ABSTRACT

As high-dimensional queries in online systems gain importance with the widespread adoption of Machine Learning and Neural Networks, high-dimensional databases have emerged as a significant research topic. Tasked with processing vast amounts of data (both vector and scalar) in various contexts and conducting searches in high-dimensional metrics and scalar columns, contemporary DBMS face two primary challenges: 1. Treating vector indices as a black box without unlocking their optimization potential, and 2. Lacking a feasible query planning mechanism for complex vector queries. This situation conflicts with the online nature of these applications.

Building on the novel work of VBase, which deploys an iterative abstraction on vector indices to overcome the black-box limitation, this paper presents a unified cost abstraction for high-dimensional indices. We demonstrate that this model effectively captures various types of queries and seamlessly integrates into relational databases. As a result, high-dimensional databases can execute queries using the optimal plan, yielding superior performance (multiple times) compared to basic strategies.

Key Words: Cost Model; High-dimensional Databases

Contents

Chapter 1 Introduction

Driven by the feature extraction[1] capabilities of deep learning models and machine learning techniques, recent years have seen significant progress in large-scale applications of high-dimensional data queries. These include recommendation systems[2] (identifying users with shared interests), image processing (classification[3-5] and deduplication[6]), search[7], and more. High-dimensional queries[8] involve finding nearest neighbors in high-dimensional spaces. Given the prohibitive cost of exact search, researchers have developed Approximate High-dimensional Nearest Neighbor Search(ANNS)[9] techniques that balance search accuracy with efficiency. Traditional applications have focused on two basic query types for ANNS: top-k and range search, which have been comprehensively investigated[10-12] in recent decades. However, emerging applications such as e-commerce[13-14] demand more diverse query types. For instance, consider the example in AnalyticDB-V[15], an online shopper seeking a dress similar to one worn by a celebrity (vector search over style embedding) for under 100$ (scalar search on the price column). Efficiently executing these *mixed queries* is a pressing research challenge.

Previous attempts to address this problem involved integrating high-dimensional indices with traditional relational database management systems (DBMS). However, these approaches treated the vector search engine as a black box, executing vector and scalar searches separately[16] before merging the results. VBase[17] successfully bridged separate search paths by leveraging the relaxed monotonicity in high-dimensional indices, permitting iterative result retrieval akin to scalar indices within the volcano model[18]. Unlocking vector indices leads to both opportunities and challenges for optimization. Revisiting the online shopping example, two potential execution paths exist for the query in the iterative model: 1) scanning high-dimensional indices first and then applying a price filter on the results; or 2) scanning the B-Tree index on price and filtering out the nearest goods by range filter. Accurate cost prediction is crucial for guiding the executor toward the optimal path.

Query planning[19] is a crucial optimization subsystem in traditional DBMS, responsible for estimating[20] the costs of various execution plans beforehand and guiding

the DBMS to select the optimal plan. For instance, inner joins across multiple tables can be optimized by altering the order to reduce intermediate results. In the case of ”mixed” queries (combining vector and scalar filters) like the online shopping example, optimization can be achieved by applying the stricter filter first. However, accurate estimation of vector search costs is pivotal for this purpose. Prior to VBase, vector search was an independent component of DBMS, and query planning was not required, resulting in only a few[21-22] early studies on cost modeling for high-dimensional indices. Therefore, this paper represents a pioneering attempt to model the costs of contemporary high-dimensional indices.

The challenge of accurately estimating costs for high-dimensional indices arises from the selectivity estimation of queries and the complex mechanisms employed by various indices to reduce amount of computation. VBase[17] introduced a simple cost prediction model for vector search and range filtering, enabling DBMS search engines to choose better plans. However, the cost model offered in VBase only applies to specific indices.

