DEPARTMENT OF COMPUTER SCIENCE, HANYANG UNIVERSITY

(DOCUMENT VERSION 1.1)

Simple Two-phase locking with Readers-writer Lock

Concurrent Programming

Programming Project #2

Final due date: Oct 22, 2017 (HARD DEADLINE)

1 TASK OVERVIEW

Database System에서 사용되는 Concurrency control 기법 중 하나인 Two-phase locking을 Readers-writer lock과 함께 간단한 형태로 구현해 본다.

2 TASK DETAIL

*N*개의 thread와 *R*개의 record가 있다. 각 record는 64bit integer(초기값 100) 하나를 가진다. 각 thread는 *global execution order*(초기값 0)가 정해진 값 *E*에 다다를 때 까지 반복하며 transaction을 수행한다.

각 thread가 반복적으로 하는 일(transaction)은 다음과 같다.

1. 3개의 서로 다른 record *i*, *j*, *k*를 임의로 선택한다.
2. Acquire global mutex
3. Record *i*의 값을 읽기 위해 reader lock을 잡는다.
    - waiting을 해야 하는 경우, Deadlock checking을 수행
4. Release global mutex
5. Record *i*의 값을 읽는다.
6. Acquire global mutex
7. Record *j*의 값을 쓰기 위해 writer lock을 잡는다.
    - waiting을 해야 하는 경우, Deadlock checking을 수행
8. Release global mutex
9. Record *j*에 읽었던 record *i*의 값 + 1을 더한다.
10. Acquire global mutex
11. Record *k*의 값을 쓰기 위해 writer lock을 잡는다.
    - waiting을 해야 하는 경우, Deadlock checking을 수행
12. Release global mutex
13. Record *k*에 읽었던 record *i*의 값을 뺀다.

    —————— commit ——————
14. Acquire global mutex
15. 현재 transaction에서 잡은 모든 reader/writer lock을 해제한다.
16. Global execution order를 1 증가시키고, 그 값을 받아온다. (*commit_id*)
    - *commit_id*가 *E*보다 크다면 현재 transaction이 변경한 record의 값을 되돌리고 (Undo), release global mutex, thread 종료

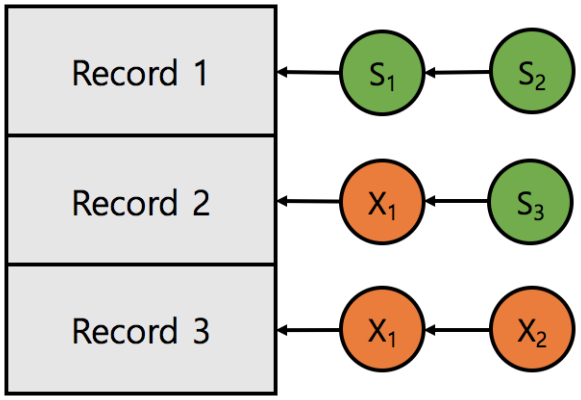
17. thread#.txt 파일에 다음과 같은 commit log를 append한다.

    [*commit_id*] [*i*] [*j*] [*k*] [*value of record i*] [*value of record j*] [*value of record k*]

18. Release global mutex

3 READERS-WRITER LOCK

각 record를 접근할 때, thread는 record에 대한 lock을 잡아야 한다. Readers-writer lock을 구현하여 여러 transaction이 동시에 record를 읽는 것을 허용하도록 한다. Lock을 요청하는 순서대로 획득할 수 있도록 linked-list 형태로 줄 세워 관리한다. Lock을 해제할 때에는 대기하고 있는 lock이 있을 경우 lock을 획득하도록 깨워주어야 한다.

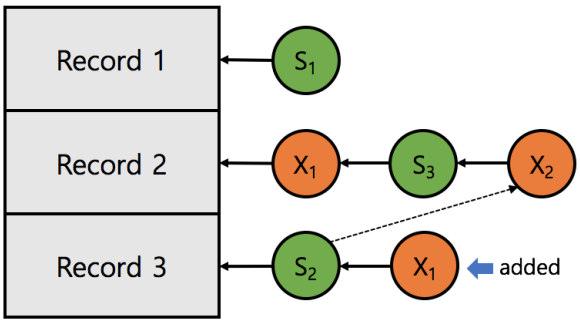


< S_i: thread i의 reader lock, X_i: thread i의 writer lock >

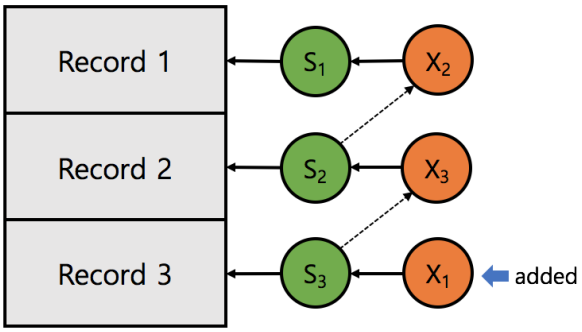
위의 그림에서, thread 1은 3개의 record의 lock을 잡고 commit 직전의 상태에 있다. thread 2는 record 1의 read lock을 잡고 record 3의 write lock을 잡기 위해 대기하고 있다. thread 3은 record 2의 read lock을 잡기 위해 대기하고 있는 상태이다.

4 DEADLOCK CHECKING

Thread가 reader/writer lock을 잡을 때, 자신 앞에 conflicting lock이 존재해서 waiting해야 하는 경우 deadlock checking을 수행해야 한다. Waiting cycle이 존재하는지 확인하여 deadlock을 검사할 수 있다. Deadlock이라고 판단되면 transaction이 변경했던 record의 값들을 되돌리고(Undo) transaction을 재시작 해야 한다.



< Deadlock example 1: thread1 -> thread 2 -> thread 1 >



< Deadlock example 2: thread1 -> thread 3 -> thread 2 -> thread 1 >

5 TEST PROTOCOL

프로그램은 3개의 command line argument를 입력받아야 한다. 실행 예제는 아래와 같다.

$ ./run *N R E*
N: trasaction을 수행할 thread의 수
R: record의 갯수
E: thread가 종료 여부를 판단할 global execution order

프로그램이 종료되면, *N*개의 output file이 생성되어야 한다. Output file의 이름은 thread#.txt (*N*이 3이라면 thread1.txt, thread2.txt, thread3.txt)로 한다. Output file에는 transaction의 commit log가 한 줄에 하나씩 출력되어 있어야 한다.

Output file은 다음과 같은 형식이다. (*N*=3, *R*=3)

thread1.txt
1 1 2 3 100 201 0 2 2 3 1 201 202 -101 4 3 1 2 102 2 200 ...

thread2.txt
3 1 3 2 -101 102 302 ...

thread3.txt
5 2 1 3 200 203 -98 ...

6 SUBMISSION

과제 제출은 GitLab으로 하며, project2 디렉토리를 생성하여 과제를 제출한다. project2 디렉토리 내에는 필수적으로 Makefile을 생성해야 한다. 실행파일명은 run으로 하며 마찬가지로 project2 디렉토리에 위치해야 한다.

보고서는 GitLab의 Wiki를 통해 제출한다. project2 라는 이름의 page를 생성하여 작성한다.