Value of Serializability

INSTRUCTIONS

1. Submit one ZIP file, `<student number>.zip` (for example: "`A012345L.zip`"), containing the following files to the folder "Serializability Submissions" in Luminus Files by **Friday 19 April 2019 at 18:30**.
   - Your report (PDF file).
   - Six python source code files named `db_connect.py`, `create_account.py`, `run_sums.py`, `run_exchanges.py`, `run_all_exchanges.py`, and `run_experiments.py`.
2. Make sure to write your **name** and **student number** on the front page of the report.
3. There is no option for late submissions for this project.
4. Do not modify any of the template files unless otherwise indicated.
5. Structure your report according to the questions and parts. Use clear headings for the questions and parts, e.g. `Question 1` and `Question 1.b`.
6. Make sure to clearly highlight the answer to the question from other discussions and comments.

The goal of this project is to run an experiment that illustrates the value of serializability in PostgreSQL. The database contains a single `account` table, with a `balance` as one of its columns. Two types of transactions are executed concurrently: `Sum`, which calculates the sum of all account balances, and `Exchange`, which exchanges the balances of two arbitrary accounts. The experiment consists in running those transactions in parallel at different isolation levels and assess both correctness and performance. You should read and understand how the transaction isolation level in PostgreSQL works[1].

Download the zip file "`project5.zip`" from Luminus files to answer the questions. Use Python 2.7 and SQLAlchemy library[2] to answer the questions.

[1] `https://www.postgresql.org/docs/10/sql-set-transaction.html`
[2] `https://docs.sqlalchemy.org/en/latest/core/tutorial.html`

The file `db_connect.py` has a `get_conn` function that returns a connection to the PostgreSQL database server. The file `create_account.py` creates an `account` table that stores an account number (integer) as primary key, a branch number (integer) and a balance (float) by using the connection from the `db_connect.py` file. It populates the `account` table with 100,000 records where the account number is numbered from 1 to 100,000, the branch number is randomized in the [1, 20] interval and the balance is randomized uniformly in the [0, 100,000) interval rounded to 2 decimal places. Run the PostgreSQL database in the LAPP stack and run the `create_account.py` file.

Question 1 [5 marks]

The `Sum` transaction calculates the sum of all `balances` over the `account` table.

The file `run_sums.py` is a Python code that takes S and I as inputs, where S is the number of sums and I is the isolation level (`'read uncommitted'`, `'read committed'`, `'repeatable read'`, `'serializable'`). The function `sum_balance` takes a session as an input and execute a query that calculates the sum of all `balances` over the `account` table. You should fill this function using the SQLAlchemy library. Please refer to the SQLAlchemy documentation.

The function `S_sums` runs the `sum_balance` function S times in sequence with isolation level I and returns all sum values upon completion. This function has been done for you.

The code prints all sum values upon completion. Submit the Python file, namely `run_sums.py`, with the filled `sum_balance` function to answer this question.

Question 2 [5 marks]

The `Exchange` transaction swaps the balance of two accounts. More specifically, it proceeds in four steps:

1. it reads the balance from a first account A_1 (picked at random) into a variable V_1,
2. it reads the balance from a second account A_2 (picked at random) into a variable V_2,
3. it writes the value V_1 as the new balance of the account A_2,
4. it writes the value V_2 as the new balance of the account A_1.

Using `run_sums.py` as a template, write a similar Python code that takes E and I as inputs, where E is the number of exchanges in a subprocess and I is the isolation level (`'read uncommitted'`, `'read committed'`, `'repeatable read'`, `'serializable'`).

Create a function `exchange` that takes a session as an input and execute the `Exchange` transaction in four steps defined previously. This function is similar to the `sum_balance` function in `run_sums.py`. You should fill this function using the SQLAlchemy library. Please refer to the SQLAlchemy documentation.

Create a function `E_swaps` that runs the `swap` function E times in sequence, with isolation level I and a 0.1 ms pause in-between transactions (use the `time.sleep` function). This function is similar to the `S_sums` function in `run_sums.py`.

The code measures and prints the execution time. Submit a Python file, namely `run_exchanges.py`, function to answer this question.

Question 3 [10 marks]

The file `run_all_exchanges.py` takes E, P and I as inputs, where E is the number of exchanges in a subprocess, P is the number of subprocesses and I is the isolation level (`'read uncommitted'`, `'read committed'`, `'repeatable read'`, `'serializable'`). The code runs P `run_exchanges.py` subprocesses simultaneously. The code measures and returns the average execution time of all subprocesses. This file and function has been done for you.

The file `run_all_experiments.py` takes S, E, P and I as inputs, where S is the number of sums, E is the number of exchanges in a subprocess, P is the number of subprocesses and I is the isolation level (`'read uncommitted'`, `'read committed'`, `'repeatable read'`, `'serializable'`). It calculates the sum of all `balances` over the `account` table once. This value will be called the *true sum*. It runs the `run_sums.py` and `run_all_exchanges.py` codes concurrently with their respective parameters. After their execution finishes, it calculates the `correctness` which is the number of correct sums from the output of `run_sums.py` with respect to the *true sum* divided by S (number of sums). The threshold for the differences is set to 0.01. This file and function has been done for you.

Set the variables for the number of sums, $S = 100$, the total number of exchanges, $E \times P = 2000$ for the following experiments. The total number of exchanges, $E \times P$, must remain constant over all experiments. Every time you change the number of subprocesses, P, you must adjust the number of exchanges, E. Vary P from 1 to 100 in the incremental of 10. For each value of P, repeat the experiment 20 times. Report the mean values (over the 20 experiments) of average execution time (over P subprocesses) and the mean values (over the same 20 experiments) of the correctness in the following 2 plots (each plot has 4 graphs). Notice that for each experiments you must measure the execution time for each process and the overall correctness at the same time.

(a) Plot the mean average execution time as a function of the total number of subprocesses P for each of the 4 isolation levels I. (5)

(b) Plot the mean average correctness as a function of the total number of subprocesses P for each of the 4 isolation levels I. (5)

Comment on these graphs in your report: describe what you observe, formulate hypotheses that explain your observations and check them with the documentation.

– END OF PAPER –