

Проектування високонавантажених систем.

Лабораторна 3, звіт

Михайло Голуб

8 грудня 2025 р.

1 Завдання лабораторної роботи

1.1 Загальне

Необхідно декількома способами реалізувати оновлення значення каунтера в СКБД PostgreSQL та оцінити час кожного із варіантів.

Таблиця `user_counter` з колонками `USER_ID`, `Counter`, `Version`.

1. Lost-update
2. Serializable update
3. In-place update
4. Row-level locking
5. Optimistic concurrency control

1.2 Вимоги до звіту та реалізації

- Мова реалізації будь-яка
- Не використовувати ORM-фреймворки (Hibernate, SQLAlchemy, ...)
- Не забувати про необхідність окремої транзакції на кожен запис

2 Хід роботи

2.1 Реалізація клієнта

Лістинг 1: `lost_update_or_serialize.py`

```
1 import threading
2 import time
3 import random
4 from tqdm import tqdm
```

```

5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 SERIALISE = True
19
20 print_lock = threading.Lock()
21
22 def get_conn():
23     if SERIALISE:
24         conn = psycopg2.connect(
25             host=DB_HOST,
26             dbname=DB_NAME,
27             user=DB_USER,
28             password=DB_PASS,
29             port=DB_PORT,
30         )
31         conn.set_isolation_level(extensions.ISOLATION_LEVEL_SERIALIZABLE)
32         return conn
33     else:
34         return psycopg2.connect(
35             host=DB_HOST,
36             dbname=DB_NAME,
37             user=DB_USER,
38             password=DB_PASS,
39             port=DB_PORT
40         )
41 def init_db():
42     conn = get_conn()
43     cur = conn.cursor()
44     cur.execute("""
45         CREATE TABLE IF NOT EXISTS user_counter (
46             USER_ID SERIAL PRIMARY KEY,
47             Counter BIGINT NOT NULL,
48             Version BIGINT NOT NULL
49         );
50     """)
51     # Reset counter to 0
52     cur.execute("DELETE FROM user_counter;")
53     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
54     conn.commit()
55     cur.close()
56     conn.close()
57
58 def worker(client_id):
59     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
60               str(client_id), leave=False)

```

```

60     conn = get_conn()
61     cur = conn.cursor()
62     for i in bar:
63
64         cur.execute("SELECT counter FROM user_counter WHERE user_id
65 = 1")
66         counter = cur.fetchone()[0]
67         counter += 1
68         cur.execute("UPDATE user_counter SET counter = %s WHERE
69 user_id = %s", (counter, 1))
70         conn.commit()
71     cur.close()
72     conn.close()
73     with print_lock:
74         bar.close()
75
76 def main():
77     if SERIALISE:
78         print("Serialise")
79     else:
80         print("Lost-upddate")
81     print("Initialising DB")
82     init_db()
83     time.sleep(1)
84
85     conn = get_conn()
86     cur = conn.cursor()
87     print("DB initialised.")
88     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
89 1;")
90     print("Counter state:", cur.fetchone()[0])
91     conn.commit()
92     cur.close()
93     conn.close()
94     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
95 calls each...")
96
97     start = time.time()
98
99     threads = []
100    for i in range(NUM_CLIENTS):
101        t = threading.Thread(target=worker, args=(i,))
102        t.start()
103        threads.append(t)
104    print("MAIN: Threads created")
105    for t in threads:
106        t.join()
107    print("MAIN: Threads joined")
108    end = time.time()
109    elapsed = end - start
110    final_count = -1
111    i = 0
112    conn = get_conn()
113    cur = conn.cursor()
114    print("DB initialised.")
115    cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =

```

```

113     1;")
114     final_count = cur.fetchone()[0]
115     conn.commit()
116     cur.close()
117     conn.close()
118
119     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
120     throughput = total_calls / elapsed
121
122     print(f"Final count: {final_count}")
123     print(f"Total time: {elapsed:.2f} s")
124     print(f"Throughput: {throughput:.2f} requests/sec")
125
126 if __name__ == "__main__":
127     main()
128     input()

