[CSED421 Project5]

# In-memory Index

[작성: 2019. 4. 15]

## 목표:

속도가 빠른 In-memory index를 구현하시오.

## 문제 정의:

Load, Transaction 데이터 파일이 파라미터로 주어지면 이에 대한 workload를 수행하고 그 결과 및 수행 시간을 출력하는 시스템을 구현한다 (main.cpp 참조). 단, 시스템은 single thread로 작동해야한다.

## 컴파일 방법:

mkdir ./build; cd ./build; cmake .. ; make -j

## 실행 방법:

./workload <load 파일> <transactions 파일> <load size> <transactions size>로 실행이 되어야 한다. (main.cpp 참조)

## 입력:

1. load 파일, transactions 파일
   A. 파일의 각 line은 하나의 index operation (INSERT, UPDATE, READ, SCAN)을 의미한다.
   B. INSERT <key>: <key>를 가지는 entry를 insert한다.
   C. UPDATE <key>: <key>를 가지는 entry를 update한다.
   D. READ <key>: <key>를 가지는 entry를 read한다.
   E. SCAN <key> <num>: <key>보다 크거나 같은 key들을 가지는 entry들을 <num>개 만큼 scan한다.
2. load size, transactions size: Load와 Transaction 각각의 index operation 개수이다.

## 출력:

프로그램은 아래 예시와 같이 values of read entries 총합, values of scanned entries 총합, load에 걸린 시간(ms), transaction 수행에 걸린 시간(ms)를 출력한다.

```
load_fname: ./data/c_zipf_hashed_10M_randint_load.dat
txns_fname: ./data/c_zipf_hashed_10M_randint_txn.dat
Execute 10000000 operations
load time(ms): 12481.6
Execute 10000000 operations
txns time(ms): 9592.83
sum of values of read entries: 10000000
sum of values of scanned entries: 0
```

**구현:**

아래의 구글 드라이브 링크에서 기본적인 코드를 받을 수 있다.

https://drive.google.com/open?id=18hzc8Wup-XBjXhbtuy_Qm6DKrtMl1ZyI

Load 파일, transaction 파일에 대한 파서는 제공된다. (util.hpp와 main.cpp 참조)

목표는 아래 main.cpp 파일 InMemoryIndex class에서 각 Index operation에 대응 되는 insert, update, read, scan 함수를 구현하는 것이다. 주어진 main.cpp에는 std::map으로 index가 구현되어 있다. 공정한 평가를 위해 timer로 성능을 구하고, 결과 개수를 구하는 부분은 수정을 금지한다.

```
class InMemoryBPtree {
public:
        std::map<uint64_t, int8_t> index;

        void insert(uint64_t key, int8_t value) {
                index[key] = value;
                // Impleent this function
        }

        void update(uint64_t key, int8_t value) {
                index[key] = value;
                // Impleent this function
        }

        int8_t read(uint64_t key) {
                return index[key];
                // Impleent this function
        }

        uint64_t scan(uint64_t key, int8_t num) {
                auto it = index.lower_bound(key);
                uint64_t result = 0;
                for (int8_t i = 0; i < num; ++i) {
                        if (it == index.end()) break;
                        result += it->second;
                        ++it;
                }
                // Impleent this function
                return result;
        }
};
```

## 제출물 및 평가:

Project 5는 2인 팀 단위로 진행된다. 모든 팀들은 5/20까지 중간결과물을 LMS에 올릴 수 있고, 중간 결과물을 제출한 팀만이 이후 6/11까지 프로젝트 진행이 가능하다. LMS에 업로드한 시스템은 연구실 서버에서 성능 및 정확성(correctness) 평가 후 매일 24:00에 다음의 사이트에 순위가 갱신된다.
https://sites.google.com/a/dblab.postech.ac.kr/db/programmingcontestindex

제출 시, main.cpp 파일이 있는 위치에서 cmake로 컴파일 가능하도록 (g++ 버전은 7.3.1), 데이터를 제외한 모든 파일을 하나로 압축하여 제출한다. (CMakeLists.txt 참조)

**만약 제출된 시스템이 정확하게 작동하지 않으면 순위에서 제외된다.**

순위 결정에 쓰이는 load, transaction 파일은 각 <workload_type>_<size>_load.dat, <workload_type>_<size>_txns.dat으로 다음 구글 드라이브에서 다운로드할 수 있다. data 디렉토리에 압축을 풀면 된다.
https://drive.google.com/drive/folders/1C1Q6jBxRcaYCTTqfCuoIwK5A99HujRWe?usp=sharing

순위 결정은 각 데이터 사이즈에 대해 workload들을 수행하는 throughput(Mops/ms)의 평균을 구한 뒤, 500M, 50M, 10M 순으로 가중치를 둔다. Transaction C의 경우 READ가 100%이며, A의 경우 READ 50%, UPDATE 50%, 그리고 E의 경우 SCAN 95%, INSERT 5%로 구성되어 있다.

평가 시 1시간 timeout을 두고, 이를 넘어가면 correctness하지 않은 것으로 간주한다.

또한, 과제가 종료되는 시점인 6/11까지 최종 보고서를 작성하여 LMS에 제출한다.

성적 산출 시 본 과제는 전체 과제의 35%를 차지하며, reference solution보다 순위가 높은 시스템을 구현한 수강생은 **성적이 한 등급 상향**된다. Reference solution 바이너리는 과제가 어느 정도 진행된 시점에서 수강생들에게 제공되며, Haswell 이하의 CPU, 4GB 이상의 메모리를 가진 머신에서 돌려 각자 구현한 시스템과 성능을 비교할 수 있다.