As a complementary extension to VBase, this paper underscores the discovery of shared search patterns common to high-dimensional indices: optimization techniques for high-dimensional searching can be categorized into **quantization** and **navigation**. Building on this insight, we generalize the cost model in VBase[17] by proposing a universal cost abstraction for high-dimensional indices. We evaluate the unified cost abstraction by implementing an actual cost estimation subsystem in PostgreSQL. Our system demonstrates satisfactory accuracy in mixed query cost prediction and a significant advantage (depending on the query) over default planning.

Chapter 2 Background

2.1 Emerging Online Vector Queries

The accelerated development of machine learning and neural networks has led to the widespread use of high-dimensional embeddings in various fields such as recommendation systems[2,23], e-commerce[13-14], and search[7]. These **online** applications necessitate fast querying of vector data. The exact search cost proves infeasible for these online scenarios, leading to the emergence of many ANNS[24-27] techniques. These indices balance speed and accuracy by relaxing recall requirements in traditional query types like kNN[28-29] and Range Query. However, new applications[16] have proposed the need for new query types, such as high-dimensional table join, mixed query (with both scalar and vector filters), and multi-column topK query. These demands have spurred researchers to develop more powerful query engines, with one promising approach being the integration of vector indices into relational DBMSs.

2.2 Combining Relational Database and Vector Indices

Recent developments have seen the establishment of DBMSs[15-16,30] and search engines[31] supporting hybrid and complex vector queries. However, these systems either treat vector indices as a black box, with mixed-filter queries processed separately by vector and scalar indices[16], or perform batch scanning with default planning[30]. VBase[17] identified the relaxed monotonicity in vector indices, opening the "black box" and integrating it into the volcano model[18] (iterative model). This innovative approach make query planning for high-dimensional operations possible and feasible.

```
SELECT count(*) FROM goods
WHERE dist(embedding, query_embedding) < 10
AND price < 50;
```

Listing 2.1 <–> refers to L2 distance, q is the embedding of the query vector, rand_id is a randomly generated numeric column.

For instance, determining the best execution path for the mixed query before actual execution can greatly enhance performance.

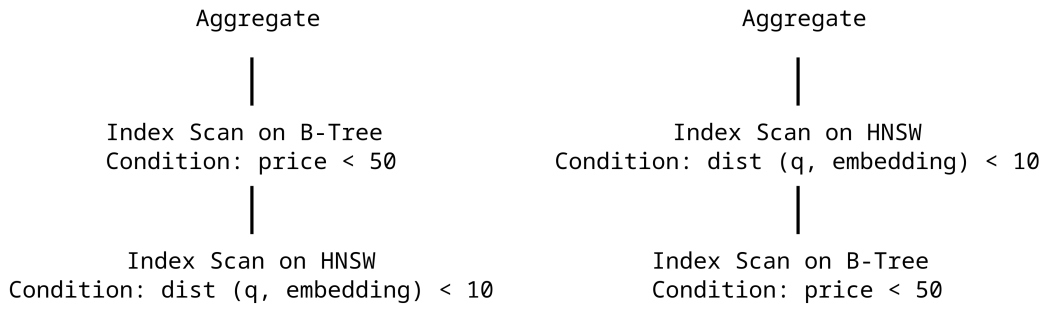

Figure 2.1 Two possible planning of query. Different planning may result in significantly differed execution time, for instance there is only a few matches for goods cheaper than 50$, scanning B-Tree first would be considerably faster.

2.3 Query Optimization in Relational DBMS

Query optimization is a crucial component of relational DBMSs, comprising three parts: cardinality estimation (CE), cost model (CM), and plan enumeration (PE)[32]. CM utilizes estimation data from CE and applies it to plans generated by PE. By estimating costs for different plans, the DBMS can determine the optimal execution path for a query. The cost model itself is a well-researched area in relational databases, encompassing logical and physical aspects[33], considering data processing properties (independent of deployment layout) and hardware specification impacts respectively. With the widespread adoption of NVME solid-state drives today, the additional cost of random disk page fetches over sequential page fetches has been eliminated[34], and the bottleneck for high-dimensional data distance search now lies in CPU operations. Consequently, we can focus on logical costs as differences in physical costs are diminished.

