TECHNICAL REPORT

In the technical report, we give detailed proofs of the complexity of our algorithms. And then, we show the result of the experiment on HDD.

Proof

We present Lemma 1 and Lemma 2 to discuss the write efficiency and query efficiency of the LSM-segment tree.

Lemma 1. *The amortized insertion I/O cost of LSM-Segment tree is* $O(\log_2 \frac{N}{M}/B)$.

Proof. We first calculate the I/O cost of merging two segment trees. To build a segment tree of $M \cdot 2^i$ leaf nodes, we need a merge sort of all the leaf nodes, which costs $O(M \cdot 2^i/B)$ I/Os. Starting from the leaf level, we perform a linear scan of all the nodes in a level to build the upper level, which means we need $O(M \cdot 2^i/B)$ I/Os to scan a level with $M \cdot 2^i$ nodes and create a new level with $M \cdot 2^i/B$ nodes. So, the total cost of the level construction is $O(M \cdot 2^i/B) + O(M \cdot 2^i/B^2) + O(M \cdot 2^i/B^3) \cdots = O(M \cdot 2^i/B)$. Therefore, to build a segment tree of $M \cdot 2^i$ leaf nodes costs $O(M \cdot 2^i/B)$ I/Os.

Next,the i-th segment tree with $M \cdot 2^{i-1}$ leaf nodes will need at most $N/(M \cdot 2^{i-1})$ times of merging, so it totally needs $O(M \cdot 2^{i-1}/B) \cdot N/(M \cdot 2^{i-1}) = O(N/B)$ IOs. There are $\log_2 \frac{N}{M}$ segment trees, so the total I/O cost is $\log_2 \frac{N}{M} \cdot O(N/B) = O(\log_2 \frac{N}{M} N/B)$. Hence, for each insertion, the amortized I/O cost is $O(\log_2 \frac{N}{M} N/B)/N = O(\log_2 \frac{N}{M}/B)$. □

Lemma 2. *The worst-case query I/O cost of LSM-Segment tree is* $O(\log_2 \frac{N}{M} \log_B \frac{N}{B})$.

Proof. We first calculate the I/O cost of query on one segment tree. We access at most two blocks in each level in the query. The reason is that the query range will partially overlap with at most two blocks as the two ends on a segment. Since a segment tree can only have $\log_B(N/B)$ levels, to answer the query in one segment tree costs $O(\log_B(N/B))$ I/Os.

Since there are $\log_2 \frac{N}{M}$ segment trees, the total cost to obtain the whole aggregate result is $O(\log_2 \frac{N}{M} \log_B \frac{N}{B})$. □

By Lemma 1 and Lemma 2, we know that the write efficiency of the LSM-Segment tree is $O(\log_2 \frac{N}{M}/B)$ and the query efficiency is $O(\log_2 \frac{N}{M} \log_B \frac{N}{B})$.

Next, we present Lemma 3 and Lemma 4 to discuss the write efficiency and query efficiency of the LSM-Partition-segment tree.

Lemma 3. *The amortized insertion I/O cost of LSM-Partition-Segment tree is* $O(\log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$.

Proof. We first calculate the I/O cost of one Partition-Segment tree. To build a partition-segment tree, we need an external sorting by $1 - st$ dimension. The external sorting costs $O(\frac{N}{B} \log_M \frac{N}{B})$ I/Os. Then we need to sort each strip and build a segment tree for each strip, which totally costs $O(\frac{N}{B} \log_M \frac{N}{B})$ I/Os. Finally, we need to sort each grid and build a segment tree for each grid, which totally costs $O(\frac{N}{B} \log_M \frac{N}{B})$ I/Os. Therefore, it costs $O(\frac{N}{B} \log_M \frac{N}{B})$ I/Os to build a partition-segment tree.

Similar to Lemma 1, for each insertion, the amortized I/O cost is $O(\log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$. □

Lemma 4. *The worst-case query I/O cost of LSM-Partition-Segment tree is* $O((N/B)^{\frac{1}{3}} \log_B N \log_2 \frac{N}{M})$.

Proof. We first calculate the I/O cost of query on one partition-segment tree. For the at most $\lceil (N/B)^{1/3} \rceil$ fully overlapped strips, search the second-dimension segment trees, which cost at most $O((N/B)^{\frac{1}{3}} \log_B \frac{N}{B})$ I/Os. For the at most 2 partially overlapped ones, there are two kinds of grids. For the at most $\lceil (N/B)^{1/3} \rceil$ fully overlapped ones, search the first-dimension segment trees, which cost at most $O((N/B)^{\frac{1}{3}} \log_B \frac{N}{B})$ I/Os. For at most 2 partially overlapped ones, scan the whole grid the obtain the answer, which cost $O((N/B)^{\frac{1}{3}})$ I/Os. Hence, the total query I/O cost is $O((N/B)^{\frac{1}{3}} \log_B \frac{N}{B}) + 2 \cdot O((N/B)^{\frac{1}{3}} \log_B \frac{N}{B}) + 4 \cdot O(N^{\frac{1}{3}}/B) = O((N/B)^{\frac{1}{3}} \log_B \frac{N}{B})$.

