Scalable Overload-Aware Graph-Based Index Construction for 10-Billion-Scale Vector Similarity Search

Yang Shi
shiyang1@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Yiping Sun*
sunyiping@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Jiaolong Du
jiaolong@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Xiaocheng Zhong
mingcheng1@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Zhiyong Wang
sunzhenghuai@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Yao Hu
xiahou@xiaohongshu.com
Xiaohongshu Inc
Shanghai, China

Abstract

Approximate Nearest Neighbor Search (ANNS) is essential for modern data-driven applications that require efficient retrieval of top-k results from massive vector databases. Although existing graph-based ANNS algorithms achieve a high recall rate on billion-scale datasets, their slow construction speed and limited scalability hinder their applicability to large-scale industrial scenarios. In this paper, we introduce **SOGAIC**, the first **S**calable **O**verload-Aware **G**raph-Based **A**NNS **I**ndex **C**onstruction system tailored for ultra-large-scale vector databases: 1) We propose a dynamic data partitioning algorithm with overload constraints that adaptively introduces overlaps among subsets; 2) To enable efficient distributed subgraph construction, we employ a load-balancing task scheduling framework combined with an agglomerative merging strategy; 3) Extensive experiments on various datasets demonstrate a reduction of 47.3% in average construction time compared to existing methods. The proposed method has also been successfully deployed in a real-world industrial search engine, managing over 10 billion daily updated vectors and serving hundreds of millions of users.

CCS Concepts

- **Information systems** → **Search engine indexing**; **Web search engines**; *Database management system engines*; • **Computing methodologies** → *Distributed computing methodologies*.

Keywords

Approximate Nearest Neighborhood Search, ANNS Graph Indexing, Data Partitioning for Distributed Computing System, System and Resource Scalability

*Corresponding Author.

1 Introduction

Approximate Nearest Neighbor Search (ANNS) has been extensively studied in many recent research [13, 15, 20] due to its crucial role in a wide range of applications, such as web search engines, recommendation systems, and Retrieval-Augmented Generation (RAG) [11] for large language models (LLMs). These applications rely heavily on ANNS for efficient retrieval of top-k results. Among all ANNS algorithms, graph-based ones [22] such as HNSW[14], NSG[7], and NGT[9] have emerged as highly effective paradigms due to their ability to represent neighbor relationships in a graph structure, which significantly reduces the computational burden required for high-quality retrieval, especially comparing to space-partitioning methods such as IVF, KD-tree [19], and LSH [8]. Recently, approaches like DiskANN [10] have been developed to construct SSD-resident ANNS graph indexes for billion-scale search, minimizing disk I/O while using limited main memory. However, their index construction process, based on divide-and-conquer [3], is slow and hard to scale, hindering daily or more frequent updates of vector embeddings in real-world applications. The difficulty arises from two key challenges:

Firstly, efficiently creating overlapped subset divisions for large datasets with overload-aware considerations remains unexplored. DiskANN introduces overlap among subsets by assigning each vector to a fixed number of subsets based on its distance to centroids from clustering. However, this approach often results in redundant overlap between geometrically close divisions and sometimes fails to ensure balanced vector assignments due to the uneven distribution of points in the clusters, a consequence of K-means clustering with limited samples. This imbalance can cause a subset's subgraph construction to exceed resource limits, also known as overload issues, influencing the overall process. While density-based methods like DBSCAN offer more precise divisions, they are impractical for large datasets due to high computational costs, parameter sensitivity, and centroid control challenges. Thus, a more adaptive data
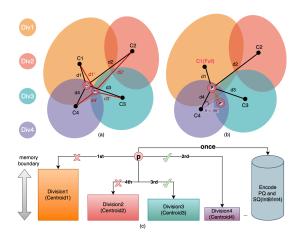
Figure 1: (a) Illustration of the assignment process for P1 and P2 based on their geometric relationships with centroids {C1 ... C4}. (b) Illustration of the deviation from P to P' for the assignment process of P, when overload happens on division1. (c) Illustration of the overall data partitioning process for P with quantization encoding executed in parallel.

partitioning approach is needed to reduce redundant assignments while keeping the overload boundary in check.