In relational DBMS, numerous optimizations[35] rely on the strict commutative property of operators, such as Inner Join. This property allows changes in execution order, enabling the selection of the optimal order. For example, TABLE A INNER JOIN TABLE B INNER JOIN TABLE C can be executed as TABLE A INNER JOIN (TABLE B INNER JOIN TABLE C). If table C is significantly smaller, the latter execution order can yield substantial performance gains by reducing the number of tuples scanned. DBMS typically predicts the optimal execution order using optimization subsystems before execution. As the mixed selection query defined in VBase[17] adheres to the relaxed commutative property, it can be optimized through cost estimation and plan enumeration. This provides opportunities for planning

optimization, underscoring the need for a precise cost model for ANNS.

$$\theta_{F1}(\theta_{F2}(A)) = \theta_{F2}(\theta_{F1}(A))$$

$F1$ and $F2$ here can be any filter, including range-based filter

2.4 Shared Concepts in Vector Search

VBase[17] identifies common search behaviors in ANNS. The shared search pattern originates from shared concepts in optimizing vector search. Current efforts to accelerate vector queries while preserving high accuracy can be broadly categorized into two tracks: Navigation and Quantization. This abstraction combines the ideas of dimensionality reduction and subspace clustering in high-dimensional cluster techniques[36], but it applies most ANNS approaches, as summerized in Chapter6. This section identifies and elucidates the two categories of concepts in vector search optimization.

（1）Navigation

Navigation focuses on minimizing the number of visited vectors in metric space to gather required results. The implementation of this concept varies across query types and ANNS, with nearly all employing it to some extent. Examples include clustering[37], NSG-based indices, hierarchical LSH[38], and tree-like indices such as R-Tree[27,39], RD-Tree[40], and KD-Tree[24]. For instance, kNN queries on HNSW[25] indices reduce the number of visited vectors by iterating through several layers of Navigable Small-World graphs(NSGs) to rapidly navigate near the target points in proximity to the query vector. When queries impose additional constraints, such as scalar filters, an effective navigation strategy should reduce the number of visited vectors before identifying and returning those that meet all criteria. Such as Milvus' partitioning on scalar columns[16] or altering the search order[17].

（2）Quantization

Quantization[41] refers to the process of reducing the dimensionality of the search space and, consequently, the cost of distance calculation. Widely adopted implementations include reducing the length of float point representation, Random Projection[42], Principal Component Analysis[43-44], and Product Quantization[45]. These methods either trade accuracy for speed, as in the case of reducing representation bits, or attempt

to extract and summarize distribution features in a compact fashion.

Upon discovering and identifying the shared concepts in ANNS, we propose a corresponding two-level cost abstraction to summarize ANNS built on these ideas.

Chapter 3 Two-Layer Cost Abstraction Design

In this section, we introduce a two-level cost abstraction consisting of navigation cost and traversal cost. Drawing on shared concepts and behaviors in vector search, this cost abstraction applies to most contemporary ANNs, including KDTree[24], HNSW[25], NSW[46], IVFPQ[47] and etc. Within this abstraction framework, we present two example cost models for two popular indices (IVFFlat and HNSW). These models are simple yet capable of providing accurate cost estimations at runtime.

3.1 Shared Search Pattern in ANNs

As all indices utilize the navigation concept defined in Section 2.4 to some extent to reduce the number of unrelated vectors fetched, there is a trend[17] in the distance of iterated vectors that includes two phases. In Figure 3.1, during phase 1, traversal approaches the target vector approximately, while in phase 2, traversal gradually moves away from the target vector in an approximately monotonic order.

