Project 1 : Implementing a Simple Zone Map

Due: Friday 20th September, 2024 at 23:59

Background

A zone map is a light-weight sparse index that maintains the minimum and maximum values of one (or more) attribute(s) of data that is stored across multiple contiguous blocks or pages. Theses blocks or pages are also referred to as *zones* [1]. A zone map helps in pruning the number of zones qualifying for a range or point query, and thereby, reduces the number of accesses (or I/Os) to slower storage. Typically, a zone map is read once from storage, and if possible, pinned in memory during workload execution. The zone map may be updated in memory as an when required as new data in ingested and moved to the storage. When realizing (point or range) queries, first we probe the in-memory zone maps. The target key (for point query) or target key range (for range queries) is first compared with the maps (i.e., min/max values) of every block or zone, and a block is read from the secondary only if the zone map probe return true. However, if the zone map probe returns false, we can avoid performing unnecessary I/Os to slower storage, and thus, improve the query performance.

Objective

The objective of this project is to implement a simple zone map and evaluate its performance on both point and range queries. You will benchmark the zone map performance for both when (i) the data is stored as a *heap file* and (ii) the data is stored as a *sorted file*.

Language of Implementation. It is strongly suggested that you use C/C++ to do your implementation if you are familiar with the languages. If you are unfamiliar with C/C++, but want to learn it while working on the project, you can find some under **Useful Resources** in the projects page. If neither of the above is suitable for you, we can do the implementation in Java.

Workflow. If you are implementing the project in C/C++, click on this link. If you plan to implement in Java, click here. The general workflow for the project is as follows.

1. Once you clone the repository and navigate into it, you will see an API that contains a header file with basic function definitions for a zone map. Your task is to implement the data structure and the corresponding operations. Make sure you go over the `README` before you start working on the project. You are free to modify certain components to improve performance.

2. While the point query API is given, you will need to implement the API for range queries.

3. To test your implementation, we have provided test workloads `T1`, `T2`, and `T3`. The workloads can be found inside the `workloads` folder.

4. Once you run the test workloads successfully, you should run the larger workloads, i.e., `W1` – `W6`. The specification of the six workloads are as follows.

`W1:` 5M inserts in sorted order followed by 10K point queries

`W2:` 5M inserts in sorted order followed by 1K range queries of selectivity 0.001

`W3:` 5M inserts in sorted order followed by 1K range queries of selectivity 0.1

`W4:` `W1` with entries in random order

`W5:` `W2` with entries in random order

`W6:` `W3` with entries in random order

Plotting the results: Note the execution latency for each workload and plot them along the y-axis of a graph. The x-axis of this graph is the six workloads (`W1`-`W6`). Within the there should be two lines (or groups of two bars – *your choice*), one for experiments on sorted data, another for the ones on unordered data.

5. Now that you have successfully implemented a zone map and run experiments with it, answer the following research question based on your understanding.

   (i) What is the expected memory footprint (in bytes) to build the zone map? (assume: number of elements is N and every zone has d entries.)

   (ii) In practice, we may have raw data stored on disk, and the zone map is maintained in memory to reduce the number of required I/Os to answer queries. What should we do if we only have a limited memory budget to build the zone map (i.e., when the memory budget of M bytes is smaller than the expected memory footprint)?

Deliverables & Submission Policy

(A) Submit a PDF with the latency numbers for each experiment represented in a plot. Feel free to add any additional results from the experiments that you think is relevant. In the same PDF, add the answers to the two research questions. (B) Submit as a ZIP the zone map implementation code that runs all the given workloads. It is required to have comments within the implementation, that explain various design decisions.

Do NOT upload your code to public repositories, such as GitHub and Bitbucket. Please submit a single PDF and a ZIP file on Gradescope. The due date for Project 1 is **Friday 20th September, 2024 at 23:59**.

References

[1] M. Ziauddin, A. Witkowski, Y. J. Kim, D. Potapov, J. Lahorani, and M. Krishna. 2017. *Dimensions based data clustering and zone maps*, PVLDB Endowment. 10(12), pp. 1622-1633. DOI: https://doi.org/10.14778/3137765.3137769.