CZ4031 Database System Principles

Project 1 Group 14

AMITA RAVI (U2023964C)
BRYAN ONG WEIXIN (N2203346D)
CHAI YOUXIANG (U2022575A)
CHANTHAROJWONG KASIDIS (U2020731L)

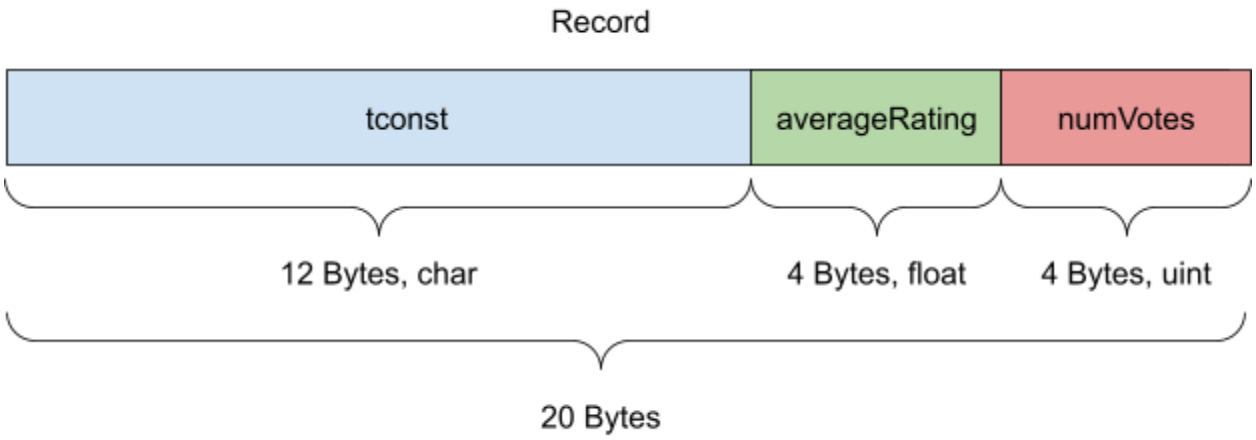
Name	Contribution
AMITA RAVI	Implementation, Documentation
BRYAN ONG WEIXIN	Implementation, Experiment
CHAI YOUXIANG	Implementation, Experiment
CHANTHAROJWONG KASIDIS	Implementation, Documentation

1. Implementation

1.1. Design of Storage Component

Disk Capacity
For simplicity, the cap of 500MB was chosen.

Content of a record
A record in the dataset contains 3 attributes: “tconst”, “averageRating” and “numVotes”.



Justification and breakdown
tconst → 12B (char)

- Length of “tconst” in the dataset is 9B Max, which is “tt9916778”
- Since the block size is 200 Bytes, we decided to give tconst 12B to fit 10 records into a block for simplicity

averageRating → 4B (float)

- Max value of “averageRating” is 10.0, which can be represented by a float safely

numVotes → 4B (unsigned int)

- Max value of “numVotes” is 2279223 which is much less than 2^{32}, therefore, it can be represented by an unsigned int safely

We have chosen a spanned record for our implementation as spanned records might reduce the amount of space that is required for a data set when data records vary significantly in length and it permits a record to extend across or span control interval boundaries. Spanned records can only be used with sequenced data.

1.2. B+ Tree Indexing Component

Following is the breakdown of the memory usage of the B+ Tree node headers:

- isLeaf: 1 bit (as boolean type)
- maxKeyNum: 32 bits (as integer type)
- maxPointerNum: 32 bits (as integer type)
- currentKeyNum: 32 bits (as integer type)
- currentPointerNum: 32 bits (as integer type)
- Struct Address addressInDisk: 96 bits

