Database Programming Project

Project 4: Concurrent B+-Tree with Versioned Lock

Due: June 15. 23:59 pm

1. Introduction

In this project, you will implement a concurrent B+-Tree by extending the slotted page structure from Project 2 and the B+-Tree from Project 3. The main goal is to support safe and efficient multi-threaded access by introducing version-based locking to each page.
Each page will maintain a version number, and concurrent operations like insert and lookup will use this version to detect conflicts and ensure consistency. This project applies the optimistic concurrency control approach and leverages a technique called latch coupling (a.k.a. crabbing) for safe traversal during concurrent modifications.

이번 프로젝트에서는 Project 2에서 구현한 slotted page와 Project 3의 B+-Tree 구조를 확장하여 동시성을 지원하는 B+-Tree를 구현합니다. 핵심은 각 페이지에 버전 기반 락(version-based lock)을 도입하여 여러 스레드가 동시에 안전하게 트리에 접근할 수 있도록 하는 것입니다.
각 페이지는 버전 번호를 유지하고, insert, lookup 등의 연산은 이 번호를 활용해 충돌을 탐지하고 일관성을 보장합니다. 이 프로젝트는 낙관적 동시성 제어(Optimistic Concurrency Control) 방식과 Latch Coupling (Crabbing) 기법을 사용해 동시 탐색을 구현합니다.

2. Getting Started

You can reuse the codebase from Project 3. You will add lock-related logic and version field to page.hpp, page.cpp, make btree.cpp thread-safe.

File Overview
- page.hpp/page.cpp: Add version field and implement locking APIs.
- btree.hpp/btree.cpp: Implement thread-safe insert and lookup functions.

Project 3의 코드베이스를 그대로 활용할 수 있습니다. page.hpp와 page.cpp에 버전 필드 및 기능을 추가하고, btree.cpp의 insert 및 lookup 함수를 thread-safe하게 수정하면 됩니다.

파일 구성
page.hpp/page.cpp: 버전 필드 추가 및 락 관련 함수 구현
btree.hpp/btree.cpp: thread-safe insert 및 lookup 구현

3. Design Overview

3.1 Version-based Locking

You will implement versioned locking. Each page has a uint64_t version field. Threads perform operations as follows:

- read_version():
  Reads the current version of the page before accessing data.
- validate_read(uint64_t old_version):
  Compares the current version with the previously read version to check for modifications. Returns true if safe, false if retry is needed.
- try_read_lock(uint64_t version):
  Checks whether the page is available through the version information, and returns true if it is available and false if it is not. If lock acquisition fails, the operation should be restarted from the beginning.
- try_write_lock():
  When the page is available, atomically marks the page as “in use” (e.g., using odd version).
  Checks whether the page is available. If it is, atomically mark the page as 'in use' (e.g., using odd version) and return true. If it is not, return false. If lock acquisition fails, the operation should be restarted from the beginning.
- read_unlock(uint64_t old_version);
  Use validate_read() to check whether the page is changed.
- write_unlock():
  Finalizes version update after writing and unlock the page.

이번 프로젝트에서는 버전 기반 락에 대해서 구현합니다. 각 페이지는 uint64_t version 필드를 가지며, 스레드는 다음과 같은 방식으로 연산을 수행합니다:

- read_version():
  데이터를 읽기 전에 현재 버전을 읽어 반환합니다.
- validate_read(uint64_t old_version):
  작업 후 현재 버전과 비교하여 중간에 변경이 있었는지 확인합니다. 변경이 없다면 true, 변경되었다면 false를 반환합니다.
- try_read_lock(uint64_t version);
  버전 정보를 통해 현재 접근 가능한 상태인지 확인하고, 사용 가능하면 true, 불가능하면 false를 반환합니다. 락 획득 실패 시 연산을 처음부터 수행해야 합니다.
- try_write_lock():
  페이지가 사용 가능한 상태인지 확인하고, 사용 가능하면 atomic 연산으로 페이지를 "사용중"으로 표시(예: 홀수 버전)한 뒤 true를 반환하고, 불가능하면 false를 반환합니다. 락 획득 실패 시 연산을 처음부터 수행해야 합니다.
- read_unlock(uint64_t old_version);
  validate_read() 함수를 사용하여, 버전이 바뀌었는지 확인합니다.

