict, serializable: bool = False):
        self.database_params = database_params
        self.serializable = serializable

    @property
    def new_connection(self):
        connection = psycopg2.connect(**self.database_params)
        connection.autocommit = False
        if self.serializable:
            connection.set_isolation_level(ISOLATION_LEVEL_SERIALIZABLE)
        else:
            connection.set_isolation_level(ISOLATION_LEVEL_READ_COMMITTED)
        return connection

    @property
    def ensure_table(self):
        with self.new_connection as connection:
            with connection.cursor() as cursor:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS user_counter (
                        user_id INTEGER PRIMARY KEY,
                        counter INTEGER NOT NULL DEFAULT 0,
                        version INTEGER NOT NULL DEFAULT 0
                    );
                """)
                connection.commit()

    # @property
    def ensure_user(self, user_id: int):
        with self.new_connection as connection:
            with connection.cursor() as cursor:
                cursor.execute("""
                    INSERT INTO user_counter(user_id, counter, version)
                    VALUES (%s, 0, 0)
                    ON CONFLICT (user_id) DO NOTHING
                """, (user_id,))
                connection.commit()

    # @property
    def reset_counter(self, user_id: int):
        connection = self.new_connection
        try:
            cursor = connection.cursor()
            cursor.execute("UPDATE user_counter SET counter = 0, version = 0 WHERE user_id = %s", (user_id,))
            connection.commit()
        finally:
            connection.close()

    # @property
    def read_counter(self, user_id: int):
        connection = self.new_connection
        try:
            cursor = connection.cursor()
            cursor.execute("SELECT counter, version FROM user_counter WHERE user_id = %s", (user_id,))
            result = cursor.fetchone()
            return result if result else (0, 0)
        finally:
            connection.close()

```

LostUpdate.py

```

from Updates.BaseUpdate import BaseUpdate

class LostUpdate(BaseUpdate):

```

```

def worker(self, thread_id):
    connection = self.utilities.new_connection
    cursor = connection.cursor()
    for _ in range(self.increments): # for (i in 1..10_000) {
        cursor.execute("SELECT counter FROM user_counter WHERE user_id = %s", (self.user_id,)) # counter = cursor.
    execute("SELECT counter FROM user_counter WHERE user_id = "1).fetchone()
    results = cursor.fetchone()
    v = results[0]
    v += 1 # counter = counter + 1
    cursor.execute("UPDATE user_counter SET counter = %s WHERE user_id = %s", (v, self.user_id)) # cursor.
    execute("update user_counter set counter = %s where user_id = %s", (counter, 1))
    connection.commit() # conn.commit()
    connection.close()

```

SerializableUpdate.py

```

from Updates.BaseUpdate import BaseUpdate
# import time

class SerializableUpdate(BaseUpdate):
    def worker(self, thread_id):
        connection = self.utilities.new_connection
        cursor = connection.cursor()
        for _ in range(self.increments):
            while True:
                try:
                    cursor.execute("SELECT counter FROM user_counter WHERE user_id = %s", (self.user_id,))
                    results = cursor.fetchone()
                    v = results[0]
                    v += 1
                    cursor.execute("UPDATE user_counter SET counter = %s WHERE user_id = %s", (v, self.user_id))
                    connection.commit()
                    break
                except Exception as e:
                    connection.rollback()
                    # time.sleep(1e-3)
        connection.close()

```

InplaceUpdate.py

```

from Updates.BaseUpdate import BaseUpdate

class InplaceUpdate(BaseUpdate):
    def worker(self, thread_id):
        connection = self.utilities.new_connection
        cursor = connection.cursor()
        for i in range(self.increments):
            cursor.execute("UPDATE user_counter SET counter = counter + 1 WHERE user_id = %s", (self.user_id,))
            connection.commit()
        connection.close()

```

RowLevelLocking.py

```

from Updates.BaseUpdate import BaseUpdate

class RowLevelLocking(BaseUpdate):
    def worker(self, thread_id):
        connection = self.utilities.new_connection
        cursor = connection.cursor()
        for i in range(self.increments):
            cursor.execute("SELECT counter FROM user_counter WHERE user_id = %s FOR UPDATE", (self.user_id,))
            results = cursor.fetchone()
            v = results[0]
            v += 1
            cursor.execute("UPDATE user_counter SET counter = %s WHERE user_id = %s", (v, self.user_id))
            connection.commit()
            # raise
        connection.close()

```

OptimisticConcurrency.py

```
from Updates.BaseUpdate import BaseUpdate
```

```

# import time

class OptimisticConcurrency(BaseUpdate):
    def worker(self, thread_id):
        connection = self.utilities.new_connection
        cursor = connection.cursor()
        for i in range(self.increments): # for (i in 1..10_000) {
            while True: # while (True) {
                cursor.execute("SELECT counter, version FROM user_counter WHERE user_id = %s", (self.user_id,)) # (
                counter, version) = cursor.execute("SELECT counter, version FROM user_counter WHERE user_id = "1").fetchone()
                results = cursor.fetchone()
                counter, old_version = results
                counter += 1
                new_version = old_version + 1
                cursor.execute("UPDATE user_counter SET counter = %s, version = %s WHERE user_id = %s AND version = %s",
                (counter, new_version, self.user_id, old_version))
                connection.commit()
                if cursor.rowcount > 0: # count = cursor.rowcount
                    break # if (count > 0) break
            connection.close()

```

Dockerfile

```

FROM python:3.13
WORKDIR /app
COPY req.txt .
RUN pip install --no-cache-dir -r req.txt
COPY ./Updates ./Updates
COPY PostgreUpdates.py .

CMD ["python", "PostgreUpdates.py"]

```

docker-compose.yml

```

services:
  database-lab3:
    image: postgres:16
    container_name: postgres-lab3
    ports:
      - "5432:5432"
    volumes:
      - postgre-data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: user_counter_db
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 0.5s
      retries: 30

  python-client:
    build: .
    container_name: python-client
    depends_on:
      database-lab3:
        condition: service_healthy
    environment:

```

```
PYTHONUNBUFFERED: 1
```

```
volumes:
```

```
  postgre-data:
```

2 Результаты

```
[INFO] Running: Lost-update
[RESULTS] Received: 10001 (10.00% from expected 100000)
[RESULTS] Elapsed: 141.15 sec
-----
[INFO] Running: Serializable update
[RESULTS] Received: 100000 (100.00% from expected 100000)
[RESULTS] Elapsed: 241.51 sec
-----
[INFO] Running: In-place update
[RESULTS] Received: 100000 (100.00% from expected 100000)
[RESULTS] Elapsed: 224.35 sec
-----
[INFO] Running: Row-level locking
[RESULTS] Received: 100000 (100.00% from expected 100000)
[RESULTS] Elapsed: 239.39 sec
-----
[INFO] Running: Optimistic concurrency control
[RESULTS] Received: 100000 (100.00% from expected 100000)
[RESULTS] Elapsed: 1544.97 sec
```