High Performance or Low Memory? A Time-Space Tradeoff Learned Index Framework Driven by Demand

Hui Wang
Tianjin University
Tianjin, China
wanghui025@tju.edu.cn

Xin Wang
Tianjin University
Tianjin, China
wangx@tju.edu.cn

Jiake Ge
Tianjin University
Tianjin, China
gejiake@tju.edu.cn

Yunpeng Chai
Renmin University of China
Beijing, China
ypchai@ruc.edu.cn

Lei Liang
Ant Group
Hangzhou, China
leywar.liang@antgroup.com

Peng Yi
Ant Group
Hangzhou, China
rongyan.yp@antgroup.com

Abstract

The first generation of learned indexes inherently achieved lower space overhead than traditional index structures, establishing this advantage as one of the pivotal research directions in index optimization. However, in their pursuit of peak performance, designers often significantly increase space overhead, which becomes infeasible in scenarios with limited storage space. Additionally, they cannot flexibly construct learned index structures based on varying time-space costs, making them less adaptable to application scenarios with different demands. For example, when space is limited, users cannot choose to sacrifice performance for lower space overhead to minimize total cost under given constraints, and vice versa. To address these issues, this paper proposes DESTO, a learned index framework that achieves time-space tradeoff by implementing an algorithm that minimizes time-space cost. Furthermore, DESTO supports the construction of time-space tradeoff structures under custom time-space cost weight or constraints. Evaluation results indicate that DESTO consistently achieves optimal time-space tradeoff under various workloads, datasets, and constraint conditions, outperforming other state-of-the-art learned indexes.

A Appendix

A.1 Existence of optimal error ϵ

As derived in the main text, the formula for the number of memory accesses is given by Equation 1. Here, K_{leaf} denotes the total number of keys stored at the leaf nodes, while α and ω are the harmonic parameters.

$$L(\epsilon) = \frac{\log(\alpha K_{leaf}) - \log(2\epsilon)}{\log(\frac{2\epsilon}{\alpha})} \left(1 + \log_2 \frac{2\epsilon}{\omega}\right) \tag{1}$$

This section aims to prove the existence of an optimal error ϵ that minimizes the number of memory accesses $L(\epsilon)$. To facilitate the analysis, let $x = \log(2\epsilon)$. Substituting this into Equation 1 yields Equation 2.

$$L(x) = \frac{\log(\alpha K_{leaf}) - x}{x - \log(\alpha)} \left(1 + \frac{x - \log(\omega)}{\log 2}\right) \tag{2}$$

To determine whether $L(x)$ has a minimum, the derivative of $L(x)$ is analyzed. The function $L(x)$ can be decomposed into two parts for differentiation:

$$A(x) = \frac{\log(\alpha K_{leaf}) - x}{x - \log(\alpha)} \tag{3}$$

$$B(x) = 1 + \frac{x - \log(\omega)}{\log 2} \tag{4}$$

Thus, $L(x) = A(x) \cdot B(x)$. The derivative of $L(x)$ is given by Equation 5.

$$L'(x) = A'(x)B(x) + A(x)B'(x) \tag{5}$$

The derivatives of $A(x)$ and $B(x)$ are computed as Equation 6 and Equation 7, respectively.

$$A'(x) = \frac{-(x - \log(\alpha)) - (\log(\alpha K_{leaf}) - x)}{(x - \log(\alpha))^2} = \frac{-\log(K_{leaf})}{(x - \log(\alpha))^2} \tag{6}$$

$$B'(x) = \frac{1}{\log 2} \tag{7}$$

Substituting Equation 6 and Equation 7 into Equation 5 yields Equation 8.

$$L'(x) = \frac{-\log(K_{leaf})}{(x - \log(\alpha))^2} \left(1 + \frac{x - \log(\omega)}{\log 2}\right) + \frac{\log(\alpha K_{leaf}) - x}{x - \log(\alpha)} \cdot \frac{1}{\log 2} \tag{8}$$

To find the critical points, the equation $L'(x) = 0$ is solved and rearranged as Equation 9.

$$\frac{-\log(K_{leaf})}{(x - \log(\alpha))^2} + \frac{-\log(K_{leaf})(x - \log(\omega))}{(x - \log(\alpha))^2 \log 2} + \frac{\log(\alpha K_{leaf}) - x}{(x - \log(\alpha)) \log 2} = 0. \tag{9}$$

