Milvus: A Purpose-Built Vector Data Management System

SIGMOD 2021

Jianguo Wang*, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, Charles Xie

Zilliz & Purdue University

Presented By:
Puxuan Wang, Anubhav Mathur, Tanmay Sutar, Ignacio B Di Leva

Georgia Tech

Agenda

- Introduction & System Design
- Heterogeneous Computing for Milvus
- Advanced Query Filtering
- System Implementation
- Applications
- Evaluation
- Related Work
- Conclusion

Georgia Tech

Motivation

- The exponential growth of high-dimensional vector data, driven by unstructured data such as images, videos, and text, has significantly increased the demand for efficient vector data management.

- With 80% of global data projected to be unstructured by 2025, a robust vector data management system is essential.

Challenges

Performance challenges

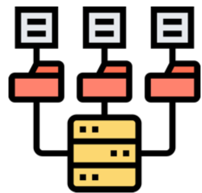
Handling large-scale and dynamic vector data efficiently is often infeasible.

Functionality constraints

Advanced query processing capabilities, such as attribute filtering and multi-vector queries, are rarely supported.

Table 1: System comparison

	Billion-Scale Data	Dynamic Data	GPU	Attribute Filtering	Multi-Vector Query	Distributed System
Facebook Faiss [3, 35]	✓	✗	✓	✗	✗	✗
Microsoft SPTAG [14]	✓	✗	✗	✗	✗	✗
ElasticSearch [2]	✗	✓	✗	✓	✗	✓
Jingdong Vearch [4, 39]	✗	✓	✓	✓	✗	✓
Alibaba AnalyticDB-V [65]	✓	✓	✗	✓	✗	✓
Alibaba PASE (PostgreSQL) [68]	✗	✓	✗	✓	✗	✗
Milvus (this paper)	✓	✓	✓	✓	✓	✓



Distributed System challenges

Systems like Facebook Faiss and Microsoft SPTAG lack support for distributed data management and dynamic updates.

Milvus

- Full support for billion-scale data, dynamic vector data, and distributed architecture.
- GPU optimizations for enhanced performance and compatibility with modern computing platforms.
- Comprehensive query processing capabilities, including attribute filtering and multi-vector queries.

Georgia Tech

System Design

- **Query Engine:** Optimized for modern CPUs with reduced cache misses and SIMD instruction utilization, it supports vector similarity search, attribute filtering, and multi-vector queries.
- **GPU Engine:** Accelerates computations with multi-GPU support and hybrid indexing, integrating CPU and GPU strengths.
- **Storage Engine:** Implements dynamic vector data management using the LSM-tree structure and supports various storage solutions like Amazon S3, HDFS, and local file systems.