- write_unlock():
  쓰기 작업이 끝난 후 버전을 갱신하여 페이지를 unlock합니다.

3.2 Latch Coupling (Crabbing)

To traverse the B+-Tree safely during insertions and lookups, use latch coupling:
1. Lock the current node. (read_lock)
2. Identify the child node.
3. Lock the child, then unlock the current node. (read_lock)
4. Repeat until the leaf is reached.
5. If a split occurs at a node:
   - Hold the latch on the parent node during the split.
   - Insert the new entry into the parent.
   - If the parent also splits, propagate upward recursively.
   - Ensure that latches are acquired in top-down order to avoid deadlocks.

This technique ensures that no node is modified without holding the appropriate latch, and latch release is carefully coordinated to maintain correctness and avoid deadlock.

B+-Tree를 안전하게 탐색하거나 삽입하기 위해 Latch Coupling 기법을 사용합니다:
1. 현재 노드를 lock합니다. (read_lock)
2. 자식 노드를 식별합니다.
3. 자식 노드를 lock한 뒤, 현재 노드를 unlock합니다. (read_lock)
4. 이를 반복하여 리프 노드에 도달합니다.
5. 만약 어떤 노드에서 split이 발생하면:
   - 부모 노드의 락을 유지한 상태에서 분할 작업을 수행합니다.
   - 새로운 엔트리를 부모 노드에 삽입합니다.
   - 부모도 분할이 발생할 경우, 연산은 재귀적으로 상위 노드로 전파됩니다.
   - 락은 항상 상위에서 하위(top-down) 순서로 획득해야 데드락을 방지할 수 있습니다.

이 방법은 어떤 노드도 적절한 락 없이 수정되지 않도록 보장하며, 락 해제 순서를 조절해 동기화의 정확성과 데드락 회피를 모두 만족시킵니다.

4. Implementation Tasks

In page.hpp:
- uint64_t version;
  Stores the current version information of the page.
  페이지의 현재 버전 정보를 저장하는 버전 필드입니다.

In page.cpp:
- uint64_t read_version();
  Return the current version before reading the page.
  페이지를 읽기 전에 현재 버전을 반환합니다.

- bool validate_read(uint64_t old_version);
  Check whether the page is changed using the version information.
  버전을 통해 페이지가 바뀌었는지 확인합니다.
- bool try_read_lock(uint64_t version);
  Check if page version is stable for read.
  페이지 버전이 읽기가 가능한 상태인지 확인합니다.
- bool try_write_lock();
  Check the page is available, acquire write lock by changing version atomically.
  페이지를 사용 가능한지 확인한 후 atomic 연산으로 쓰기 락을 획득합니다.
- bool read_unlock(uint64_t old_version);
  Check whether the page is changed using the version information.
  버전을 통해 읽기 작업 중 페이지가 변경되었는지 확인합니다.
- void write_unlock();
  Finalize the version after writing.
  쓰기 작업 이후 버전을 갱신합니다.

In btree.cpp:
- void insert(char* key, uint64_t val);
  Traverse the tree with latch coupling and insert the key-value pair.
  latch coupling을 활용해 키-값을 삽입합니다.
- uint64_t lookup(char* key);
  Safely find a key under concurrent access.
  동시 접근에서도 안전한 탐색을 수행합니다.

5. Submission Guidelines

Please submit your page.hpp, page.cpp and btree.cpp file to the e-Campus. If you have any questions, please upload your question to the e-campus.

page.hpp, page.cpp파일과 btree.cpp 파일을 e-Campus를 통해 제출해주시고, 질문 사항 또한 e-Campus를 통해 올려주시기 바랍니다.