Secondly, an efficient and scalable scheduling framework for building and merging ANNS subgraphs remains unaddressed. Existing methods, such as DiskANN, often rely on a single high-cost machine to sequentially build and merge subgraphs, limiting scalability and efficiency. To address this, SPTAG [4] employs a scalable k-NN graph construction method [21] to partition datasets into non-overlapping subsets and distribute tasks across a cluster. However, as datasets grow in size and complexity, optimizing subgraph overlap for connectivity requires extensive iterations, leading to redundant computations and storage overhead. Observing that subgraph construction time is proportional to the size of its subset, an efficient scheduling framework can be designed to minimize construction time on a cluster without additional redundant computations, provided that overlap coverage and overload constraints are carefully addressed by a cost-effective data partitioning approach.

To address these challenges, we introduce **SOGAIC**, the first **S**calable **O**verload-Aware **G**raph-Based **A**NNS **I**ndex **C**onstruction system, designed for ultra-large-scale vector databases exceeding 10 billion points. **SOGAIC** accelerates index construction through an adaptive data partitioning strategy coupled with a load-balancing task scheduling framework, ensuring scalability with computational resources while maintaining high-quality graph structures for efficient vector similarity search.

2 System Overview

The proposed system consists of two main stages. The first stage handles overlapped subsets division using an adaptive data partitioning algorithm while controlling overload bounds. The second stage schedules subgraph construction tasks across a distributed cluster and executes the agglomerative merging strategy to form the final ANNS graph index.

Algorithm 1 Adaptive Overload-Aware Vector Assignment

Require: Vector to be assigned $\mathcal{V}$
Require: Centroids from K-means $c_i \in C$
Require: Max overlapping factor $\Omega \geq 2$
Require: Adaptive relaxation parameter $\epsilon > 1$
Require: Initialized sets for all centroids $s_i \in \mathcal{S}$
Require: Maximum assignable vectors of a set Γ

1: **Initialize:** Create a priority queue Q<**CentroidIndex, Dist**>
2: **for** $i = 0$ **to** $|C| - 1$ **do**
3:     Insert $(i, \text{distance}(\mathcal{V}, c_i))$ into Q
4: **end for**
5: **Initialize:** $curOLPCnt \leftarrow 0$; $curOLPFactor \leftarrow 0$
6: **Initialize:** $accDist \leftarrow 0$; $curAVGDist \leftarrow \infty$
7: **while** Q is not empty **and** curOLPCnt $< \Omega$
8:     $(\textbf{index}, \textbf{dist}) \leftarrow \textbf{ExtractMin}(Q)$
9:     **if** dist $<= \epsilon * $ curAVGDist
10:        $curOLPFactor \leftarrow curOLPFactor + 1$
11:        $accDist \leftarrow accDist + dist$
12:        $curAVGDist \leftarrow accDist/curOLPFactor$
13:        **if** $|s_{index}| < \Gamma$
14:           $curOLPCnt \leftarrow curOLPCnt + 1$
15:           $s_{index} \leftarrow s_{index} \cup \{\mathcal{V}\}$
16:        **else**
17:           $curAVGDist \leftarrow \infty$
18:        **end if**
19:     **end if**
20: **end while**

2.1 Overload-Aware Adaptive Data Partitioning

The algorithm for partitioning the vector database into multiple overlapping subsets begins by determining the maximum number of base points per division, referred to as Γ (capacity), which is constrained by the memory limits of each container in the computational cluster. Given a dataset containing N (total points) and a maximum overlapping factor Ω (defining the maximum number of partitions a point can belong to), we can estimate the minimum number of partitions Φ (number of centroids) required to handle highly imbalanced data distributions effectively, which is: $\Phi = \lceil \Omega \times N/\Gamma \rceil$. After estimating Φ, we apply K-means clustering on a small sample of the dataset to obtain Φ centroids $C = \{c_1, c_2, c_3, \ldots, c_\Phi\}$ as the initial reference points for vector assignment.