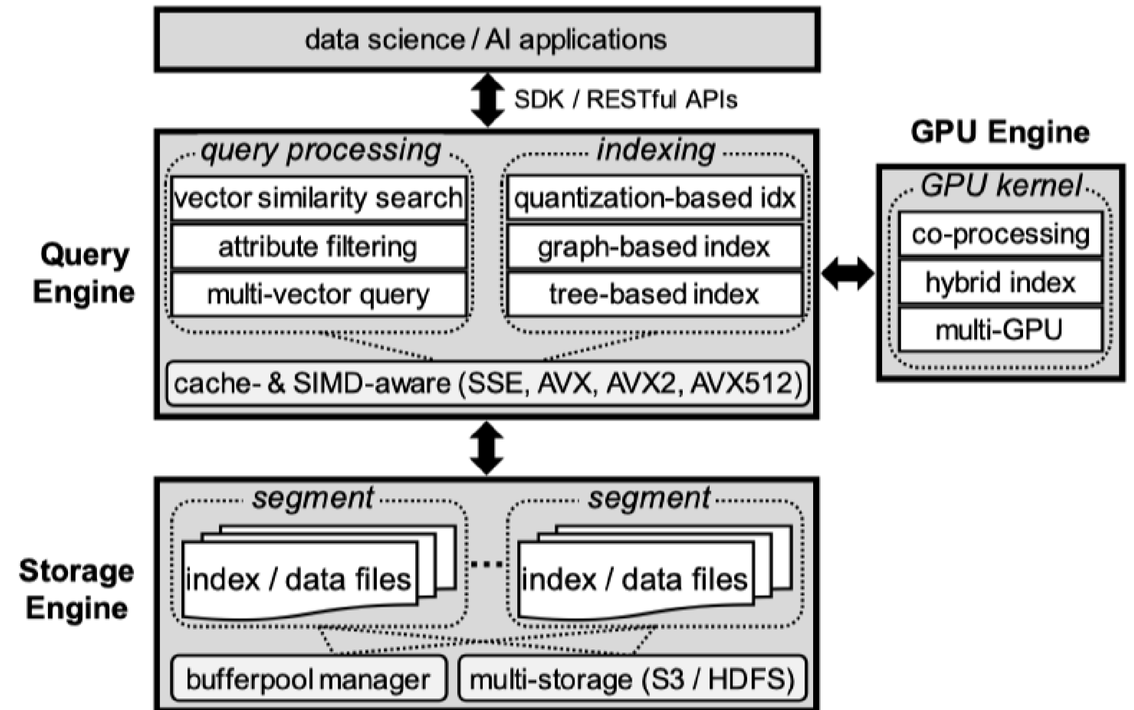


Figure 1: System architecture of Milvus

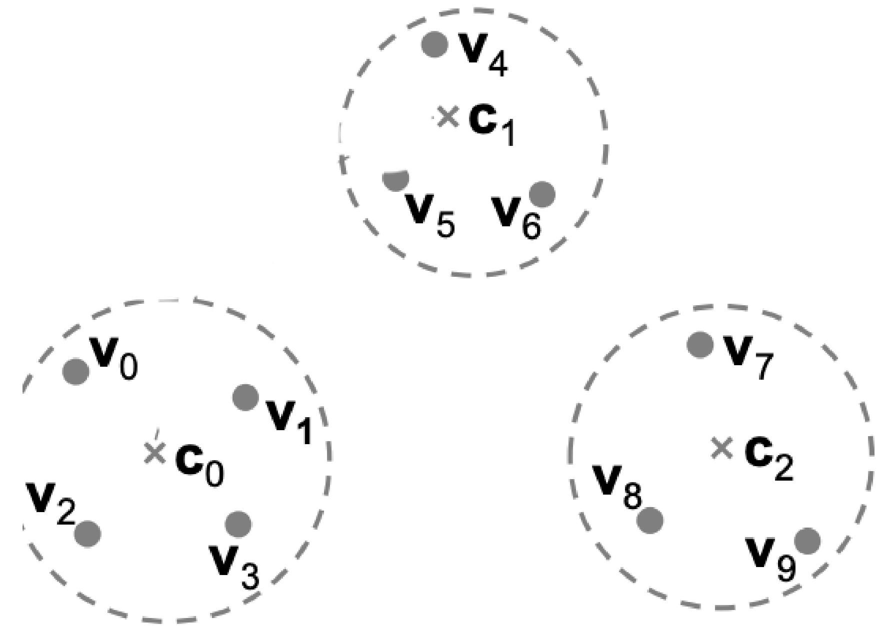
Georgia Tech

Heterogeneous Computing for Milvus

Georgia Tech

Vector Quantization

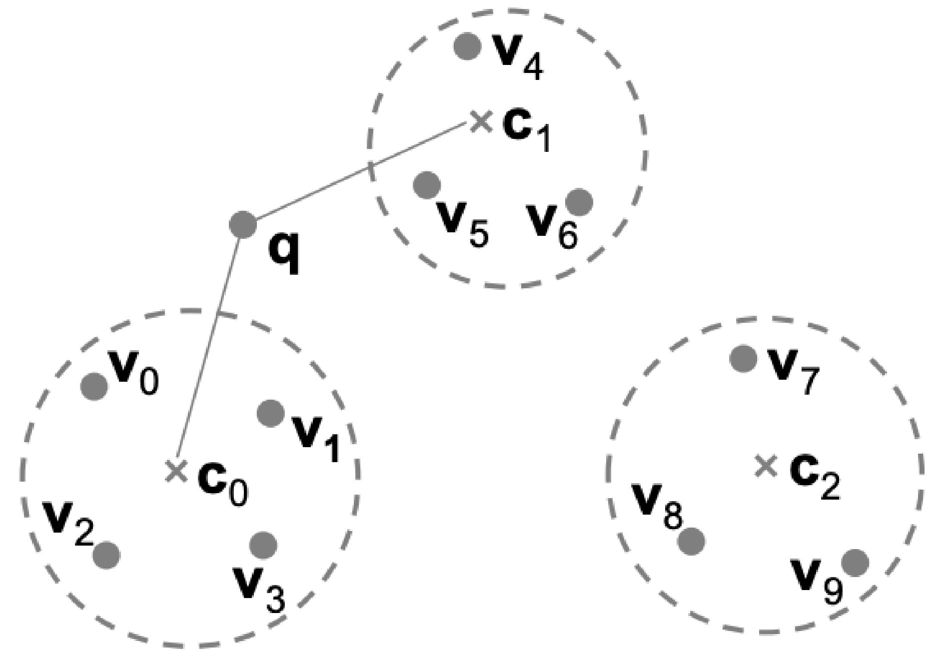
- Process that simplifies high-dimensional vectors by mapping them to **codewords** from a predefined **codebook** (K-Means)
  - The data space is divided into clusters (e.g., with centroids c_0, c_1, c_2).
  - Each vector (e.g., v_0, v_1, … ,v_9) is mapped to the nearest centroid of the cluster it belongs to.



Georgia Tech

Querying in Quantization based Indexes

- **Find Closest Centroids (Coarse Search)**:
  - The query vector q is compared to the centroids to identify the nearest n_{probe} clusters.
  - n_{probe} controls the balance between search accuracy and speed
- **Search Within Clusters (Fine Search)**:
  - After identifying relevant clusters, the query searches within the vectors of those clusters using the fine quantizer's representation.
  - Different indexes use different fine quantizers:
    - **IVF_FLAT**: Keeps the original vectors for comparisons.
    - **IVF_SQ8**: Compresses the vector values (e.g., 4-byte floats into 1-byte integers).
    - **IVF_PQ**: Splits each vector into sub-vectors and applies quantization within each



q: Return top k similar vectors

Georgia Tech

Inefficient Use of CPU Cache in Query Processing

- **Faiss Approach**: Threads process one query at a time, streaming the entire dataset through CPU caches.
- Data cannot be reused for subsequent queries.

Key Issues:

1. **Frequent Cache Misses:**
    - L3 cache (10s of MB) is underutilized as data is repeatedly loaded from main memory.
    - High latency due to excessive memory access.
2. **Underutilized Multi-Core CPUs:**
    - Threads only process individual queries, leaving CPU cores underloaded, especially for small query batches.

Georgia Tech

Milvus Cache Aware Optimization

- **Partitioning Data Vectors**:
  - Dataset is divided into **chunks** (b=n/t) and assigned to threads.
  - Each thread loads its data chunk into **L3 cache**, reusing it across multiple queries to reduce memory access.
- **Query Blocking**:
  - Queries are grouped into **blocks (s)** that fit entirely into **L3 cache.**
  - All queries in a block reuse the same cached data vectors.
- **Heap Per Query Per Thread**:
  - Each thread maintains a **separate heap** for every query in the block.
  - **Heaps store the top-k results for each query.**
  - After processing, these per-thread heaps are **merged** to produce the final top-k results per query.