```

Приклад логу під час роботи клієнта:

```

Lost-update
Initialising DB
DB initialised.
Counter state: 0
Starting 10 clients x 10000 calls each...
MAIN: Threads created
0: 0%| 42/10000 [00:03<10:55, 15.20it/s]
1: 0%| 44/10000 [00:03<11:07, 14.92it/s]
2: 0%| 44/10000 [00:03<11:21, 14.62it/s]
3: 0%| 44/10000 [00:03<10:48, 15.35it/s]
4: 0%| 40/10000 [00:02<11:04, 14.98it/s]
5: 0%| 43/10000 [00:03<14:04, 11.79it/s]
6: 0%| 44/10000 [00:03<11:08, 14.90it/s]
7: 0%| 42/10000 [00:03<12:22, 13.41it/s]
8: 0%| 43/10000 [00:03<12:32, 13.22it/s]
9: 0%| 46/10000 [00:03<11:12, 14.79it/s]

```

Приклад логу після завершення роботи клієнта:

```

Lost-update
Initialising DB
DB initialised.
Counter state: 0
Starting 10 clients x 10000 calls each...
MAIN: Threads created
MAIN: Threads joined
DB initialised.
Final count: 10661
Total time: 15.21 s
Throughput: 6573.22 requests/sec

```

2.2 Lost-update

Лістинг об'єднаного lost-update та Serializable (тип роботи обирається змінною) клієнта був наведений вище.

Результати тестування:
Час роботи: 15.21 секунди
Пропускна здатність: 6573 запитів/с
Кінцеве значення лічильника: 10661

2.3 Serializable

При запуску таких самих запитів, але з рівнем ізоляції SERIALIZABLE, програма видає наступну помилку на всіх воркерах, окрім одного:

```
cur.execute("UPDATE user_counter SET counter = %s WHERE user_id = %s", (counter, 1))
psycopg2.errors.SerializationFailure:
ПОМИЛКА:  не вдалося серіалізувати доступ через паралельне оновлення
```

Результати тестування:
Час роботи: 3 секунди
Пропускна здатність: — запитів/с
Кінцеве значення лічильника: 10к
З певного моменту, лише один клієнт робив інкрементацію лічильника.
Таку помилкову поведінку можна виправити використавши In-place update або робити повторні запити у разі помилки.

2.4 Serializable з повтором

Якщо додати до Serializable повторний запит до таблиці у разі отримання помилки серіалізації, можна отримати щось що схоже на оптимістичне блокування: якщо все ок — підтвердити операцію; якщо не вдалося серіалізувати операції — повернути помилку, клієнти очікують випадковий час і пробують зробити операцію знову

Лістинг 2: serialize_and_repeat.py

```
1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
```

```

15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     conn = psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT,
27     )
28     conn.set_isolation_level(extensions.ISOLATION_LEVEL_SERIALIZABLE)
29     return conn
30
31 def init_db():
32     conn = get_conn()
33     cur = conn.cursor()
34     cur.execute("""
35         CREATE TABLE IF NOT EXISTS user_counter (
36             USER_ID SERIAL PRIMARY KEY,
37             Counter BIGINT NOT NULL,
38             Version BIGINT NOT NULL
39         );
40     """)
41     # Reset counter to 0
42     cur.execute("DELETE FROM user_counter;")
43     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
44     conn.commit()
45     cur.close()
46     conn.close()
47
48 def worker(client_id):
49     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
50               str(client_id), leave=False)
51     for i in bar:
52         conn = get_conn()
53         cur = conn.cursor()
54         while True:
55             try:
56                 cur.execute("SELECT counter FROM user_counter WHERE
57                 user_id = 1")
58                 counter = cur.fetchone()[0]
59                 counter += 1
60                 cur.execute("UPDATE user_counter SET counter = %s
61                 WHERE user_id = %s", (counter, 1))
62                 conn.commit()
63
64                 break
65             except:
66                 try:
67                     cur.close()
68                     conn.close()
69                 except:
70                     pass

```

```

68             time.sleep(random.uniform(0.01, 0.05))
69             conn = get_conn()
70             cur = conn.cursor()
71             cur.close()
72             conn.close()
73             with print_lock:
74                 bar.close()

75
76     def main():
77         print("Serialise and repeat")
78         print("Initialising DB")
79         init_db()
80         time.sleep(1)

81         conn = get_conn()
82         cur = conn.cursor()
83         print("DB initialised.")
84         cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
85 1;")
86         print("Counter state:", cur.fetchone()[0])
87         conn.commit()
88         cur.close()
89         conn.close()
90         print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
91 calls each...")

92
93         start = time.time()

94
95         threads = []
96         for i in range(NUM_CLIENTS):
97             t = threading.Thread(target=worker, args=(i,))
98             t.start()
99             threads.append(t)
100        print("MAIN: Threads created")
101        for t in threads:
102            t.join()
103        print("MAIN: Threads joined")
104        end = time.time()
105        elapsed = end - start
106        final_count = -1
107        i = 0
108        conn = get_conn()
109        cur = conn.cursor()
110        print("DB initialised.")
111        cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
112 1;")
113        final_count = cur.fetchone()[0]
114        conn.commit()
115        cur.close()
116        conn.close()

117        total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
118        throughput = total_calls / elapsed

119
120        print(f"Final count: {final_count}")
121        print(f"Total time: {elapsed:.2f} s")