Instead of assigning each point to a fixed number of divisions, we propose a novel vector assignment method (Algorithm 1) that allocates points to divisions based on their geometric relationships with centroids, while simultaneously constraining the overload bounds for each subset. Each point is assigned to a division if its distance to the centroid is less than the average distance (lines 10-12) of previous assignments, scaled by an adaptive relaxation parameter $\epsilon > 1$ (line 9). This parameter adapts to different data distributions: It should remain small for uniformly distributed datasets to minimize redundant assignments, while being larger for structured datasets that are challenging to capture using distance-based clustering methods like K-means. A larger parameter allows points to be assigned to more divisions, even when distances to some centroids are greater, fostering overlap between divisions and enabling

the formation of long bridging edges among disconnected ANNS graphs. This improves graph connectivity, which is crucial for an efficient ANNS search with a high recall rate. Upon reaching the overload limit (line 13), the current average distance is reset (line 17) to ensure that each point is assigned to at least one division.

An example for Algorithm 1 is illustrated by Figure 1(a) and Figure 1(b). In Figure 1(a), assuming no overload happens and $\Omega = 3$, $\epsilon = 1.5$, the first assignment for **P1** is **C1** $\leftarrow$ **C1** $\cup$ {**P1**} (line 15). Due to $d_4 < 1.5 \times d_1$ and $d_3 > 1.5 \times \frac{\sum(d_1,d_4)}{2}$ (line 9-12), the final assignment is **P1** $\in$ **C1** $\cap$ **C4**. For **P2**, since **C4** $\leftarrow$ **C4** $\cup$ {**P2**}, **C3** $\leftarrow$ **C3** $\cup$ {**P2**}, $d'_1 < 1.5 \times \frac{\sum(d'_4,d'_3)}{2}$, and Ω is reached (line 7), the final assignment is **P2** $\in$ **C4** $\cap$ **C3** $\cap$ **C1**. In Figure 1(b), assuming **C1** is full and $\Omega = 3$, $\epsilon = 1.8$, since **C4** is the next nearest one (line 8), $d_3 < 1.8 \times \frac{\sum(d_1,d_4)}{2}$, and $d_2 > 1.8 \times \frac{\sum(d_1,d_4,d_3)}{3}$, the final assignment is **P** $\in$ **C4** $\cap$ **C3**. This assignment diverges from the original geometric relationships between **P** and the centroids **C1** to **C4**. It can be interpreted as relocating **P** from **C1** $\cap$ **C4** to **C4**$\cap$**C3**, resulting in a new point **P'**. However, with an appropriately chosen ϵ, this deviation can be constrained within an angle $\theta < 90°$ centered at **C4**, which has minimal impact on the neighborhood relationships. Our experiments, as well as the findings from the ANGN algorithm in [17], demonstrate that the graph quality for ANNS is preserved under these conditions.

Finally, since the vector assignment process is independent and requires limited memory per task, the proposed system parallelizes the quantization encoding alongside the vector assignment without any redundant computations, offering greater efficiency than the sequential approach used by DiskANN. The quantization is usually used to accelerate distance approximations while reducing storage for similarity search. As shown in Figure 1(c), each vector is encoded only once, and the resulting codes are merged in the subsequent stage, ensuring scalability and efficiency.

2.2 Distributed ANNS Graph Construction with Agglomerative Subgraph Merging

This section introduces a comprehensive and highly efficient framework designed specifically for the construction and seamless merging of subgraphs within a distributed computing environment. First, we employ a load-balancing scheduling method for subgraph construction, leveraging the near-linear relationship between ANNS graph construction time and dataset size. As shown in Figure 2(a), subsets produced by the previous stage (Section 2.1) are first sorted by size in descending order to prioritize larger tasks, which have a greater impact on load balancing. After sorting, graph-building tasks are iteratively assigned to the least-loaded machine or container, ensuring that each task contributes the least to the maximum load. By always choosing the least-loaded machine, the scheduling process greedily maintains a minimum overall load balance across all machines or containers in a cluster. Additionally, the maximum subset size is constrained by the previous stage (Section 2.1), mitigating overload issues such as out-of-memory errors and insufficient disk capacity. Otherwise, more advanced scheduling methods like BDSC [12] or LSSP[18] would be required. As a result, the system scales efficiently by incorporating low-resource workers to accelerate construction, rather than relying on a single high-resource worker.
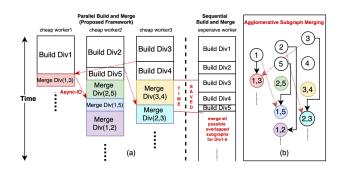

Figure 2: (a) Parallel build and merge framework versus Sequential one. (b) Agglomerative subgraph merging topology.