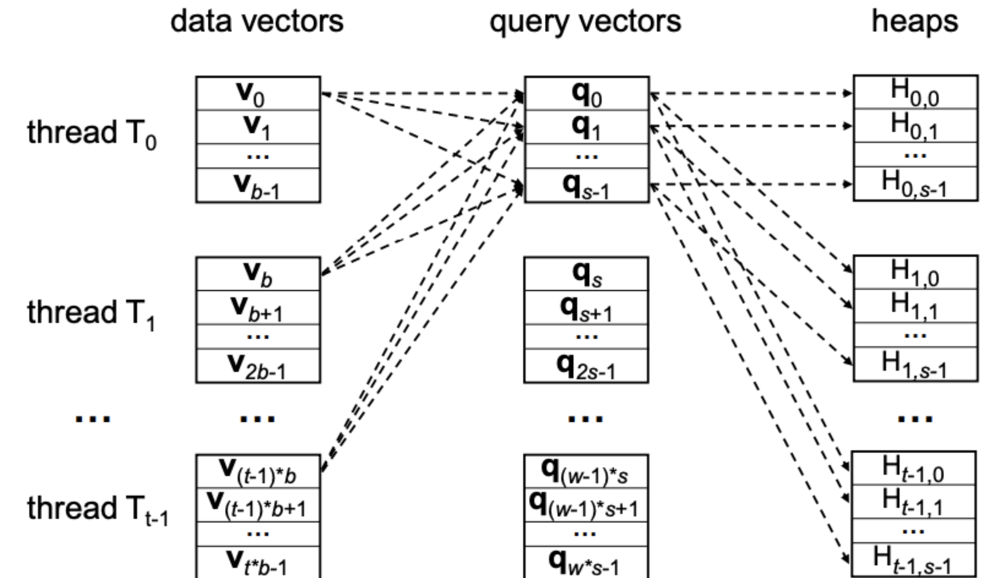


Figure 3: Cache-aware design in Milvus

Georgia Tech

Milvus GPU Optimizations

Optimization	Problem in Faiss	Milvus Solution	Benefits
Supporting Larger k	Faiss supports $k \leq 1024$ only.	Round-based query processing to support $k > 1024$.	Enables applications like video surveillance and recommendation systems requiring $k > 1024$.
Multi-GPU Support	GPU count must be declared at compile time.	Dynamic GPU selection with segment-based scheduling.	Flexible deployment on servers with varying GPU resources.

Georgia Tech

GPU and CPU Co-Design in Milvus

- **CPU is better for:**
  - Tasks with **scattered I/O**, such as scanning vectors within buckets.
  - Handling smaller query batch sizes where GPU overhead is significant.
- **GPU is better for:**
  - **Parallel computation-intensive tasks**, such as comparing queries against centroids.
  - Processing large query batch sizes, where data transfer overhead is amortized.

Milvus proposes SQ8H (Hybrid IVF_SQ8): A hybrid algorithm that dynamically partitions the query workload between GPU and CPU.

- **GPU** handles centroid comparisons (Step 1), and **CPU** handles bucket scanning (Step 2) for small query batches.
- For large query batches, the entire workload is processed on the GPU with efficient data transfers.

Algorithm 1: SQ8H

1. let n_q be the batch size;
2. **if** $n_q \geq$ *threshold* **then**
3. run all the queries entirely in GPU (load multiple buckets to GPU memory on the fly);
4. **else**
5. execute the step 1 of SQ8 in GPU: finding n_{probe} buckets;
6. execute the step 2 of SQ8 in CPU: scanning every relevant bucket;

Georgia Tech

Advanced Query Filtering

Georgia Tech

1. Attribute Filtering

It considers attribute constraints (Ca) and vector constraints (Cv) where Ca is within a range [p1, p2].

Strategy A: attribute-first-vector-full-scan

- Uses Ca to get entities
- Scans these entities to get top-k results
- Only useful when Ca is highly selective.

Strategy B: attribute-first-vector-search

- Uses Ca to get entities
- makes bitmap of resultant entity IDs.
- Checks Cvs against bitmap to get top-k results.
- Suitable when Ca or Cv is moderately selective.