Since there are $\log_2 \frac{N}{M}$ partition-segment trees, the total cost to obtain the whole aggregate result is $O((N/B)^{\frac{1}{3}} \log_B N \log_2 \frac{N}{M})$. □

By Lemma 3 and Lemma 4, we know that the write efficiency of the LSM-Partition-Segment tree is $O(\log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$ and the query efficiency is $O((N/B)^{\frac{1}{3}} \log_B N \log_2 \frac{N}{M})$.

Next, we present Theorem 1 and Theorem 2 to discuss the write efficiency and query efficiency of the WORQ-tree.

Theorem 1. *The amortized insertion I/O cost of WORQ-tree is* $O(d \log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$.

Proof. Given that there are d levels of construction and in each level the construction costs $O(\frac{N}{B} \log_M \frac{N}{B})$ I/Os, it costs $O(d \frac{N}{B} \log_M \frac{N}{B})$ I/Os to build one tree in a WORQ-tree.

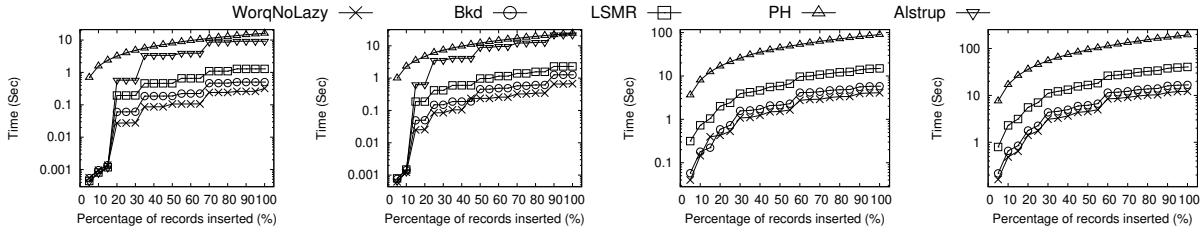
Hence, similar to Lemma 1, considering there are $\lceil \log_2 \frac{N}{M} \rceil$ trees, the amortized cost of insertion is $O(d \log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$. □

Theorem 2. *The worst-case query I/O cost of WORQ-tree is* $O((N/B)^{\frac{2d-3}{2d-1}} \log_B \frac{N}{B} \log_2 \frac{N}{M})$.

Proof. We prove it by induction. Assume that the time complexity for d-dimension queries in one tree is $O((N/B)^{\frac{2d-3}{2d-1}} \log_B \frac{N}{B})$. When $d = 2$, it obviously holds. When $d > 2$, there are two kinds of strips: at most two strips partially overlapped with the query range in the d-th dimansion and at most $\lceil (N/B)^{\frac{2}{2d-1}} \rceil$ strips fully overlapped. For the at most two partially overlapped strips, it costs $O((N/B)^{\frac{2d-3}{2d-1}})$ I/Os to scan the whole strip to obtain the result. For the at most $\lceil (N/B)^{\frac{2}{2d-1}} \rceil$ fully overlapped strips, it costs $O(\left((N/B)^{\frac{2d-3}{2d-1}}\right)^{\frac{2d-5}{2d-3}} \log_B N) = O((N/B)^{\frac{2d-5}{2d-1}} \log_B \frac{N}{B})$ I/Os in each strip, so it totally cost $O((N/B)^{\frac{2d-3}{2d-1}} \log_B \frac{N}{B})$ I/Os.

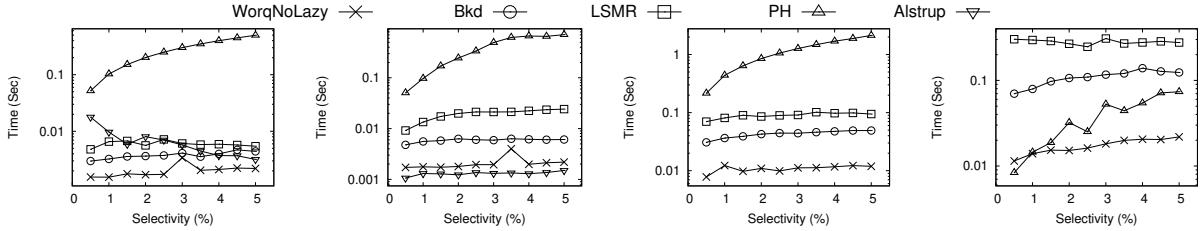
Hence, since there are $\lceil \log_2 \frac{N}{M} \rceil$ trees, the I/O cost of query is $O((N/B)^{\frac{2d-3}{2d-1}} \log_B \frac{N}{B} \log_2 \frac{N}{M})$. □

By Theorem 1 and Theorem 2, we know that the write efficiency of the WORQ-tree is $O(d \log_M \frac{N}{B} \log_2 \frac{N}{M}/B)$ and the query efficiency is $O((N/B)^{\frac{2d-3}{2d-1}} \log_B \frac{N}{B} \log_2 \frac{N}{M})$.



