CS 2550 – PRINCIPLES OF DATABASE SYSTEMS (SPRING 2020)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Term Project: *myPTA* (Phase 2)

Release: Mar. 26, 2020 Due: 8PM, Thursday, Apr. 23, 2020.

Goal

The objective of the project is to develop *myPTA* (my Pitt Transaction Agent), a transactional row store database that efficiently supports concurrent execution of OLTP (i.e., transactions) workloads. *myPTA* will provide limited *transactional* support. By limited support, we mean that it does not support full durability. In addition to serializable and atomic access, it will also provide the standard uncontrolled access to files.

Description

myPTA consists of three components: *Data Manager, Scheduler,* and *Transaction Manager*. The *Data Manager* is responsible for managing the data on the primary and secondary storage. The *Scheduler* is responsible for scheduling operations from transactions. The *Transaction Manager* is responsible for reading in transaction operations. In the first phase you have already implemented Data Manager. In the second phase (i.e., this assignment) you are required to implement both Scheduler and Transaction Manager.

Scheduler and Transaction Manager

Script files should be executed either as "transactions" or as "processes." Transactions access files in an *atomic mode* whereas processes access files in a *normal mode*. *myPTA* supports concurrent access to files by both transactions and processes by employing the *Strict Two-Phase Locking* protocol to ensure serializability for transactions.

myPTA uses *wait-for graphs* for deadlock detection and is free from livelocks. It ensures transactions' atomicity by adopting an undo recovery strategy where all before images are kept in the buffer and they are written to the file on the disk at commit time. Transactions aborted by the system due to deadlocks should be ignored. You are not required to implement restarts.

Design

Your design should include the data structures (such as lock tables) and methods adopted, for example, to handle initializations/configuration, deadlocks and transaction failures, i.e., to obliterate the effects of aborted transactions. As mentioned above, *myPTA* does not support "durability", thereby you should assume system failure free environment.

In order to illustrate the functionality of your prototype *myPTA*, you will be required to execute concurrently a number of application programs that operate on the shared data file in various modes (atomic or normal). All such programs have the same structure specified in script files. A script file consists of a series of file operations, one per line, introduced by a *Begin* program primitive associated with the execution mode that the file operations are supposed to execute. An operation series ends with either a *Commit* or *Abort* program primitive. In the case of processes (normal mode), Commit and Abort primitives behave alike, terminating the execution of the process. A script file may contain multiple such sequences of file operations.

Here is the list of commands:

B EMode : Begin (Start) a new transaction (EMode=1) or new process (EMode=0).

C : Commit (End) current transaction (process).

A : Abort (End) current transaction (process).

R, M, W, E, and D : already implemented and should operate as specified in Phase 1 of the project.

An example of a script file is:

```
-----------------------------------------------
     B 1
     R X 13
     W X (2, Thalia, 412-656-2212)
     R X 2
     M X 412
     C
-----------------------------------------------
```

Statistics

Furthermore, at the end of the execution of a set of applications programs, the statistics of the execution should be displayed. These statistics will be the *number of committed transactions*, the *percentage of all involved operations* and the *average response time*.

Data Consistency

All transactions have to operate only on committed data. No stale data is allowed. In case of transactions, the isolation level should be *Serializable* and in case of processes, it should be *Read Committed* (where *Non-repeatable reads* are possible).

You need to build a compatibility table for all available operations to drive your scheduler. The compatibility table should take into account both transactions and processes. Multi-granularity locking should be implemented in the following way: operations R, W, and E should lock at the tuple level based on the primary key, operation M should lock at the tuple level based on the area code, and operation D should lock at the file level.

Summary

In summary, you will implement two modules: Transaction Manager and Scheduler. The Transaction Manager is responsible of reading commands from different program files concurrently and pass the commands to the Scheduler. You need to implement two methods of concurrent reading from program files: a Round Robin (which reads one line from each file at a time in turns) and a Random (which reads from files in random order, and reads a random number of lines from each file). You should allow the user to specify multiple program files at start time, the buffer size for the Data Manager as well as the seed of the random number generator. The Scheduler implements the multi-granularity lock manager and deadlock detector.