Then, the agglomerative subgraph merging strategy, illustrated in Figure 2(b), enables efficient merging by allowing immediate processing of completed subgraphs. Using distributed computing frameworks such as Apache Spark or MapReduce[5], subgraphs are asynchronously exchanged among different machines or containers to enable the next level of merging. The computationally intensive part involves neighbor selection for overlapping regions, while disjoint parts carry over to the next stage without additional computation. By employing a hierarchical or tree-based structure, the complexity decreases from $O(n)$ in traditional methods to $O(\log n)$. Unlike on-disk merging used in methods such as DiskANN, which must handle memory constraints when merging all subgraphs simultaneously, our approach prioritizes in-memory merging for subgraphs that fit within the main memory at each stage of execution. This strategy not only mitigates I/O bottlenecks but also enhances graph quality by allowing immediate access to vectors and neighbor candidate distances, leading to more precise pruning and selection. Furthermore, we optimize task scheduling by dynamically monitoring subgraph overlap counts, ensuring that merges with higher overlap receive higher priority.

3 Experiment

3.1 Experimental Setting

Datasets. We conduct experiments on five datasets, including image embeddings (**SIFT1M**, **SIFT1B**, and **ISD3B** - Our image search dataset), text embeddings (**GloVe**), and video embeddings (**VDD10B** - Our video de-duplication dataset). Their main characteristics are summarized in Table 1. LID [1] indicates local intrinsic dimensionality which implies the hardness of a dataset.

Compared approaches. We selected the following methods for comparison with the proposed approach (**SOGAIC**): HNSW [14] as implemented in Faiss [6], DiskANN [10], and SPTAG [4].

Evaluation metrics. For each dataset, performance is evaluated by measuring index construction time against the recall rate with a fixed number of CPU cores, while scalability is assessed by tracking construction time relative to the number of CPU cores, keeping the recall rate constant. To ensure fairness, we tested the proposed method using all graph structures employed by the compared methods and averaged the results to minimize variations. To ensure fairness, we tested the proposed method using all graph structures

Table 1: Statistics of real-world datasets.

Dataset	Dim	# Base	# Query	LID [1]
SIFT1M[2]	128	1,000,000	10,000	9.3
SIFT1B[2]	128	1,000,000,000	10,000	12.9
GloVe[16]	100	1,183,514	10,000	20.0
ISD3B	256	3,645,232,672	10,000	29.1
VDD10B	512	10,483,835,016	10,000	10.9

employed by the compared methods and averaged the results to minimize variations.

3.2 Experimental Result and Analysis

3.2.1 Construction Performance. As shown in Figure 3 (left column), **SOGAIC** achieves an average time reduction of 47.3% across all datasets larger than *SIFT1M* compared to the baselines. For datasets exceeding one billion points, such as *SIFT1B*, the HNSW implementation in Faiss failed to complete due to out-of-memory issues. On the *ISD3B* dataset, which has a high LID value similar to *GloVe*, DiskANN failed to complete due to severe data partition imbalance, while SPTAG required significantly more time to achieve higher recall, as it performs numerous iterations to introduce sufficient overlap. In contrast, **SOGAIC** not only resolved the overload issue but also reduced the overlap needed to maintain comparable accuracy, decreasing it from the preset maximum overlapping factor (Ω=4) to an average of 1.93 subsets per vector (total points to build for all subsets / base points). This 51.8% improvement is achieved through the use of a finely tuned adaptive parameter (ϵ=1.8), which dynamically adjusts the assignment process to balance efficiency and accuracy without unnecessary computational overhead.