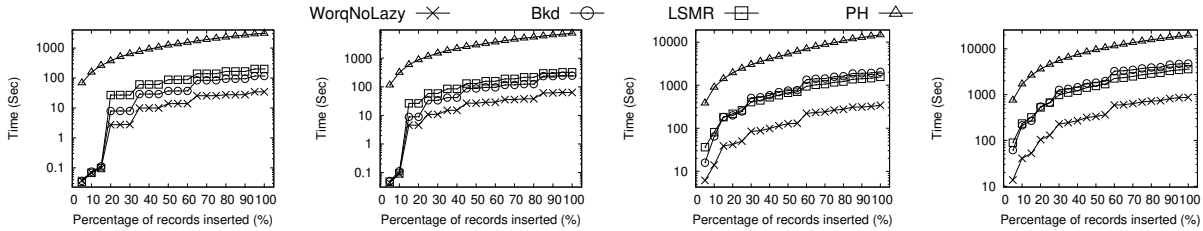
(a) Gowalla-small (b) London-small (c) Chicago-small (d) Berlin-small

Figure 15: Insertion time for small-datasets on HDD



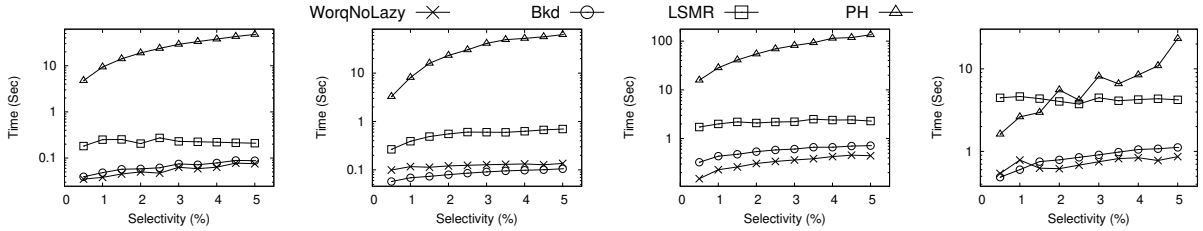
(a) Gowalla-small (b) London-small (c) Chicago-small (d) Berlin-small

Figure 16: Query time for small-datasets on HDD



(a) Gowalla (b) London (c) Chicago (d) Berlin

Figure 17: Insertion time for original-datasets on HDD



(a) Gowalla (b) London (c) Chicago (d) Berlin

Figure 18: Query time for original-datasets on HDD

Additional Experiment: Efficiency on HDD

Besides SSD, we also conduct experiments on HDD to evaluate the efficiency and effectiveness of our data structures.

The write efficiency and query efficiency results for small-datasets on SSD are shown in Figure 15 and Figure 16, respectively. Although the efficiency is not so high as the results on SSD, we can also get the same observation as above. Alstrup's data structure theoretically has high query efficiency, but it is not that usable due to its unacceptable write efficiency and space efficiency.

The results of write efficiency and query efficiency for original-datasets on SSD are shown in Figure 17 and Figure 18, respectively. There is an exception that in Figure 18(b), our WORQ-tree shows 1.2× slower than Bkd-tree. However, it is for a lower-dimensional and smaller-scale dataset. For the other larger and higher dimensional datasets, WORQ-tree keeps showing a 6.0× and 5.5× write efficiency and 1.6× and 1.4× query efficiency in Figure 17 and Figure 18.

Additional Experiment: Deletions

We also test the efficiency of deletions and find that there is no observable difference between all the mothods. We choose Chicago-large as the dataset, randomly perform 1000 point deletions and calculate the running time. Figure 19 shows the results. Since all these four methods are using the same LSM technique, they are

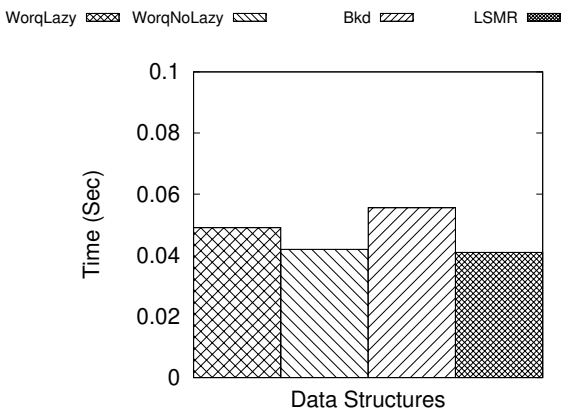


Figure 19: Deletion time for Chincago-large on SSD

supposed to perform the same on deletions. In addition, we focus on write-intensive workloads, so the deletion will not change the ranks of our methods and the competitors.