CS 2550 – PRINCIPLES OF DATABASE SYSTEMS (SPRING 2017)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Term Project: *myPTA*

Release: March 18, 2017 Due: 11:59 PM, April 3 (Design), April 27 (Protype), 2017

Goal

The goal of this project is to provide limited *transactional* support for a small Row store database system. By limited support we mean that it does not support durability. Specifically, the objective is to develop the concurrency control module (*myPTA*: my Pitt Transaction Agent) which, in addition to the standard uncontrolled access to files, it also provides both serializable and atomic access.

Description

myPTA will consist of three components: Data Manager, Scheduler, and Transaction Manager. *myPTA* will take script files as the input and each script files will 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 a hybrid concurrent control (HCC) scheme. HCC adaptively combines both the *Strict Two-Phase Locking* (2PL) and the *Optimistic Concurrency Control* (OCC) to ensure serializability for transactions. *myPTA* should be able to automatically switch between these two approaches based on the observed workload/access patterns (e.g., your system should use OCC for an analytical workload, and switch to 2PL for transactional/write heavy workload). *Each group should suggest a strategy that optimizes the usage between 2PL and OCC*. In case of OCC, if a transaction attempts to commit modifications to data that has changed since the transaction began or its read data were changed since the transaction began, the transaction will roll back and an error will be raised. *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.

For this project, all records (tuples) have the following fields (schema):

- ID: 4-byte integer (Primary Key)
- ClientName: 16-byte long string
- Phone: 12-byte long string

In order to illustrate the functionality of your prototype *myPTA*, you will be required to execute concurrently a number of application programs 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 operations, one per line, introduced by a *Begin* program primitive associated with the execution mode that 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 operations.

Here is the list of command lines operations:

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 table val : Retrieve the record with ID=val in table. If the record does not exist, it returns -1. If table does not exist, the read is aborted.

M table val : Retrieve the record(s) which have val as area code in Phone attribute in table. If the record does not exist, it returns -1. If table does not exist, the read is aborted.

W table (t) : Write the record t into table. If table does not exist, it is created.

D table : Delete table.

All operations should be called from a script file.

An example of a script file (assuming we have table X) is:

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

Disk Organization

Records are kept on persistent storage (disk) in *directed files* in a *row store* fashion, meaning that each table is kept in a separate file. Each file consists of slotted pages of size 2048 bytes each. Every inserted tuple is hashed based on its primary key using hash function h(x) = x mod 16. Conflicts are resolved with chaining by appending new slotted pages at the end of the file as necessary.

Main Memory Organization

The records will have to be brought to main memory (database buffer) to be accessed. However, the number of buffer pages available in database buffer is limited and should be specified at the beginning of each execution. You are required to implement Least Recently Used (LRU) page replacement mechanism to swap pages in and out.

All updates made to the database buffer should be propagated to the disk when corresponding pages from database buffer get swapped out, or when the script execution completes.

All meta-data and control structures (such as page tables) in main memory that are needed for efficient processing are kept outside of the database buffer. As opposed to the database buffer, there is always sufficient space for meta-data and control structures.

In the following phases of this project your job will be to keep the data synchronized and consistent between the memory, and the disk while executing multiple transactions simultaneously. In this

phase you just need to make sure that a single script file can execute from the beginning to the end producing correct output (log) and leaving the database in a consistent state.

Logging

Data Manager should keep a log file in which it can record all its actions. You need to record all Data Manager actions such as performed operations, creating new pages and swapping existing pages in and out of main memory. Following is the accepted logging format:

```
R X 13
SWAP OUT T-X P-6 B-11
SWAP OUT T-X P-2 B-11
SWAP IN  T-X P-8 B-13
Read: 13, John, 412-222-3333
W Y (18, Bob, 412-111-2222)
CREATE T-Y P-15 B-2
SWAP IN T-Y P-15 B-2
Inserted: 18, Bob, 412-111-2222
D Y
Deleted: Y
M X 609
MRead: 16, Tim, 609-222-3333
MRead: 19, Jim, 609-222-3333
```

Note that T-X P-6 B-11 means Table X, Page 6, Hash-Bucket 11.

Logging can significantly help you with debugging your project. In following phases these log records will be modified and utilized to restore the database to the previous consistent state if a transaction is aborted.

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 read and write operations* and the *average response time*.

Implementation

You have the option of implementing your prototype *myPTA* in any language and on Windows or Unix-based operating system. You will be required to demonstrate your system and submit an electronic copy of your code, log and data files.

Data Consistency

All transactions have to operate only on committed data. No stale data is allowed. Multigranulatity locking should be implemented in the following way: operations W and R should lock at the primary key level, operation M should lock at the level of the area code, and operation D sould lock at the file level.

Summary

In summary, you will implement three modules: Transaction Manager (TM), Scheduler, and Data Manager (DM). The TM 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 as well as the seed of the random number generator. The Scheduler implements the multi-granularity lock manager and deadlock detector. The DM is responsible of maintaining the data in the data files, i.e., executing the reads and the writes and ensuring atomicity of transactions.

Submission

Requirements:

- All submissions are electronic via the homework submission page (link available in course web page).
- You must work in teams of 4. You will have to declare the members and team name (if you wish) of your team by **5:00pm, Wednesday, March 22, 2017**. Send e-mail to cs2550-staff@cs.pitt.edu with your team name and the name and pitt username of each member, CCing the email to all members of the team.
- You must first submit a 2-page design document in (PDF) by **11:59pm, Monday, April 3, 2017**. Name the submission file using the team name (e.g., `wildcats.pdf`) or your usernames (e.g., `pitt2-usr9.pdf`), if you don't have a team name
- As part of your final prototype submission (using the same naming as in the case of your design document), you will need to submit a README file, in which you briefly describe the features of your concurrency control, any shortcomings and how to run it.
- You must submit your assignment before the due date (**11:59pm, Thursday, April 27, 2017**).
- The demos will be scheduled on *Friday, April 28, 2017*.

Academic Honesty

The work in this project is to be done *independently* as a 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.