Size of node = size of header + size of key array + size of pointer array
=127+96+32*(k)+ 64*(k+1) bits

k is the maximum number of keys

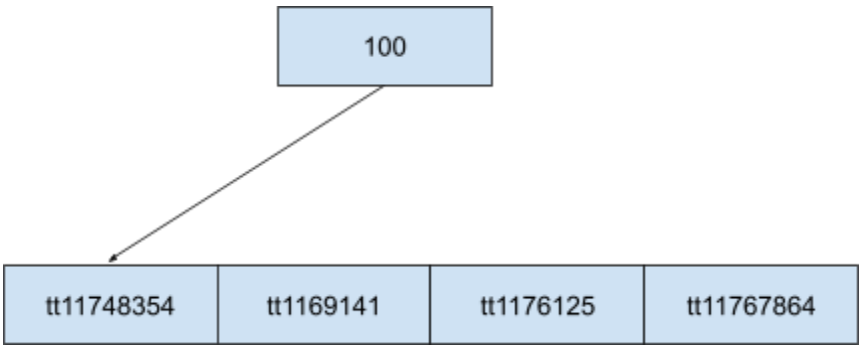
From the above calculation,
Size of node= 287+96*(k)= 200 bytes= 200*8 bits
k = (200*8)-287/96
= 13.67 ~ 13 (Taking the floor function)

The depth of the B+ tree is log(N)/log(k) where N is the total number of record and k is the maximum number of keys in a node.

Therefore the depth of our b+ tree = log(1070319)/log(13)
= 5.413 ~ 5

1.3. Duplicate Key Handling

Before inserting into our storage component, the records are sorted in ascending order according to the key, “numVotes”. Therefore, it is only necessary to store the pointer to the first record of each key. We can simply perform a simple linear traversal to access individual records of a key.



2. Experiments

Note: For measuring running time, we used c++'s high_resolution_clock to get the start and end times, then subtract the difference for the time taken.

2.1. Experiment 1

Number of records: 1,070,318
Size of a record: 20 Bytes
Number of records stored in a block: 10
Number of blocks for storing the data: 107,032

2.2. Experiment 2

Parameter n of the B+ tree: 13
Number of nodes of the B+ tree: 5,530
Number of levels of the B+ tree: 5
Content of the root node:

3591	7911	16859	47952									

2.3. Experiment 3

Number of index nodes accessed: 5
Number of data blocks accesses: 5
Average of "averageRating's" of the records that are returned: 6.74419
Running time of the retrieval process: 0.423ms
Number of data blocks by a brute-force linear scan: 107,032
We will have to scan through all the data blocks since it is non-sequential.
Running time of brute-force linear scan: 35.260ms

2.4. Experiment 4

Number of index nodes accessed: 5
Number of data blocks accesses: 5
Average of "averageRating's" of the records that are returned: 6.91905
Running time of the retrieval process: 0.408ms
Number of data blocks by a brute-force linear scan: 107,032
We will have to scan through all the data blocks since it is non-sequential.
Running time of brute-force linear scan: 36.335ms

2.5. Experiment 5

Number nodes of the updated B+ tree: 5,528
Number of levels of the updated B+ tree: 5
Content of the root node of the updated B+ tree (only the keys):

3591	7911	16859	47952									

Running time of the process: 0.71ms
Number of data blocks by a brute-force linear scan: 101,309
Running time of brute-force linear scan: 34.925ms

3. Source Code & Installation guide

3.1. Source Code

Our project's source code is at https://github.com/ckasidis/cz4031-proj-1

3.2. Docker Installation Guide

**requirement: have Docker installed on your system
1. **docker build -t cz4031 .**
2. **docker run --rm cz4031**

3.3. Linux/macOS Installation Guide

**requirement: have g++ available on your system
run **./run_main.sh**
OR
run the following commands
1. **compile g++ -o main src/*.cpp --std=c++17 main.cpp -I include**
2. **./main**

3.4. Linux/macOS Installation Guide

**requirement: have g++ available on your system
run **./run_main_windows.sh**
OR
run the following commands
1. **compile g++ -o main src/*.cpp --std=c++17 main.cpp -I include**
2. **./main.exe**