Concentric Vector Chord:

A Kernelized Vector Distributed Hash Table

Technical Summary and Prototype Findings

Shyamal Suhana Chandra
Sapana Micro Software
2905 Woodgate Drive
Pittsburg, KS 66762
sapanamicrosoftware@gmail.com

June 2026

Abstract

We present *Concentric Vector Chord* (CVC), a modular C99 prototype that combines order-preserving embedding-to-ring mapping, locality-sensitive hashing (LSH), optional kernelized LSH (KLSH), and a Chord-style overlay to support approximate k-nearest-neighbor (kNN) retrieval over high-dimensional vectors in both Euclidean and kernel-defined similarity spaces. Unlike classical distributed hash tables (DHTs), which scatter keys uniformly and provide exact lookup only, CVC preserves semantic locality: vectors that are similar under the chosen metric are mapped to nearby Chord keys and LSH buckets, enabling decentralized similarity search without a centralized index. We describe the algorithmic design, report results from an in-process simulation harness and Vulkan-based visualization tool, discuss engineering lessons from the reference implementation, and outline directions for a production-grade peer-to-peer deployment.

1 Introduction

Embedding-based retrieval underpins modern information systems: dense vector representations power semantic search, retrieval-augmented generation (RAG), deduplication, and content discovery. Centralized vector databases (e.g., FAISS-backed services, Milvus, Pinecone) scale well but introduce operator trust, single points of failure, and infrastructure cost.

Distributed hash tables such as Chord [9] and Kademlia [7] excel at exact key–value lookup in $\mathcal{O}(\log N)$ hops but deliberately *destroy* key locality: similar keys are unrelated to similar *values*. Prior work on metric DHTs (e.g., MCAN [3], GHT* [1]) and distributed LSH [4] shows that similarity search in peer-to-peer settings requires specialized placement and routing, not uniform hashing alone.

Concentric Vector Chord addresses this gap with a composable architecture:

1. **Concentric shell mapping** projects embeddings onto polar coordinates and maps (shell, angle) to an order-preserving Chord key;
2. **LSH / KLSH** assigns similarity-preserving bucket signatures for Euclidean, cosine, and RBF-kernel metrics;
3. **Chord routing** locates responsible nodes and records hop paths;
4. **Hybrid kNN search** merges LSH bucket probes, centroid filtering, and local reranking;

5. **Vulkan visualization** renders shells, ring nodes, stored vectors, and query routes for design validation.

This document summarizes the design and findings of the reference implementation (version 0.1.0).

2 Background

2.1 Distributed Hash Tables

A DHT maps keys to a logical ring of 2^m identifiers. Each node maintains a finger table of distant successors, enabling greedy routing in $\mathcal{O}(\log N)$ hops [9]. Lookup returns the *successor* of a key—the first node whose identifier is $\geq$ the key on the ring.

2.2 Locality-Sensitive Hashing

LSH families map nearby points to identical hash buckets with probability higher than distant points [5]. Random hyperplane hashing [2] supports cosine similarity; p-stable LSH supports L_2 distance. Distributed deployments map LSH buckets to peers with locality-preserving placement to reduce cross-node probe traffic [4].

2.3 Kernelized LSH

When embeddings $\phi(x)$ are implicit or infinite-dimensional, standard LSH does not apply directly. Kernelized LSH (KLSH) [6] constructs hash functions using only kernel evaluations $k(x, y)$, via anchor-based approximations related to random projections in reproducing kernel Hilbert space (RKHS). This extends approximate similarity search to non-Euclidean and non-explicit-embedding settings.

2.4 Motivation for Concentric Mapping

Uniform hashing of vector-derived keys would not preserve neighborhood structure in embedding space. CVC instead treats magnitude (shell) and direction (angle) as a polar decomposition after a random 2D projection, then packs normalized (r, θ) into a 32-bit mixed integer before a final mixing hash onto the Chord ring. Nearby vectors in the projected plane tend to receive nearby ring keys, analogous to learned order-preserving hashes but with a lightweight geometric interpretation.

3 Algorithm

3.1 System Model

The prototype simulates N DHT nodes and stores up to M vectors of dimension $d \leq 256$. Each vector v receives:

- a **primary key** $K_{\text{cvc}}(v)$ from concentric mapping;
- an **LSH signature** $H_{\text{lsh}}(v)$ (16 hyperplane bits);
- an optional **KLSH signature** $H_{\text{klsh}}(v)$ when kernel mode is enabled;
- an **owner node** on the Chord ring, with successor overflow if the primary bucket is full.