Solving Equation 9 yields two distinct solutions, where Δ is a non-zero real number.

$$x_1 = -\frac{\log(K_{leaf})}{2} + \frac{\log(\alpha)}{2} + \frac{\log(K_{leaf}\alpha)}{2} - \frac{\sqrt{\Delta}}{2} \tag{10}$$

$$x_2 = -\frac{\log(K_{leaf})}{2} + \frac{\log(\alpha)}{2} + \frac{\log(K_{leaf}\alpha)}{2} + \frac{\sqrt{\Delta}}{2} \tag{11}$$

To determine the nature of the critical points, the second derivative $L''(x) = \frac{d}{dx}L'(x)$ is computed, and the results can be verified using symbolic computation software such as SymPy.

At $x = x_1$:

$$L''(x_1) > 0, \quad \text{indicating a local minimum.} \tag{12}$$

At $x = x_2$:

$$L''(x_2) < 0, \quad \text{indicating a local maximum.} \tag{13}$$

The analysis demonstrates that $L'(x) = 0$ has two solutions: x_1 (a local minimum) and x_1 (a local maximum). This confirms that $L(x)$ has a minimum at x_1, implying the existence of an optimal parameter x that minimizes the number of memory accesses. Furthermore, since $x = \log(2\epsilon)$ is a monotonically increasing function, there exists an optimal ϵ that minimizes the memory access counts.

A.2 Existence of optimal density d

As presented in the main text, the total cost function is defined by Equation 18.

$$COST = T_{total}(d)^{\gamma} \times S_{total}(d) \tag{14}$$

where $T_{total}(d)$ and $S_{total}(d)$ are given in Equation 15 and Equation 16, respectively.

$$T_{total}(d) = \beta L(d) + \sigma T_{CPU} \tag{15}$$

$$S_{total} = \sum_{i=1}^{H} \left(N_i \cdot S_{params} + K_i \cdot \left(S_{key} + \left(\frac{1}{d} - 1 \right) \cdot S_{empty} \right) \right) \tag{16}$$

Here, $L(d)$ is defined in Equation 17.

$$L(d) = \frac{\log(\alpha K_{leaf}) - \log(2\delta\epsilon d)}{\log(\frac{2\delta\epsilon d}{\alpha})} \left(1 + \log_2 \frac{2\delta\epsilon d}{\omega} \right) \tag{17}$$

The objective function $f(d) = T_{total}(d)^{\gamma} \times S_{total}(d)$ is decomposed into two components: $g(d) = T_{total}(d)^{\gamma}$ and $h(d) = S_{total}(d)$, such that $f(d) = g(d) \cdot h(d)$. Using the product rule, the first derivative is $f'(d) = g'(d) \cdot h(d) + g(d) \cdot h'(d)$.

To simplify the derivation, intermediate terms are defined as follows:

$$A = \log(\alpha K_{leaf}),\ B = \log(2\delta d),\ C = \log\left(\frac{2\delta d}{\alpha}\right),\ D = \log_2\left(\frac{2\delta d}{\omega}\right)$$

Using these definitions, $g(d)$ can be expressed as Equation 18.

$$g(d) = \left(\beta \cdot \frac{A - B}{C} \cdot (1 + D) + \sigma T_{CPU} \right)^{\gamma} \tag{18}$$

The derivative $g'(d)$ is then computed as Equation 19.

$$g'(d) = \gamma \cdot (E + \sigma T_{CPU})^{\gamma - 1} \cdot E' \tag{19}$$

where $E = \beta \cdot \frac{A-B}{C} \cdot (1 + D)$.

The derivative E' is calculated by applying the quotient rule to the term $\frac{(A-B)(1+D)}{C}$.

$$\frac{d}{dd}\left(\frac{(A-B)(1+D)}{C} \right) = \frac{(A-B)'(1+D) + (A-B)(1+D)'}{C} - \frac{(A-B)(1+D)C'}{C^2} \tag{20}$$

The individual derivatives are computed as follows:

1. $(A - B)' = -\frac{1}{d}$
2. $C = \log\left(\frac{2\delta d}{\alpha}\right) \Rightarrow C' = \frac{1}{d}$
3. $D = \log_2\left(\frac{2\delta d}{\omega}\right) \Rightarrow D' = \frac{1}{d \ln 2}$
4. $(1 + D)' = \frac{1}{d \ln 2}$