Testing

You need to conduct two types of testing: *Normal* and *Benchmark*. Normal tests are used to verify the correctness of the modules. Benchmark tests are used to stress test the modules and evaluate the efficiency. The tests should mainly focus on the below functionality:

- Concurrency Control:
  - Normal Tests: Test the correctness of your concurrency control functionality between all conflicting operation pairs under different execution modes.
  - Benchmark Tests: Create a benchmark test for each normal test. Each benchmark test should have at least 10 times the transaction count and at least 100 times the conflicting operation pair count as those of the corresponding normal test.
- Deadlock Detection:
  - Normal Tests: Test the correctness of your deadlock detection functionality between all conflicting operation pairs.
  - Benchmark Tests: Create a benchmark test for each normal test. Each benchmark test should have at least 10 times the deadlock loop length and at least 10 times the deadlock loop count as those of the corresponding normal test.
- Recovery:
  - Normal Tests: Test the correctness of your recovery functionality for all relevant operations.
  - Benchmark Tests: Create a benchmark test for each normal test. Each benchmark test should have at least 100 times the operation count as that of the corresponding normal test.
- Others
  - Normal Tests: Any other tests you find necessary to demonstrate your work in the project.
  - Benchmark Tests: Create a benchmark test for each normal test. Each benchmark test should have an appropriate multiple of the workload as that of the corresponding normal test.

The test result summary for each test needs to be output to a test log file. Please keep the test result summaries concise.

Documentation

You need to write a detailed documentation about your design and implementation of the Scheduler and Transaction Manager for *myPTA*. The documentation should cover at least the following aspects:

- For the Scheduler:
  - Detailed description of the design and implementation of the concurrency control strategies under different execution modes, including the compatiblility table, the locking mechnisums for each operation, and so on.

  - Detailed description of the design and implementation of the deadlock detection and handling mechanisms.
  - Detailed description of the design and implementation of the recovery mechanisms under transaction abortion.

- For the Transaction Manager:

  - Detailed description of the design and implementation of the operation reading mechanisms.

- For the Testing:

  - For each normal test case, state clearly the transaction setup, the expected output with reasons, and the actually output. If the actual output is different from the expected output, analyze the reason.
  - For each benchmark test case, state clearly the transaction setup, the execution result (e.g., if the program crashed), and the efficiency measures in throughput.

- Others

  - Detailed description of the contribution of each team member based on the implemented components. The granularity of the components should be as fine as possible.

Submission Guidelines

- The project code will be collected by the TA via the GitHub repository that you share with only your TA (GitHub ID: xzhangedu) and your team members. To turn in your code, you must do three things by the deadline:

  1. Make a commit to your project repository that represents what should be graded as your group's submission for this phase. The message for this commit should be "Phase 2 submission".
  2. Push that commit to the GitHub repository that you have shared with the TA. Please make sure that the commit is pushed before the due date.
  3. Prepare a link to your GitHub repository, the commit ID (Hash) of the commit to be graded and a suitable title for the submission - i.e. 'Project Phase 2 Submission'. Send this information into an email to cs2550-staff@cs.pitt.edu. Submit the same information into a document, named "Project Phase 2 Submission" via the assignment submission web interface. Commit ID can be found on GitHub or as part of output of "gitlog" command.

- The following items are required to be in the GitHub repository to be collected:

  - Your source code (You have the option to implement your prototype *myPTA* in any language and on Windows or Unix-based operating system).
  - The documentation file.
  - Your test input script files for all documented experiments. Please submit a file generator instead of the actual script files for the benchmark tests.
  - A readme file about how to generate the benchmark test scripts, compile the code, run the program, and where to find the test log file.

- You may submit Phase 2 multiple times. The last commit before the due date (**8PM, Thursday, Apr. 23, 2020**) will be graded.

Grading

The grading of the project phase 2 is based on the following aspects:

- The successful compilation and running of the source code.
- The quality of the documentation.
- The comprehensiveness and completeness of the test cases.
- The efficiency of the program. The most efficient myPTA will earn 5% bonus points.

Academic Honesty

The work in this project is to be done *independently* by each group. Discussions with other groups on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.