Each node maintains a centroid of its stored vectors for coarse filtering.

3.2 Concentric Key Mapping

Given random orthonormal-ish basis vectors $b_u, b_v \in \mathbb{R}^d$, project:

$$x = v \cdot b_u, \qquad y = v \cdot b_v, \qquad r = \sqrt{x^2 + y^2}, \qquad \theta = 2(y, x) \bmod 2\pi. \tag{1}$$

Normalize $\hat{r} = \min(r/r_{\max}, 1)$ and $\hat{\theta} = \theta/2\pi$, then form a mixed word and apply a 64-bit mixing function $\mathrm{mix}(\cdot)$ to obtain the Chord key:

$$K_{\mathrm{cvc}}(v) = \mathrm{mix}\big(\lfloor \hat{r} \cdot 65535 \rfloor \ll 16 \parallel \lfloor \hat{\theta} \cdot 65535 \rfloor\big) \bmod 2^{16}. \tag{2}$$

3.3 LSH and KLSH Signatures

Standard LSH: For each of $k = 16$ random unit hyperplanes h_j, set bit j if $v \cdot h_j \geq 0$.

KLSH: Select up to 32 anchor vectors from the database. For hash bit b, compute

$$s_b(v) = \sum_{i=1}^{p} \sigma_{b,i}\, w_i\, \mathrm{sim}(v, a_i), \tag{3}$$

where sim is the configured metric (e.g., RBF kernel $k(v, a) = \exp(-\gamma\|v - a\|^2)$), $w_i = 1/\sqrt{p}$, and $\sigma_{b,i} \in \{-1, +1\}$ is a deterministic sign pattern. Bit b is set if $s_b(v) \geq 0$.

3.4 Insertion and Overflow Placement

Algorithm 1 summarizes vector insertion.

Algorithm 1 Vector insertion with Chord successor overflow

Require: DHT state D, embedding v

```
k ← K_cvc(v)
n ← ChordSuccessor(k)
while BucketFull(n) and not wrapped ring do
    n ← ChordSuccessor(n)
end while
Store v on node n; update centroid
return owner node id
```

Overflow placement prevents bucket saturation when concentric mapping clusters many vectors onto the same successor—an issue observed when ≥ 300 vectors were assigned to 24 nodes without overflow.

3.5 kNN Query Processing

Given query q and result count k:

1. Compute $K_{\mathrm{cvc}}(q)$ and route on the Chord ring from a bootstrap node, recording hop ids;
2. Compute LSH/KLSH bucket $H(q)$;
3. Collect candidate vectors whose bucket Hamming distance to $H(q)$ is ≤ 2, plus all vectors on the primary owner node;
4. Expand nodes whose centroid similarity to q exceeds 0.5;
5. Score all candidates with the configured metric, sort descending by similarity, return top-k.

This is an *approximate* search: recall depends on LSH parameters, shell mapping quality, and the centroid threshold.

3.6 Visualization Pipeline

A Vulkan/GLFW renderer projects the DHT state each frame: concentric shell guides, the Chord ring, node ticks, 2D-projected vector crosses, and an orange polyline of Chord routing hops for the active query. The tool validated concentric clustering, overflow behavior, and routing paths during development.

4 Implementation

The reference implementation is written in C99 with a modular layout: `vec`, `metric`, `lsh`, `klsh`, `concentric`, `chord`, `dht`, `search`, `sim`, and `viz`. Large structures are heap-allocated (`cv_dht_create`, `cv_sim_create`) to avoid stack exhaustion on embedded arrays up to 2048 vectors $\times$ 512 nodes.

Default configuration limits include a 2^{16} Chord ring, 16 shells, 16 LSH bits, 128 vectors per node (with overflow), and support for L_2, cosine, and RBF metrics.

5 Benchmarks and Observations

All measurements below were taken with the in-process `-demo` CLI on Apple Silicon (macOS), seed = 42, unless noted. These are **simulation benchmarks**, not wide-area network trials; routing hop counts reflect the local Chord finger-table model without RPC latency.