Substituting these derivatives into the expression for E', we obtain Equation 21.

$$E' = \beta \cdot \left(\frac{-\frac{1}{d}(1 + D) + \frac{A-B}{d \ln 2}}{C} - \frac{(A - B)(1 + D)}{dC^2} \right) \tag{21}$$

The derivative of $h(d)$ is computed as Equation 22.

$$h'(d) = \sum_{i=1}^{H} K_i \cdot \left(-\frac{S_{empty}}{d^2} \right) = -\frac{S_{empty}}{d^2} \sum_{i=1}^{H} K_i \tag{22}$$

Setting $f'(d) = 0$, the critical points satisfy Equation 23.

$$\gamma \cdot (E + \sigma T_{CPU})^{\gamma - 1} \cdot E' \cdot h(d) = g(d) \cdot \frac{S_{empty}}{d^2} \sum_{i=1}^{H} K_i \tag{23}$$

The existence of a critical point $d^* \in (0, 1)$ is established by analyzing the boundary behavior of $f(d)$. As $d \to 0^+$, $g(d)$ diverges to $+\infty$ due to the logarithmic singularity $\log(2\delta d) \to -\infty$ in $L(d)$. Simultaneously, $h(d)$ also diverges because $\frac{1}{d} - 1 \to +\infty$. Consequently, $f(d) = g(d) \cdot h(d)$ necessarily tends to $+\infty$.

As $d \to 1^-$, $g(d)$ converges to a finite value, since $\log(2\delta d) \to \log(2\delta)$ (a constant). $h(d)$ approaches $\sum_{i=1}^{H} \left(N_i \cdot S_{params} + K_i \cdot S_{key} \right)$, as $\frac{1}{d} - 1 \to 0$. Thus, $f(d)$ asymptotes to a finite constant.

Within the intermediate regime $d \in (0, 1)$, the function $f(d)$ exhibits monotonic decay from $+\infty$ to its finite asymptotic limit at $d \to 1^-$. By the Extreme Value Theorem, the continuity of $f(d)$ guarantees the existence of at least one critical point $d^* \in (0, 1)$ where the first derivative vanishes, i.e., $f'(d^*) = 0$.

To confirm that the critical point $d \in (0, 1)$ corresponds to a local minimum, the second derivative $f''(d)$ is analyzed. Applying the product rule to $f'(d) = g'(d) \cdot h(d) + g(d) \cdot h'(d)$, we derive $f''(d) = g''(d) \cdot h(d) + 2g'(d) \cdot h'(d) + g(d) \cdot h''(d)$.

Differentiating $g'(d)$, we obtain Equation 24.

$$g''(d) = \gamma(\gamma - 1)(E + \sigma T_{CPU})^{\gamma - 2} \cdot (E')^2 + \gamma(E + \sigma T_{CPU})^{\gamma - 1} \cdot E'' \tag{24}$$

Here, E'' is the second derivative of the intermediate term E, as shown in Equation 25.

$$E'' = \beta \cdot \frac{d}{dd}\left(\frac{-\frac{1}{d}(1 + D) + \frac{A-B}{d \ln 2}}{C} - \frac{(A - B)(1 + D)}{dC^2} \right) \tag{25}$$

Through algebraic manipulation, it can be shown that $E'' > 0$ due to the convexity of the logarithmic terms and the positivity of α, β, δ and K_{leaf}.

The second derivative of $h(d)$ is shown in Equation 26, and it is strictly positive for $d \in (0, 1)$, since $S_{empty} > 0$ and $K_i > 0$.

$$h''(d) = \frac{2S_{empty}}{d^3} \sum_{i=1}^{H} K_i > 0 \tag{26}$$

At the critical point d^*, where $f'(d^*) = 0$, the second derivative simplifies to $f''(d^*) = g''(d^*) \cdot h(d^*) + g(d^*) \cdot h''(d^*)$. Since all terms in the expression are strictly positive. Thus, $f''(d^*) > 0$, confirming