```

```

122     print(f"Throughput: {throughput:.2f} requests/sec")
123
124 if __name__ == "__main__":
125     main()
126     input()

```

Результати тестування:
 Час роботи: 860 секунд
 Пропускна здатність: 116.15 запитів/с
 Кінцеве значення лічильника: 100к

2.5 In-place update

Лістинг 3: Inplace.py

```

1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28 def init_db():
29     conn = get_conn()
30     cur = conn.cursor()
31     cur.execute("""
32         CREATE TABLE IF NOT EXISTS user_counter (
33             USER_ID SERIAL PRIMARY KEY,
34             Counter BIGINT NOT NULL,
35             Version BIGINT NOT NULL
36         );
37     """)
38     # Reset counter to 0
39     cur.execute("DELETE FROM user_counter;")
40     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
41     conn.commit()

```

```

42     cur.close()
43     conn.close()
44
45 def worker(client_id):
46     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
47               str(client_id), leave=False)
48     conn = get_conn()
49     cur = conn.cursor()
50     for i in bar:
51
52         cur.execute("UPDATE user_counter SET Counter = Counter + 1
53 WHERE USER_ID = %s", (1,))
54         conn.commit()
55     cur.close()
56     conn.close()
57     with print_lock:
58         bar.close()
59
60 def main():
61     print("In-place update")
62     print("Initialising DB")
63     init_db()
64     time.sleep(1)
65
66     conn = get_conn()
67     cur = conn.cursor()
68     print("DB initialised.")
69     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
70 1;")
71     print("Counter state:", cur.fetchone()[0])
72     conn.commit()
73     cur.close()
74     conn.close()
75     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
76 calls each...")
77
78     start = time.time()
79
80     threads = []
81     for i in range(NUM_CLIENTS):
82         t = threading.Thread(target=worker, args=(i,))
83         t.start()
84         threads.append(t)
85     print("MAIN: Threads created")
86     for t in threads:
87         t.join()
88     print("MAIN: Threads joined")
89     end = time.time()
90     elapsed = end - start
91     final_count = -1
92     i = 0
93     conn = get_conn()
94     cur = conn.cursor()
95     print("DB initialised.")
96     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
97 1;")
```

```

94     final_count = cur.fetchone()[0]
95     conn.commit()
96     cur.close()
97     conn.close()
98
99     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
100    throughput = total_calls / elapsed
101
102    print(f"Final count: {final_count}")
103    print(f"Total time: {elapsed:.2f} s")
104    print(f"Throughput: {throughput:.2f} requests/sec")
105
106 if __name__ == "__main__":
107     main()
108     input()

```

Результати тестування:

Час роботи: 12.56 секунд

Пропускна здатність: 7963.53 запитів/с

Кінцеве значення лічильника: 100к

2.6 Row-level locking

Лістинг 4: Rowlevel.py

```

1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28 def init_db():
29     conn = get_conn()
30     cur = conn.cursor()
31     cur.execute("""
32         CREATE TABLE IF NOT EXISTS user_counter (

```

```

33         USER_ID SERIAL PRIMARY KEY,
34         Counter BIGINT NOT NULL,
35         Version BIGINT NOT NULL
36     );
37 """
38 # Reset counter to 0
39 cur.execute("DELETE FROM user_counter;")
40 cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
41 conn.commit()
42 cur.close()
43 conn.close()

44 def worker(client_id):
45     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
46     str(client_id), leave=False)
47     conn = get_conn()
48     cur = conn.cursor()
49     for i in bar:
50         cur.execute("SELECT Counter FROM user_counter WHERE USER_ID
51 = 1 FOR UPDATE")
52         counter = cur.fetchone()
53         counter = counter[0] + 1
54         cur.execute("UPDATE user_counter SET Counter = %s WHERE
55 USER_ID = %s", (counter, 1))
56         conn.commit()
57     cur.close()
58     conn.close()
59     with print_lock:
60         bar.close()

61 def main():
62     print("Row-level locking")
63     print("Initialising DB")
64     init_db()
65     time.sleep(1)

66     conn = get_conn()
67     cur = conn.cursor()
68     print("DB initialised.")
69     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
70 1;")
71     print("Counter state:", cur.fetchone()[0])
72     conn.commit()
73     cur.close()
74     conn.close()
75     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
76 calls each...")

77     start = time.time()

78     threads = []
79     for i in range(NUM_CLIENTS):
80         t = threading.Thread(target=worker, args=(i,))
81         t.start()
82         threads.append(t)
83     print("MAIN: Threads created")