To identify the pivot (or turning point) in search patterns, we can use a simplified definition from VBase[17]. M_q^s defines the median distance in the $2w+1$-sized traversal window (we utilize the median to account for varying "speeds" in distance changes between the two phases).

$$M_q^s = \text{Median}(\text{Dist}(q, v_i) | i \in [s - w, s + w]) \tag{3.1}$$

The pivot in traversal distance can be defined as follows:

$$\exists N, M_q^N \leq M_q^n, \forall n \tag{3.2}$$

With the pivot defined in Equation 3.2, we can propose a two-layer cost model reflecting the search phases divided by it. This model accounts for the distinct characteristics of each phase in the search pattern.

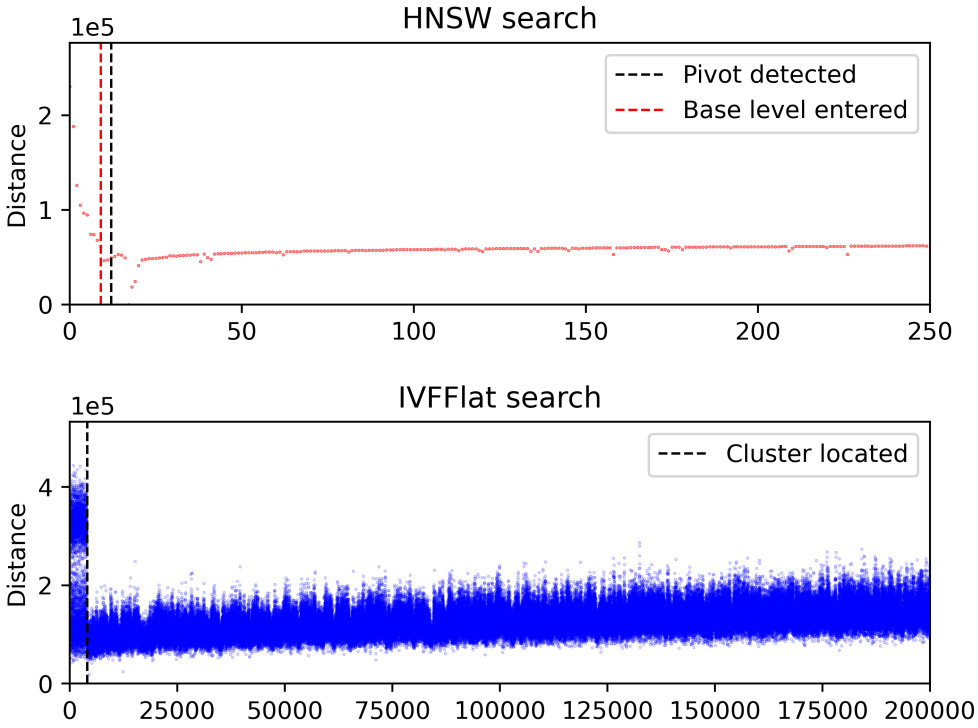

Figure 3.1 Trend in HNSW and IVFFlat search. The pivot in HNSW is detected with Equation 3.2. In IVFFlat, the first phase involves distance calculations between cluster centroids and the query vector. The pivot occurs when the nearest centroid is found, and the iterator begins searching within the corresponding cluster.

3.2 Cardinality Estimation of Range Filter

To model traversal cost in the second phase, we need to predict the number of tuples the iterator scans through. Cardinality (or selectivity) of the filter refers to the ratio of unique values that pass the filter condition to the total number of tuples in the table. It can help to estimate the number of iterator search steps and ultimately the cost in the second phase. As a kNN query typically returns k results in most cases (with cardinality being k/N, where N is the table size), we use range queries coupled with an additional scalar filter to demonstrate this aspect.

```
SELECT *
FROM sift1m
WHERE embedding <-> q < r
AND rand_id < p;
```

Listing 3.1 <-> refers to L2 distance, q is the embedding of the query vector, rand_id is a randomly generated numeric column.

There is a lot of work[48-51] focus on estimating the cardinality of the range filter. This cost abstraction is open to any of them, as it accepts cardinality as one of its inputs.