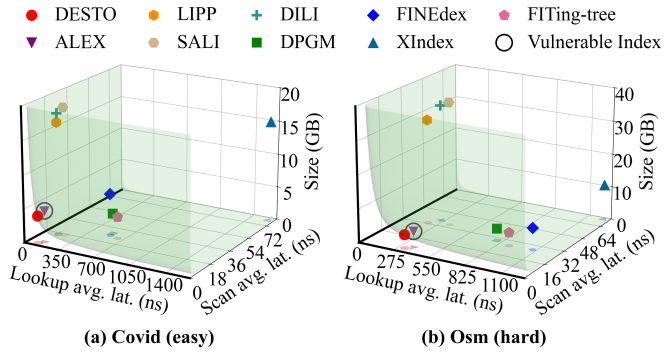

Figure 1: Lookup-scan-size tradeoff evaluation on the Covid/Osm dataset, where the number of scan is 1000.

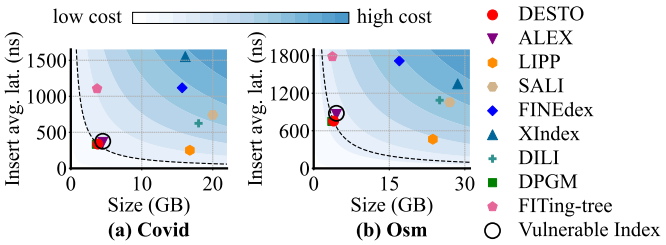

Figure 2: Insert-size tradeoff evaluation on the Covid/Osm dataset.

that d^* is a local minimum. This guarantees the existence of an optimal density $d^* \in (0, 1)$ that minimizes the total cost, achieving the desired time-space tradeoff.

A.3 The determination of the space factor β

When the leaf nodes of DESTO trigger a SMO, the expanded space $n.new\_size$ follows Equation 27.

$$n.new\_size = \beta \cdot \min\left(1, \frac{n.insert\_rate_t}{n.insert\_rate_{t-1}}\right) \cdot n.slot\_num \quad (27)$$

For the determination of the space factor β, see Equation 28.

$$\beta = \begin{cases} \gamma, & n.slot\_num \geq 1M \\ 3\gamma, & n.slot\_num \geq 500K \\ 6\gamma, & n.slot\_num < 100K \end{cases} \quad (28)$$

As indicated by Equation 28, nodes of varying sizes should be equipped with corresponding space factors, denoted as β. Equation 27 demonstrates that smaller nodes require larger space factors to ensure adequate space reservation. For example, suppose two hot nodes have slot capacities of 8 and 16, respectively; they would require expansion factors of 6 and 3 to achieve a total expansion of 48 units. The weight value of the space factor β can be dynamically adjusted based on different workloads.

A.4 Supplementary illustrations for single-threaded time-space evaluation

In Figure 1, we present the lookup-scan-size tradeoff for nine indexes on the Covid and Osm datasets in a single-threaded setup. As a supplementary experiment to the main text, DESTO consistently outperforms the other SOTA indexes, achieving the best tradeoff between lookup performance, scan efficiency, and space utilization, aligning with the conclusions presented in Insight 5.1.

In Figure 2, we present the insert-size tradeoff for nine indexes on the Covid and Osm datasets. This supplementary experiment confirms that DESTO achieves the best insert-size tradeoff, consistent with the findings discussed in Insight 5.1.

A.5 Supplementary illustrations for multi-threaded time-space evaluation

This section presents the lookup-size tradeoff (Figure 3(a)(d)), insert-size tradeoff (Figure 3(b)(g)), and balance-size tradeoff (Figure 3(c)(e)) for six multi-threaded indexes on the Covid and Osm datasets under a 64-thread setup. DESTO again demonstrates superior performance, corroborating the conclusions outlined in Insight 5.2.

A.6 Single-threaded DESTO deletion and update operations

To delete a key, a lookup operation is first performed to locate the position of the target key, followed by the removal of both the key and its associated payload. When the density of a node falls below a predefined threshold due to key deletions, the data node undergoes contraction to preserve space utilization efficiency.

The update operation is implemented as a sequential combination of deletion and insertion. The target key is first located through a lookup operation, followed by the deletion of the existing payload and the insertion of the new value.

A.7 Lightweight DESTO concurrency control implementation

This section describes the lightweight concurrency control implementation of the DESTO index, focusing on memory management, concurrent strategies for lookup and insertion operations, and the maintenance mechanism of the leaf node linked list, as well as deletion and update operations.