Strategy C: vector-first-attribute-full-scan

- Uses Cv to get entities
- Scans these entities to check for Ca.
- Only useful when Cv is highly selective.

Strategy D: cost-based

- Estimates cost of strategies A,B and C and picks the least cost strategy.

Georgia Tech

Strategy E: Partition based strategy for Attribute Filtering

- Divides the dataset into partitions based on frequently searched attributes.
- Example: Dataset split into 5 partitions for price ranges:

  P0 [1-100], P1 [101-200], P2 [201-300], etc.
- For each query, only search partitions whose ranges overlap with the query's attribute filter.
- If a partition is fully covered by the query range, skip attribute checks and focus only on vector search.
- Partitions are created offline based on historical data.
- The number of partitions (ρ) is user-configurable

2. Multi-Vector Queries:

Vector Fusion Approach

- Concatenate vectors for each entity: [e.v0,e.v1,...,e.vμ−1] [e.v0 ,e.v1 ,...,e.vμ−1].
- Aggregate query vectors using g, e.g., weighted sum: [w0×q.v0,w1×q.v1,...,wμ−1×q.vμ−1][w0 × q.v0 ,w1 ×q.v1 ,...,wμ−1 ×q.vμ−1].
- Search aggregated query vector against concatenated vectors in dataset.

Iterative Merging Approach

- Iteratively query top-k' vectors for each q.v_i on D_i.
- NRA algorithm is used over results to determine top-k entities.
- If top-k results are not fully determined, increase k'.
- No reliance on getNext() for efficient indexing.
- Adaptive k' to control the number of steps.

Georgia Tech

System Implementation

System Implementation

Asynchronous Process

- Offloads write requests and index building to background
- This makes it responsive to new queries and inputs while also processing past data in the background.

Snapshot Isolation

- Milvus manages dynamic data following the LSM-style: new data inserted to memory first and then flushed to disk as immutable segments.
- Each segment has multiple versions and a new version is generated whenever the data or index in that segment is changed.

Georgia Tech

System Implementation

Distributed System

- **Shared-Storage Architecture**: Separates computing and storage for elasticity, leveraging Amazon S3 for high-availability storage.
- **Three-Layer Design**:
  - **Storage Layer**: Stores data on S3.
  - **Computing Layer**: Stateless layer with a single writer for updates and multiple readers for queries.
  - **Coordinator Layer**: Handles metadata, sharding, and load balancing using Zookeeper for high availability.
- **Optimizations for Performance**:
  - Logs (not raw data) are sent to storage to reduce network overhead.
  - Local caching on SSDs and memory minimizes frequent S3 access.
- **Scalability and Reliability**:
  - Kubernetes manages instances for auto-scaling and crash recovery.
  - Write-ahead logging ensures data consistency during writer crashes.

