CS561-Project 0

Anwesha Saha

January 2025

1 Required Workload

On Local System

- W1: 10K point queries on 1M sorted data = **189.667 microseconds**
- W2: 1K range queries, selectivity: 0.001 on 1M sorted data = **19.3333 microseconds**
- W3: 1K range queries, selectivity: 0.1 on 1M sorted data = **26.6667 microseconds**
- W4: W1 with unsorted data = **341.667 microseconds**
- W5: W2 with unsorted data = **20.6667 microseconds**
- W6: W3 with unsorted data = **60.6667 microseconds**

On SCC (using make for compiling - 1 core)

- W1: : 10K point queries on 1M sorted data = **804 microseconds**
- W2: 1K range queries, selectivity: 0.001 on 1M sorted data = **71 microseconds**
- W3: 1K range queries, selectivity: 0.1 on 1M sorted data = **72.6667 microseconds**
- W4: W1 with unsorted data = **821.333 microseconds**
- W5: W2 with unsorted data = **78.6667 microseconds**
- W6: W3 with unsorted data = **73 microseconds**

821.333 microseconds

2 Experiments

2.1 Workload W1

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp

2. Generate 1 M sorted data with 10K point queries and 0 range queries
   Command Run: ./generator -N 1000000 -P 10000 -R 0 –sort

3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm

4. Run Queries with Generated workload and query file
   Command Run: ./run_zm

5. Output observed:
   Time taken to perform point queries from zonemap = 189.667 microseconds

2.2 Workload W2

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp

2. Generate 1 M sorted data with 0 point queries and 1K range queries with selectivity 0.001
   Command Run: ./generator -N 1000000 -P 0 -R 1000 -s 0.001 –sort

3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm

4. Run Queries with Generated workload and query file
   Command Run: ./run_zm

5. Output observed:
   Time taken to perform range query from zonemap = 19.3333 microseconds

2.3 Workload W3

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp

2. Generate 1 M sorted data with 0 point queries and 1K range queries with selectivity 0.1
   Command Run: ./generator -N 1000000 -P 0 -R 1000 -s 0.1 –sort

3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm

4. Run Queries with Generated workload and query file
   Command Run: ./run_zm

5. Output observed:
   Time taken to perform range query from zonemap = 26.6667 microseconds

2.4 Workload W4

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp

2. Generate 1 M unsorted data with 10K point queries and 0 range queries
   Command Run: ./generator -N 1000000 -P 10000 -R 0

3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm

4. Run Queries with Generated workload and query file
   Command Run: ./run_zm

5. Output observed:
   Time taken to perform point queries from zonemap = 341.667 microseconds

2.5 Workload W5

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp

2. Generate 1 M unsorted data with 0 point queries and 1K range queries with selectivity 0.001
   Command Run: ./generator -N 1000000 -P 0 -R 1000 -s 0.001

3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm

4. Run Queries with Generated workload and query file
   Command Run: ./run_zm

5. Output observed:
   Time taken to perform range query from zonemap = 20.6667 microseconds

2.6 Workload W6

1. Compile code for workload generation using:
   g++ -std=c++11 -o generator workload_generator.cpp
2. Generate 1 M unsorted data with 0 point queries and 1K range queries with selectivity 0.1
   Command Run: ./generator -N 1000000 -P 0 -R 1000 -s 0.1
3. Compile zonemaps.cpp and main.cpp
   Command Run: g++ -std=c++11 -O2 main.cpp zonemaps.cpp -o run_zm
4. Run Queries with Generated workload and query file
   Command Run: ./run_zm
5. Output observed:
   Time taken to perform range query from zonemap = 60.6667 microseconds

3 Technical Questions

3.1 Expected memory footprint in bytes to build the zonemap

Number of elements = N (given)
Number of entries in each zone = d (given)
Therefore number of zones required for N elements = N/d
Let us calculate first the size of one zone
In one zone, we have 4 components:
Let us assume that the size of type T is T bytes

- The first component is a vector of elements and we know this has d elements each of a type T so the total size in bytes is T*d bytes = Td
- The next component is the min and max values which are also of type T so the total memory required = 2*T = 2T
- The last component is an unsigned integer to hold the number of elements in the zone = 4 bytes (assuming that usually an unsigned int is of size 4 bytes)

So total size of one zone = $Td + 2T + 4$
Let us now calculate the size of the zonemap The zonemap class also has multiple components:

- It contains a vector of all elements of type T and we know there are N elements in total. So total memory = TN

- The next item it contains is a vector of zones. We know we have N/d number of zones in the zonemap and we know the size of each zone from previous calculation.
  So total size = $N/d * (Td + 2T + 4)$

- Next we have 2 more variables both of which are unsigned integers to maintain number of elements per zone and the number of zones in the zonemap. So considering each is of size 4 bytes we have 8 bytes from these 2 variables.
  Therefore total size of zonemap =

$$\mathbf{TN + N/d * (Td + 2T + 4) + 8} \tag{1}$$

Total size of zonemap without element storage =

$$\mathbf{N/d * (2T + 4) + 8} \tag{2}$$

3.2 Coping mechanism of memory footprint is larger than available memory

If the zonemap memory footprint is larger than the available memory, the most obvious choice could be to modify the design to accommodate more elements per zone so the number of zones would reduce - this would also deteriorate the efficiency of the zonemap but could help in reducing size. We can use other small space-efficient structures like Bloom filters to determine whether a specific key is within the block or not to maintan efficiency if the memory budget permits. If the zone is really large, can find the required zone and then use the bloom filter to check whether the entry is present. For cases where a dimension and fact table exist separately, we could use the technique mentioned in the paper "Dimensions Based Data Clustering and Zone Maps" and cluster data based on an attribute in the dimension table that is used frequently for querying. This helps because randomly distributed data will not have optimized min/max ranges that actually help and might have a more spread out number whereas grouping data by a meaningful attribute will ensure that zones have a more compact range of values.