A.7.1 Memory management DESTO employs an epoch-based memory reclamation strategy to ensure safe memory deallocation in concurrent environments. Each worker thread maintains a local epoch counter, which is periodically synchronized with the global epoch. When a node is replaced, the old node is not immediately deallocated but is instead added to a deferred reclamation queue. The node is only deallocated once the epoch values of all active threads exceed the epoch of the node by at least two cycles. This mechanism avoids the risk of threads accessing deallocated memory while eliminating the need for global locks.

A.7.2 Lookup operation The lookup operation enables non-blocking reads through lightweight bit-flag locks in leaf nodes. Each leaf node contains an atomic flag, with specific bits indicating the lock state. Threads read the flag using the `__atomic_load_n` atomic instruction. If the node is not locked, the thread directly accesses the data; if the node is locked, the thread briefly waits and retries. This design ensures that read operations do not block other threads while maintaining data consistency.

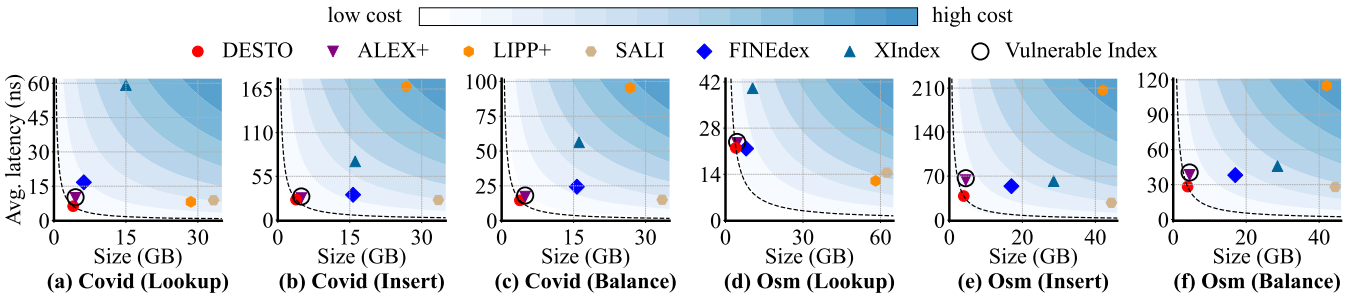

Figure 3: 64-thread time-space tradeoff evaluation on the Covid/Osm dataset.

A.7.3 Insertion operation Insertion operations ensure thread safety through atomic operations and a copy-and-replace mechanism. When the target leaf node is unlocked and the insertion cost is low, a thread attempts to acquire the lock via a CAS (Compare-And-Swap) operation. If successful, the thread inserts the data and releases the lock; if the CAS fails, the thread waits briefly and retries.

When a node is triggered for SMO adjustment, the thread creates a copy of the leaf node while holding the lock and performs structural updates atomically. The thread first creates a new copy of the current leaf node, reorganizes the original and new data into the copy, and performs operations such as expansion or splitting (see Section 4.1 in the main text for details). Subsequently, the thread atomically updates the parent node pointer using a CAS operation, replacing the original node with the new copy or split nodes. Concurrently, the leaf node linked list is updated atomically to ensure correctness during range query traversals. This design achieves efficient concurrency for insertion operations by ensuring that write threads are blocked only during the copy construction phase of a single leaf node, while read threads can continue to access the previous version, i.e., the original leaf node, thereby avoiding blocking and enhancing concurrency performance.

A.7.4 Deletion operation Deletion operations are implemented through atomic marking and deferred reclamation. A thread first attempts to set the deletion flag of the target leaf node using a CAS operation. If successful, the target key is marked as logically deleted, preventing subsequent read and write operations from accessing invalid data. Just like in a single-threaded approach, when the density of a node drops below a predefined threshold due to key deletions, the data node is contracted to ensure efficient space utilization. The mechanism guarantees atomicity and consistency without blocking normal access from other threads.

A.7.5 Update operation Update operations directly modify the value of the target node via the CAS instruction. When a thread initiates an update, it first looks up the current value of the node and then attempts to atomically replace the old value with the new value through a CAS operation. If the CAS succeeds, i.e., the value remains unmodified by other threads during the operation, the update is completed. If it fails, the thread retries the process, thereby ensuring the atomicity of update operations in concurrent environments.