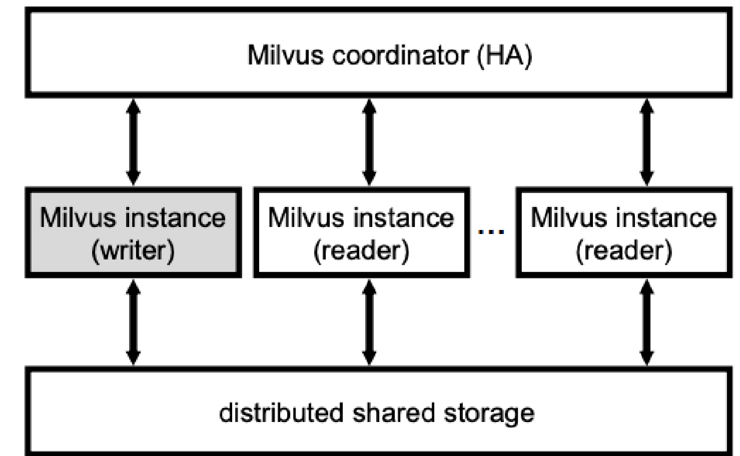


Figure 5: Milvus distributed system

Georgia Tech

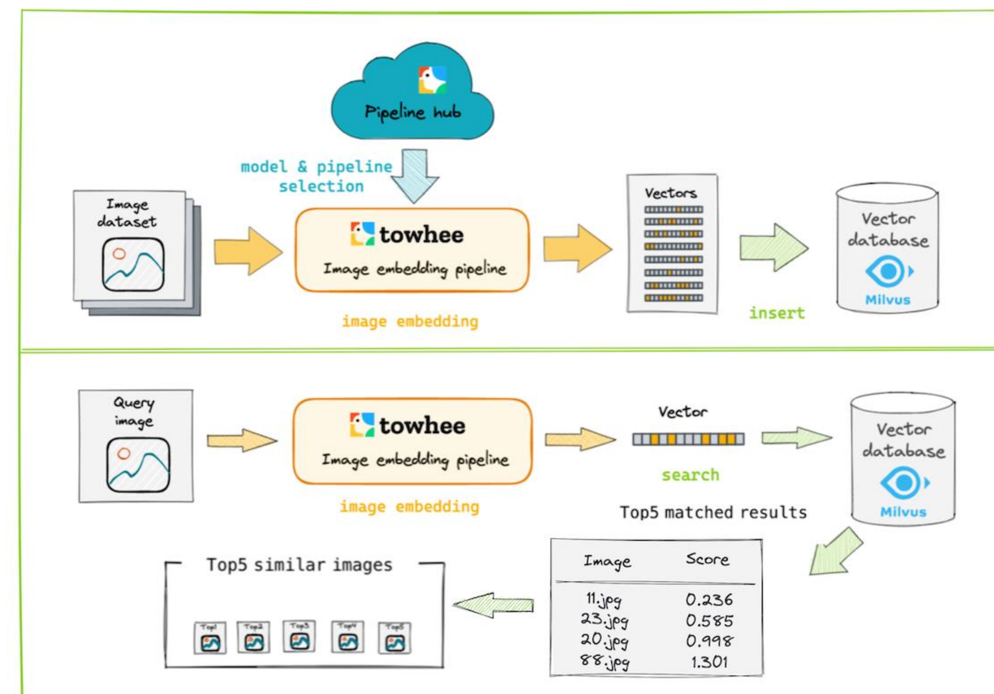
Applications

Georgia Tech

Applications

- **Image Search**: Images are converted to vectors using deep learning, such that similar images tend to produce similar vectors. Images are stored with their vectors so that one can query to obtain similar images.

- **Chemical Structure Analysis**: Chemical structures are encoded to high-dimensional vectors, and with vector similarity queries, one can find similar chemical structures that can help understand a particular structure.



Source: https://github.com/milvus-io/bootcamp

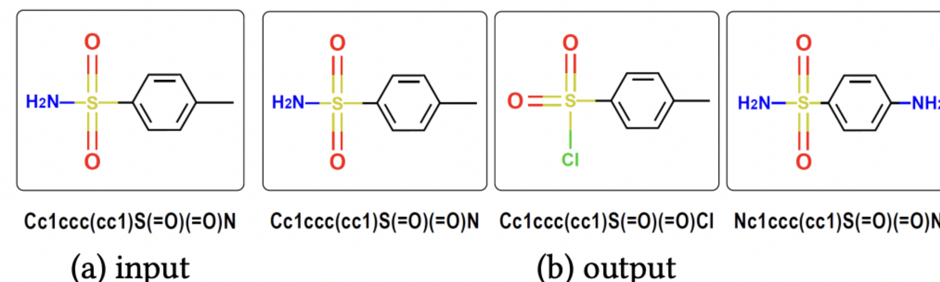


Figure 7: Milvus for chemical structure analysis

Georgia Tech

Evaluation

Evaluation: Comparison with Prior Systems

- **Evaluation metrics**: recall to evaluate the accuracy of some top-k queries, as well as throughput (Queries Per Second) of randomized queries.
- **Competitors**: Two open source systems, Jingdong Vearch and Microsoft SPTAG, as well as three anonymized commercial ones (Systems A, B, C).
- **Datasets**: Based on public datasets SIFT1B and Deep1B, but truncated to facilitate building indexes on prior systems.
- **Result**: Milvus is the most performant at each recall threshold.