```

```

85     for t in threads:
86         t.join()
87     print("MAIN: Threads joined")
88     end = time.time()
89     elapsed = end - start
90     final_count = -1
91     i = 0
92     conn = get_conn()
93     cur = conn.cursor()
94     print("DB initialised.")
95     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID = "
96                 "1;")
97     final_count = cur.fetchone()[0]
98     conn.commit()
99     cur.close()
100    conn.close()
101
102    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
103    throughput = total_calls / elapsed
104
105    print(f"Final count: {final_count}")
106    print(f"Total time: {elapsed:.2f} s")
107    print(f"Throughput: {throughput:.2f} requests/sec")
108
109 if __name__ == "__main__":
110     main()
111     input()

```

Результати тестування:

Час роботи: 20.06 секунд

Пропускна здатність: 4985.25 запитів/с

Кінцеве значення лічильника: 100к

2.7 Optimistic concurrency control

Листинг 5: Optimistic.py

```

1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():

```

```

21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28 def init_db():
29     conn = get_conn()
30     cur = conn.cursor()
31     cur.execute("""
32         CREATE TABLE IF NOT EXISTS user_counter (
33             USER_ID SERIAL PRIMARY KEY,
34             Counter BIGINT NOT NULL,
35             Version BIGINT NOT NULL
36         );
37     """)
38     # Reset counter to 0
39     cur.execute("DELETE FROM user_counter;")
40     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
41     conn.commit()
42     cur.close()
43     conn.close()
44
45 def worker(client_id):
46     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
47     str(client_id), leave=False)
48     conn = get_conn()
49     cur = conn.cursor()
50     for i in bar:
51         while True:
52             cur.execute("SELECT Counter, Version FROM user_counter
53             WHERE USER_ID = 1")
54             counter, version = cur.fetchone()
55             counter = counter + 1
56             cur.execute("update user_counter set counter = %s,
57             version = %s where user_id = %s and version = %s",
58             (counter, version + 1, 1, version))
59             conn.commit()
60             count = cur.rowcount
61             if (count > 0): break
62     cur.close()
63     conn.close()
64     with print_lock:
65         bar.close()
66
67 def main():
68     print("Row-level locking")
69     print("Initialising DB")
70     init_db()
71     time.sleep(1)
72
73     conn = get_conn()
74     cur = conn.cursor()
75     print("DB initialised.")
76     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
77     1;")
```

```

74     print("Counter state:", cur.fetchone()[0])
75     conn.commit()
76     cur.close()
77     conn.close()
78     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
79     calls each...")
80
81     start = time.time()
82
83     threads = []
84     for i in range(NUM_CLIENTS):
85         t = threading.Thread(target=worker, args=(i,))
86         t.start()
87         threads.append(t)
88     print("MAIN: Threads created")
89     for t in threads:
90         t.join()
91     print("MAIN: Threads joined")
92     end = time.time()
93     elapsed = end - start
94     final_count = -1
95     i = 0
96     conn = get_conn()
97     cur = conn.cursor()
98     print("DB initialised.")
99     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
100    1;")
101    final_count = cur.fetchone()[0]
102    conn.commit()
103    cur.close()
104    conn.close()
105
106    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
107    throughput = total_calls / elapsed
108
109    print(f"Final count: {final_count}")
110    print(f"Total time: {elapsed:.2f} s")
111    print(f"Throughput: {throughput:.2f} requests/sec")
112 if __name__ == "__main__":
113     main()
114     input()

```

Результати тестування:

Час роботи: 88 секунд

Пропускна здатність: 1136.31 запитів/с

Кінцеве значення лічильника: 100к

3 Результати

Варіант	Час роботи	Пропускна здатність	Значення лічильника
Lost-update	15.21	6573	10661
Serializable	3	—	10k
Ser. з повтором	860	116.15	100k
Inplace update	12.56	7963	100k
Row-level locking	20.06	4985	100k
Optimistic concurrency control	88	1136.31	100k