Assignment 6

# Concurrency Tuning

Database Tuning

New Group 8

Frauenschuh Florian, 12109584

Lindner Peter, 12101607

Weilert Alexander, 12119653

June 16, 2024

## Notes

- You will need to run transactions concurrently using threads in Java. See `https://dbresearch.uni-salzburg.at/teaching/2020ss/dbt/account.zip` for an example.

## Experimental Setup

For our experiments we used the following hardware and software:

| Component | Specs |
|---|---|
| Processor | i7-13700H 3.7-5.0 GHz |
| Memory | 32 GiB |

Table 1: Hardware: Dell XPS 15 9530

| Software | Version |
|---|---|
| OS | Ubuntu 22.04 |
| Postgres | 2.3.4 |
| postgresql | 42.7.3 |
| MariaDB | 10.6.16 |
| mariadb-java-client | 3.3.3 |
| Java | 18 |

Table 2: Software

Before each test run we created the table `accounts` by

```
CREATE TABLE IF NOT EXISTS accounts (
  account INT,
  balance INT
);
```

and inserted the initial data by using batch statements. The isolation levels were set by executing

```
SET TRANSACTION ISOLATION LEVEL (READ COMMITTED | SERIALIZABLE);
```

## Solution (a)

**Read Committed** Throughput and correctness for solution (a) with isolation level `READ COMMITTED`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 714.28 | 1.0 |
| 2 | 1123.59 | 0.87 |
| 3 | 1176.47 | 0.89 |
| 4 | 1020.4 | 0.87 |
| 5 | 1315.79 | 0.93 |

**Serializable** Throughput and correctness for solution (a) with isolation level `SERIALIZABLE`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 632.91 | 1.0 |
| 2 | 917.43 | 0.87 |
| 3 | 1123,59 | 0.98 |
| 4 | 1111.11 | 0.89 |
| 5 | 1265.82 | 0.96 |

## Solution (b)

**Read Committed** Throughput and correctness for solution (b) with isolation level `READ COMMITTED`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 917.43 | 1.0 |
| 2 | 1219.51 | 1.0 |
| 3 | 1315.79 | 1.0 |
| 4 | 1333.33 | 1.0 |
| 5 | 1408.45 | 1.0 |

**Serializable** Throughput and correctness for solution (b) with isolation level `SERIALIZABLE`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 892.85 | 1.0 |
| 2 | 1219.51 | 1.0 |
| 3 | 1298.7 | 1.0 |
| 4 | 1298.7 | 1.0 |
| 5 | 1351.35 | 1.0 |

## Discussion

**Solution (a)** For solution (a), the performance and correctness varied slightly between the two isolation levels: `READ COMMITTED` and `SERIALIZABLE`.

**Read Committed:** The `READ COMMITTED` isolation level is the default in PostgreSQL [1] and ensures that any data read during a transaction is committed at the moment it is read. This level prevents dirty reads, which means no transaction can read data that another transaction has written but not yet committed. However, `READ COMMITTED` does not protect against non-repeatable reads or phantom reads.

In our experiments, as the number of concurrent transactions increased, the throughput also increased. This increase in throughput is expected as more transactions are processed simultaneously. However, the correctness decreased with increased concurrency. This decline is due to the fact that `READ COMMITTED` allows other transactions to modify data between reads within the same transaction, leading to anomalies like non-repeatable reads and phantom reads. These anomalies contribute to inconsistencies, hence the drop in correctness.

**Serializable:** The `SERIALIZABLE` isolation level provides the highest level of isolation by ensuring that transactions are executed in a way that produces the same result as if they were run serially, one after another. This level eliminates anomalies such as dirty reads, non-repeatable reads, and phantom reads, thus maintaining data consistency and correctness.

Under the `SERIALIZABLE` isolation level, the throughput in most cases was slightly lower compared to `READ COMMITTED`. However, the correctness remained high across all levels of concurrency. This stability is due to the stringent measures in place to maintain serializability, even though it may involve higher overhead due to potential transaction rollbacks and retries to resolve conflicts.

**Solution (b)** For solution (b), where each transaction consists of two update statements, the performance and correctness were high across both isolation levels.

**Read Committed:** In this scenario, `READ COMMITTED` performed really well. The throughput consistently increased with the number of concurrent transactions. The correctness remained at 100% across all levels of concurrency. This high performance and correctness are due to the simplicity of the transactions, which involve only a couple of updates. Hence, the likelihood of encountering non-repeatable reads or phantom reads is minimized.

**Serializable:** The `SERIALIZABLE` isolation level also showed high performance and correctness. The throughput was slightly lower than `READ COMMITTED`. Again, the correctness remained at 100% across all concurrency levels.

**General Discussion:** In PostgreSQL, isolation levels are essential for maintaining data consistency and integrity. The results indicate that for simple transactions involving a small number of updates, both `READ COMMITTED` and `SERIALIZABLE` can achieve high throughput and correctness. However, for more complex transactions, `SERIALIZABLE` provides higher correctness at the cost of some throughput, whereas `READ COMMITTED` offers higher throughput but lower correctness.

## Time Spent on this Assignment

Time in hours per person:

- Florian Frauenschuh: **2**
- Peter Lindner: **4**
- Alexander Weilert: **1.5**

## References

[1] PostgreSQL Global Development Group. *SET TRANSACTION*. `https://www.postgresql.org/docs/current/sql-set-transaction.html`. Accessed: 2024-06-16. 2023.