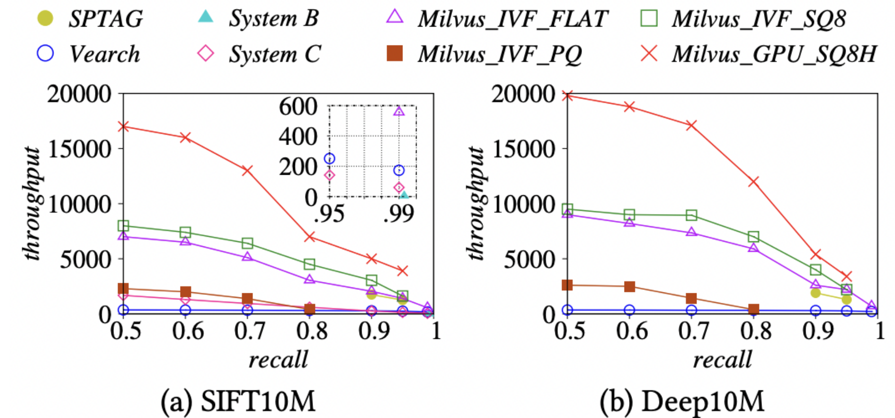


Figure 8: System evaluation on IVF indexes

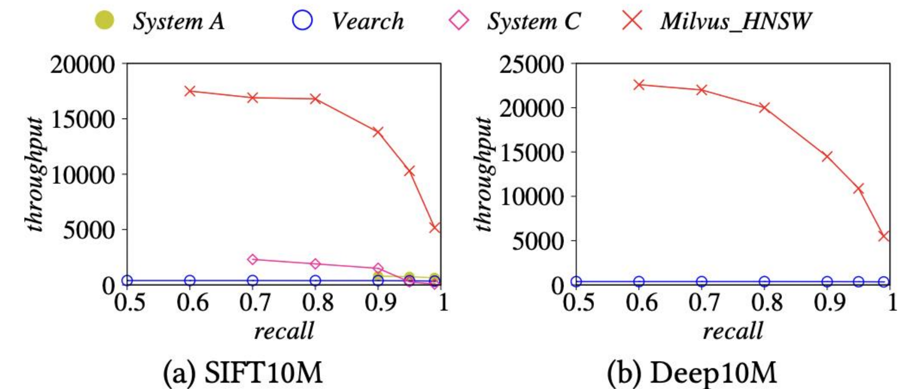
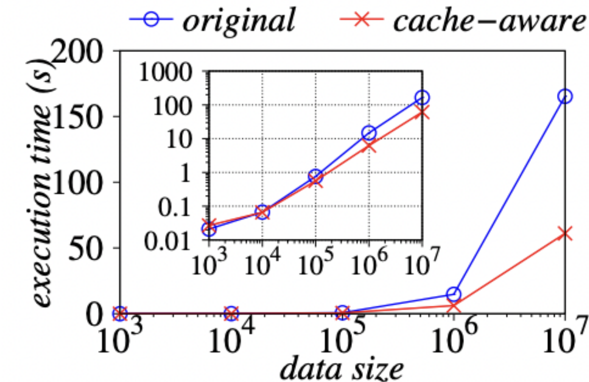