5.1 Experimental Setup

Table 1: Benchmark configurations

Config	Nodes	Vectors	Dim	Metric
A (baseline)	16	256	32	cosine
B (kernel)	16	256	32	RBF + KLSH
C (scale)	24	512	32	cosine

5.2 Query Quality

Table 2 reports top-8 retrieval for a self-query (query vector equals stored id 0 or 42). Self-similarity is 1.0 in all configs. Neighbor ranks illustrate metric-dependent neighborhood structure.

Table 2: Top-k similarity scores (self-query, $k = 8$)

Cfg	Rank	Vector id	Similarity
A	1–8	0, 67, 63, 41, 50, 20, 23, 48	1.00, 0.35, 0.34, 0.26, 0.22, 0.20, 0.20, 0.18
B	1–8	0, 2, 26, 41, 20, 23, 1, 13	1.00, 0.96, 0.96, 0.95, 0.95, 0.95, 0.95, 0.95
C	1–8	42, 30, 29, 51, 63, 28, 20, 35	1.00, 0.35, 0.30, 0.27, 0.21, 0.21, 0.20, 0.20

Finding: KLSH with RBF kernel (Config B) yields tighter neighbor clusters in kernel space than cosine LSH (Config A), as expected when the hash and scoring metrics align. Cosine LSH produces sparser top-k similarity decay, reflecting the broader geometry of random unit vectors.

5.3 Routing and Placement

- **Chord hops:** For the tested queries, hop count was often 0 because the bootstrap node was already the successor of the query key in small simulations ($N \leq 24$). Larger N and remote bootstrap selection would increase hop counts toward $\mathcal{O}(\log N)$.
- **Overflow:** Without successor overflow, initialization failed at $\sim$300 vectors / 24 nodes (`CV_ERR_RANGE`) due to bucket saturation from concentric clustering. Overflow restored stable insertion up to 512 vectors.
- **Init latency:** Full simulation init + query completes in < 100 ms on tested hardware (256–512 vectors).

5.4 Visualization

The Vulkan viewer successfully renders shell guides, ring topology, projected vectors, and query routes on macOS/MoltenVK after pipeline-layout initialization fixes. It serves as an effective debugging aid for placement skew and routing paths, though it is not a performance benchmark.

6 Future Work

1. **Network transport:** Replace the in-process simulator with UDP/TCP RPCs, stable node ids, and failure detection.
2. **Learned or adaptive mapping:** Explore order-preserving learned hashes (cf. LEAD-style CDF mapping) alongside geometric shells.
3. **Stronger ANN per node:** Integrate HNSW or IVF indexes locally while retaining DHT-level bucket routing [8].
4. **Replication and repair:** Store r replicas on the r nearest ring successors of each vector key for churn tolerance.
5. **Rigorous evaluation:** Measure recall@k, probe count, and bytes transferred versus centralized FAISS and distributed LSH baselines on standard embedding datasets (e.g., GloVe, SIFT, deep descriptors).
6. **Security:** Address Sybil attacks, false centroid announcements, and kernel/embedding poisoning in open peer-to-peer settings.
7. **Dynamic metrics:** Pluggable Mercer kernels and pivot-based metric routing for non-kernel distances (edit distance, Jaccard).

7 Conclusion

Concentric Vector Chord demonstrates that classical DHT infrastructure can be extended to approximate vector similarity search by combining three ideas: order-preserving concentric ring keys, LSH/KLSH bucket signatures, and Chord successor routing with overflow-aware placement. The C99 prototype validates the architecture through a modular API, CLI simulation, and Vulkan visualization. Kernelized hashing measurably changes neighbor structure compared to cosine LSH, and concentric clustering motivates overflow policies essential at modest scale.

The system is a research prototype, not a production P2P database. Its principal contribution is a clear, composable blueprint for *kernelized vector DHTs* spanning Euclidean and implicit kernel spaces, with an open path toward decentralized embedding retrieval.

References

[1] Michal Batko, David Novák, and Pavel Zezula. GHT∗: A peer-to-peer system for metric data. In *Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2006.

[2] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002.

[3] Paolo Ciaccia, Marco Patella, and Pavel Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, 1997.

[4] Parisa Haghani, Sebastian Michel, Philippe Cudré-Mauroux, and Karl Aberer. Distributed similarity search in high dimensions using locality sensitive hashing. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT)*, pages 1314–1319, 2009.

[5] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998.

[6] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[7] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. *IPTPS*, 2002.

[8] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2006.

[9] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.