Technical Report

Index Structure Choice for Range-Heavy Query Workloads

K. Almeida*[a,1] and D. Farrow[b]

[a] *Data Systems Group, Whitmore Institute*
[b] *Whitmore Institute*

This report was compiled on August 27, 2026

Abstract—We compare a B+-tree index against an LSM-tree index on a workload dominated by range scans over a time-ordered key space, the common shape of query traffic against an events or metrics table. At our tested write rate, the B+-tree gives 2.1× faster range-scan throughput at the cost of 3.4× slower point-insert latency. We recommend the B+-tree for this workload class specifically because range-scan latency, not insert latency, is what user-facing queries in this system are bound by.

Keywords—*index structures, B-tree, LSM-tree, range queries, technical report*

1. INTRODUCTION

Our events table receives roughly 4,000 inserts per second and is queried almost exclusively by time-range scans ("all events between t_1 and t_2 for account X"), rarely by single-key point lookups. The two dominant index structures for this shape of workload — B+-trees and LSM-trees — make different trade-offs between write throughput and range-scan latency, and the standard advice (LSM-trees for write-heavy workloads) does not by itself settle which is right here, since our workload is write-heavy in volume but latency-sensitive on reads. This report measures both directly against our actual query shape rather than relying on the general advice.

2. METHOD

We loaded 50 million synthetic events (16-byte keys, 128-byte values, monotonically increasing timestamps with account ID as a secondary sort key) into both a B+-tree index and a levelled LSM-tree index, then ran two benchmarks against each:

- **Range scan:** 10,000 range queries, each spanning a random 1-hour window, measuring wall-clock time to iterate the full result set.
- **Point insert:** 10,000 single-row inserts interleaved with background compaction (LSM-tree) or page splits (B+-tree), measuring per-insert latency including any compaction/split stalls attributable to that insert.

3. RESULTS

Table 1

Table 1. Median and p99 latency by index structure

	Range scan (ms)		Point insert (µs)	
Structure	Median	p99	Median	p99
B+-tree	4.2	9.8	340	1,850
LSM-tree	8.9	22.4	100	610

The B+-tree's range-scan advantage (2.1× at the median, 2.3× at p99) comes from data locality: a time-ordered B+-tree keeps a scan's result set in physically adjacent leaf pages, whereas the LSM-tree must merge results across multiple sorted runs even for a single contiguous key range. The LSM-tree's insert advantage (3.4× at the median) comes from the reverse property: writes are sequential appends to a memtable, deferring the cost of maintaining sort order to background compaction, whereas the B+-tree pays a page-split cost inline on the insert path whenever a leaf fills.

4. DISCUSSION

Neither result is surprising in isolation — this is the textbook trade-off both structures are known for. What this report adds is the workload-specific answer: for our actual traffic (read-bound by range scans, write volume high but not latency-sensitive per insert), the B+-tree's range-scan advantage matters far more to user-facing latency than the LSM-tree's insert advantage does, since inserts here are a background ingestion pipeline with no user waiting on individual insert latency, while every dashboard query is a user-facing range scan.

A caveat: this result is specific to a workload whose keys are already time-ordered at write time, giving the B+-tree's locality advantage the best possible conditions. A workload with random-order keys and the same range-query pattern would narrow this gap considerably, since a B+-tree loses its locality advantage once range-scanned keys are no longer physically adjacent regardless of insertion order.

5. CONCLUSION

For range-scan-dominated, time-ordered query workloads where insert latency is not user-facing, a B+-tree index measurably outperforms an LSM-tree on the metric that matters. We are migrating the events table's primary index to a B+-tree on this basis.

*k.almeida@example.org

Internal technical report. No errors were found during compilation.