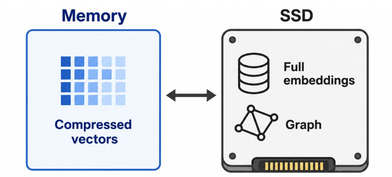
Passing the Baton: High Throughput Distributed Disk-Based Vector Search with BatANN

Nam Anh Dang, Ben Landrum, Ken Birman (Cornell University)

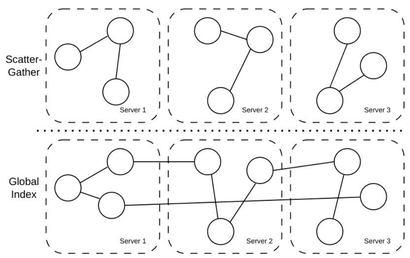
1. Introduction

1.1 Disk-based Graph Vector Search

- Stores **full vectors + adjacency list** on **SSD**. Compressed Vector representation (**PQ**) stored **in-memory**, used to guide search. Reranking with full embeddings after search.



1.2 Distributed Graph-based Vector Search:



Scatter Gather: Low communication cost but high computation cost

Global Index: Low computation cost but high communication cost

Global Index Methods:

- DistributedANN: centralized orchestration + repeated remote scoring requests.
- CoTra (RDMA): multiple distributed beams + remote access and periodic synchronization.

How can we distribute and query a global search graph at high throughput without excessive cross-server communication?

2.1 BatANN overview

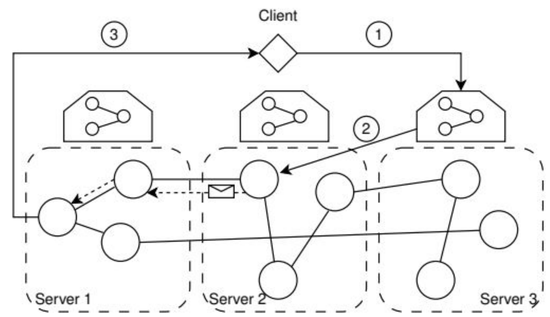


Figure 1: BatANN searches a partitioned global graph. A server first searches a replicated in-memory head index to find starting nodes. During disk-based traversal, the active server either expands local candidates or forwards the full query state to the server storing the next useful graph region.

2. Methods

2.2 Effect of In-Memory Head Index

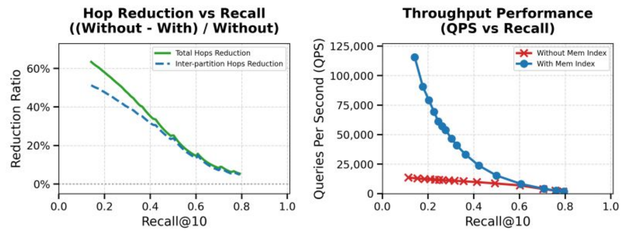


Figure 2: Effect of the in-memory head index on Text2Image 100M with 10 servers. At lower recall, the head index reduces total hops and inter-partition hops by more than 50%, producing a large throughput improvement.

2.3 Effect of Graph Partitioning

- **Graph partitioning improves locality:** random partitioning causes **4.71× more cross-partition hops** and raises P99 latency from **11.42 → 18.81 ms**.
- **With random partitioning for both systems, BatANN is still 1.48× faster than DistributedANN.**

2.4 Effect of I/O Pipeline Width (W)

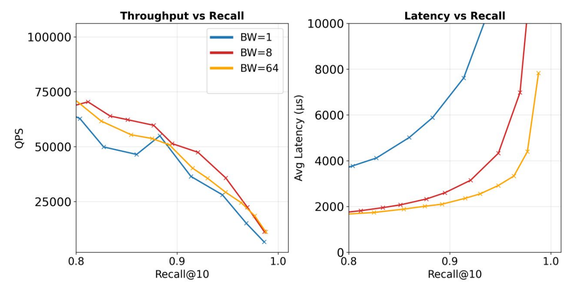


Figure 6: Throughput and latency comparison for $W = 1, 8, 64$ for BIGANN 1B on 10 servers. $W = 8$ has highest throughput, while $W = 64$ has lowest latency.

Experiment Setup

- Graph Construction: Used ParlayANN to construct R=64,L=128,alpha=1.2 Vamana graphs
- Graph Partitioning: Used the Graph Partitioning algorithm by Gottesburen et al.
- Baselines:
  - ScatterGather: partition dataset to servers. Each server builds its local index. Searches all servers to get top-k.
  - DistributedANN: distributed disk-based global graph approach. Uses centralized server + scoring servers

3. Results

3.1 Throughput & Scalability

- BatANN achieves **1.44–2.09×** DistributedANN and 3.50–5.59 × ScatterGather throughput on 1B datasets with 10 servers, and scales better as servers are added.

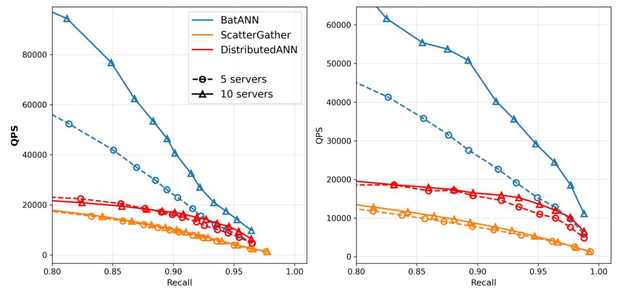


Figure 10: QPS recall curve of BIGANN, MSSPACEV on 5 and 10 servers for 1B. BatANN outperforms ScatterGather on all setups with $W = 64$ for BatANN and DistributedANN and $W = 8$ for ScatterGather

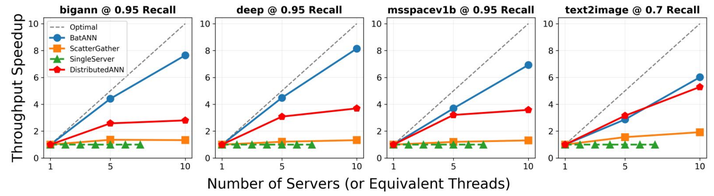


Figure 9: Throughput speedup on 100M-scale datasets up to 10 servers. BatANN exploits additional cores and SSD bandwidth more effectively than ScatterGather and DistributedANN.

3.2 Latency

- At 10K QPS, BatANN achieves **11.42 ms P99** vs. **19.63 ms** for DistributedANN (P50/P95/P99: 4.62/9.06/11.42 vs. 4.53/7.82/19.63 ms).
- - Near peak throughput at 16K QPS, BatANN maintains **12.59 ms P99**, stable at **12.56–12.69 ms** across four runs.

Conclusion

- BatANN reduces cross-server coordination through state passing while preserving global-graph efficiency.
- It achieves higher throughput than ScatterGather and DistributedANN, while maintaining low, stable latency on commodity TCP.