3.2.2 System Scalability. As shown in Figure 3 (right column), the results demonstrate that the proposed system, **SOGAIC**, significantly reduces construction time as more computational resources are utilized. Unlike other baselines, **SOGAIC** maintains a near-linear relationship between performance and resource usage across all datasets, making it highly scalable with the addition of machines. For the ultra-large-scale, high-dimensional dataset *VDD10B*, SPTAG failed to complete within 100 hours when attempting to achieve a previously controlled recall rate (0.95) with 128 CPUs. Meanwhile, DiskANN suffered from consistently high construction times across all resource configurations due to poor scalability. In contrast, **SOGAIC** completed the process in approximately 80 hours, further reducing the time to under one day with 512 CPUs. This improvement is attributed to the load-balanced scheduling framework and the agglomerative merging strategy.

4 Conclusion

In this paper, we introduce **SOGAIC**, a scalable and overload-aware system specifically designed for constructing ultra-large-scale graph-based ANNS indexes. To address the performance and scalability limitations of existing systems, we propose a dynamic data partitioning algorithm that adaptively introduces overlaps among subsets while adhering to overload constraints, along with a distributed load-balancing subgraph construction framework with
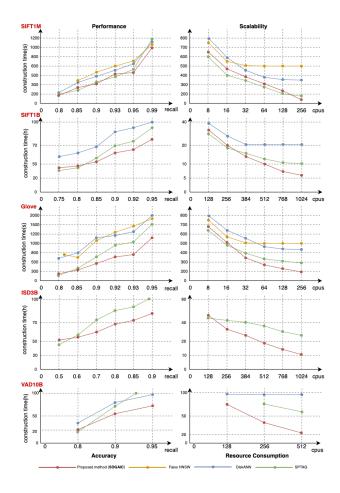

Figure 3: Comparisons of the proposed method (SOGAIC) with others (Faiss HNSW, DiskANN, SPTAG) on both performance and scalability for different datasets: *SIFT1M*, *SIFT1B*, *GloVe*, *ISD3B*, *VDD10B*.

an agglomerative merging strategy. Experimental results demonstrate that **SOGAIC** delivers a 47.3% average performance improvement compared to existing methods while preserving scalability. Furthermore, this approach has been successfully deployed in a real-world industrial search engine, managing over 10 billion daily updated vectors and serving hundreds of millions of users.

References

[1] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 29–38.
[2] Anon. 2010. Datasets for approximate nearest neighbor search. Retrieved October 05, 2020 from http://corpus-texmex.irisa.fr/.
[3] Jon Louis Bentley. 1980. Multidimensional divide-and-conquer. *Commun. ACM* 23, 4 (1980), 214–229.
[4] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search*. https://github.com/Microsoft/SPTAG
[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[7] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[8] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[9] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355* (2018).

[10] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[11] Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. 2024. Piperag: Fast retrieval-augmented generation via algorithm-system co-design. *arXiv preprint arXiv:2403.05676* (2024).

[12] Dounia Khaldi, Pierre Jouvelot, and Corinne Ancourt. 2015. Parallelizing with BDSC, a resource-constrained scheduling algorithm for shared and distributed memory systems. *Parallel computing* 41 (2015), 66–89.

[13] Saim Khan, Somesh Singh, Harsha Vardhan Simhadri, Jyothi Vedurada, et al. 2024. BANG: Billion-Scale Approximate Nearest Neighbor Search using a Single GPU. *arXiv preprint arXiv:2401.11324* (2024).

[14] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[15] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4236–4247.

[16] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2015. GloVe: Global Vectors for Word Representation. Retrieved April 15, 2020 from http://nlp.stanford.edu/projects/glove/.

[17] Shahin Pourbahrami and Leyli Mohammad Khanli. 2018. A Survey of Neighbourhood Construction Models for Categorizing Data Points. *arXiv preprint arXiv:1810.03083* (2018).

[18] Andrei Radulescu and Arjan JC Van Gemund. 2002. Low-cost task scheduling for distributed-memory machines. *IEEE transactions on parallel and distributed systems* 13, 6 (2002), 648–658.

[19] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 1378–1388.

[20] Yiping Sun, Yang Shi, and Jiaolong Du. 2024. A Real-Time Adaptive Multi-Stream GPU System for Online Approximate Nearest Neighborhood Search. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4906–4913.

[21] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-NN graph construction for visual descriptors. In *CVPR 2012*. 1106–1113.

[22] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.