Figure 9: System evaluation on HNSW indexes

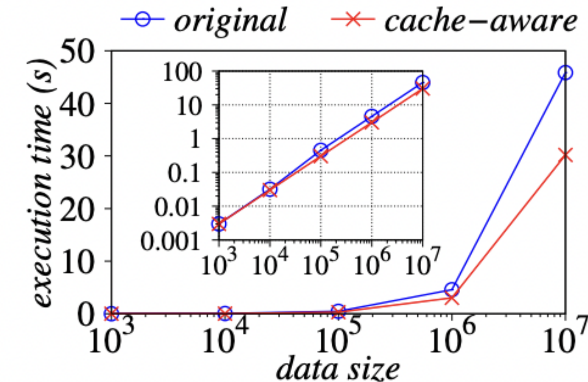
Georgia Tech

Evaluation: Optimizations

- **Cache-Aware Design**: The authors analyze the impact of their cache-aware implementation against the original implementation in Faiss and confirmed the speedup.
- **SIMD Optimizations**: The authors added AVX512 SIMD support, which isn't fully supported in Faiss, and confirmed the speedup for large amounts of data.
- **Hybrid Algorithm SQ8H**: The authors show that their hybrid algorithm is better than SQ8 in full-CPU or full-GPU, highlighting the benefits of heterogeneous computing.



(a) 12MB L3 cache



(b) 35.75MB L3 cache

Figure 11: Evaluating the cache-aware design

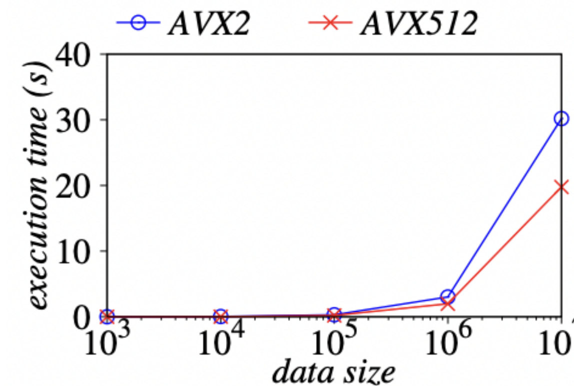


Figure 12: SIMD optimizations

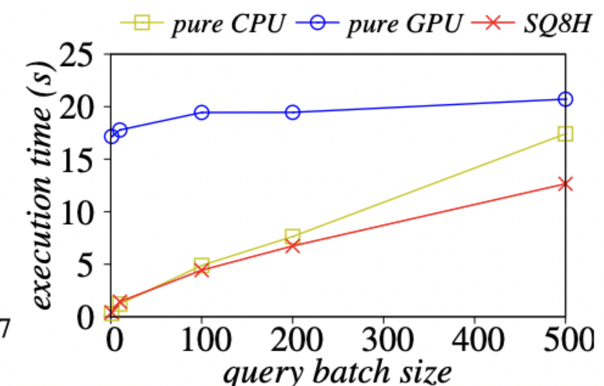


Figure 13: GPU indexing

Georgia Tech

Related Work

Georgia Tech

Related Work and Key Difference of Milvus

- Vector similarity search has already been extensively studied in both approximate search and exact search variants. Milvus focuses on approximate search.
- Prior works on approximate search can be classified into four categories:
  - LSH-based: Uses Locality Sensitive Hashing for Approximate Nearest Neighbor.
  - Tree-based: Uses data structures like k-d trees and random projection trees.
  - Graph-based: Uses approximation graphs to find the nearest neighbors in an original graph.
  - Quantization-based: Explored in presentation, includes Faiss and some Milvus indexes.
- A key difference between the related work and the Milvus work is that Milvus is a **vector data management system**, and it is composed of many engines (query, GPU, storage); it is more than one index or one search algorithm. It is also the first scalable system that specializes in vectors.

Georgia Tech

Conclusion

Conclusion

- Milvus addresses the problem of searching large-scale vector data, which has applications in data science and AI applications.
- The work identifies issues in systems that are not specialized or not prepared for large-scale vector data input. It proposes and implements a solution using many computing techniques.
- The authors show applications of their vector data management system and performance improvements over prior systems.

Study Questions

Study Questions

1. How does Milvus address the limitations of GPU memory and PCIe bandwidth in its GPU and CPU co-design, and why is the hybrid SQ8H algorithm effective for balancing computation and I/O between GPU and CPU?

2. What does it mean for Milvus to be cache-aware? How does Milvus leverage the L3 cache for its processing in CPU?

Georgia Tech