Increasing Throughput with Concurrency Control for ANN Indexes

Collin Drake
University of Colorado Boulder
collin.drake@colorado.edu

Abstract

Implementing thread-safety in ANN indexes presents a variety of design choices. Ultimately, the tradeoffs converge to two axes: locking granularity and conflict resolution. In IVFFlat, for example, locking can be applied to each bucket; while effective, this approach is coarse-grained. Conversely, HNSW allows for locking at the finer granularity of a node's neighbor list. In both architectures, conflicts may be handled optimistically or pessimistically. In this paper, we implement and evaluate the cross-product of these strategies.

1. Introduction

As vector databases increasingly support streaming workloads, the need for concurrent queries against continuously updated indexes has become apparent. While approximate nearest neighbor (ANN) search has been extensively studied under the assumption of a static index [1], the concurrency control mechanisms required for dynamic workloads remain underexplored. Existing systems either sacrifice read-write concurrency entirely or rely on batch processing strategies incompatible with streaming insertions. This paper systematically examines the design space of concurrency control for ANN indexes, focusing on the interplay between locking granularity and conflict resolution strategies.

Existing approaches to parallel ANN index construction fall short of these requirements. FAISS, a well-known example of fine-grained locking, utilizes this locking strictly for parallel index construction (protecting concurrent writers), rather than for isolation between readers and writers. Consequently, it does not support simultaneous search and update. Furthermore, while FAISS exploits thread-level parallelism to batch-process search queries, the search path itself is completely lock-free [2].

ParlayANN takes a different approach, offering two strategies for parallel construction. Its deterministic batch construction uses prefix sums and parallel primitives to process points in a fixed order, ensuring reproducible results across runs. For incremental updates, it employs lock-free batch insertion using compare-and-swap operations, allowing threads to modify neighbor lists without acquiring locks [3]. However, neither technique is well-suited for online systems. Prefix sum-based construction requires the entire batch of points to be known upfront—offsets must be computed before any insertions can proceed, making it fundamentally incompatible with a stream of arriving vectors. Similarly, batch conflict scheduling assumes all operations are available for analysis before execution, enabling the scheduler to order them to minimize conflicts. In an online setting, there is no batch to analyze. The system must handle each operation as it arrives while maintaining consistency with concurrent operations already in flight.

JVector implements lock-free concurrent construction but is not benchmarked on standard suites, making performance comparisons difficult [4]. And neither FAISS's parallel construction nor ParlayANN's batch strategies address the online read-write concurrency problem we examine.

This gap motivates our systematic exploration of the concurrency control design space for online ANN indexes. We identify two key dimensions along which strategies may vary: locking granularity (coarse-grained at the partition or bucket level vs. fine-grained at the individual neighbor list level) and conflict resolution policy (optimistic with validation and retry vs. pessimistic with blocking). The cross-product of these dimensions yields four distinct strategies. We implement each strategy for both IVFFlat and HNSW indexes and evaluate their performance under mixed read-write workloads. Our evaluation provides the first empirical comparison of concurrency control approaches for streaming vector search.

2. Methodology

3. Results

4. Discussion

- ***Why hasn't this been done before?*** That's an excellent question. Techniques as trivial for database concurrency control as these should have been implemented long ago. Well, one possiblity is that for most applications these techniques are unnecessary; most ML workflows simply don't require this level of concurrency control. Another possiblity is that these indexes have had *relatively* short times spent in the research limelight.

5. Conclusion

Bibliography

[1] L. Ma *et al.*, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge." [Online]. Available: https://arxiv.org/abs/2310.11703

[2] Meta Platforms, Inc., "FAISS: IndexHNSW Implementation." [Online]. Available: https://github.com/facebookresearch/faiss/blob/main/faiss/IndexHNSW.cpp

[3] M. D. Manohar *et al.*, "ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms." [Online]. Available: https://arxiv.org/abs/2305.04359

[4] DataStax, "JVector: The Most Advanced Embedded Vector Search Engine." [Online]. Available: https://github.com/datastax/jvector