3.3 Two Layer Cost model

With the pivot defined in Section 3.1, we can establish a unified two-level abstraction corresponding to the two phases in vector traversal. Interestingly, this two-level abstraction is also present[20] in traditional scalar indices like B-Tree[52], which simplifies the implementation in Section 4.2. We name the costs before and after the pivot as *Start Up Cost* and *Iteration Cost*, respectively. C_{start} refers to the interval between the open() and next() operations in the Volcano model[18], while C_{iter} denotes the interval between next() calls, or the minimum time taken to prepare the next tuple.

$$C = C_{\text{start}} + C_{\text{iter}} \tag{3.3}$$

Before proceeding, we should also define the cost of the operator <–>. This operator calculates the L2 distance between two vectors. In modern processors, with the optimization of vector operations using SIMD, the cost can be defined as:

$$C_{\text{dist}} = c \times \text{dim} \tag{3.4}$$

c is a coefficient dependent on the environment (including ISA implementation and pre-defined cost in DBMS). This equation effectively captures the cost of calculating the L2 distance between two vectors within the given context.

3.3.1 Start Up cost

This cost is analogous to the cost with the same name in relational databases. For example, in the B-Tree index, this cost represents the process of navigating down the tree. In high-dimensional indices, it captures the cost of the initial search phase, i.e., navigation. The cost is proportional to the number of nodes (N_{start}) scanned.

$$C_{\text{start}} = N_{\text{start}} \times C_{\text{dist}} \tag{3.5}$$

For instance, consider HNSW as an example; the start-up cost describes the steps for iterating down higher levels, from the entry point to the query's nearest neigh-

bor. Given that the expected number of search steps is $\Theta(\log N)$[25], we can define the HNSW-specific Start-Up Cost:

$$C_{\text{start,hnsw}} = R_{\text{start}} \times \log N \tag{3.6}$$

R_{start} is a coefficient related to hyper-parameters (efConstruction, M_L, M) used when creating index, it can be a priori calculated from hyper-parameters or obtained through empirical statistics in DBMS's analyze routine. N is the number of nodes in the search space.

For IVFFlat and IVFPQ, the Start-Up Cost reflects the process of calculating the distance between query and cluster centroids and then sorting out the nearest clusters using Inverted Index. $N_{cluster}$ is the number of clusters when training index.

$$C_{\text{start,IVF}} = N_{\text{cluster}} \times C_{\text{dist}} \tag{3.7}$$

3.3.2 Iteration Cost

In relational databases, this cost reflects the cost of applying the filter to tuples in the area located during the initial phase. Assuming accurate navigation, the cost scales linearly with cardinality multiplied by the table size.

From 3.1, we observe that HNSW aligns well with this expectation, as it navigates down to the nearest point first. Here, we can safely simplify the estimation by assuming the recall of HNSW is equal to 1[25].

$$C_{\text{iter,hnsw}} = R_{\text{scan}} \times S(q) \times N \times C_{\text{dist}} \tag{3.8}$$

$S(q)$ is the selectivity of query q. R_{scan} is a coefficient reflecting the average number of distance function calls per scan. It is heavily influenced by hyper-parameters M, efConstruction, and the distribution of vectors. This coefficient can also be obtained empirically.

IVF indices do not exhibit a clear trend in 3.1. However, their scanning behavior can be easily modeled. After the N_{query} closest clusters are located, all vectors within these clusters are scanned and filtered. Thus, the iteration cost can be modeled as:

$$\begin{aligned} C_{\text{iter,ivf}} &= N_{\text{query}} \times N_{\text{vec,cluster}} \times C_{\text{dist}} \\ &= N_{\text{query}}/N_{\text{cluster}} \times N \times C_{\text{dist}} \end{aligned} \tag{3.9}$$

$N_{\text{vec,cluster}}$ is the average number of vectors per cluster. N_{query} is the number of clusters scanned, which is a runtime parameter in the FAISS implementation[47].

By defining the start-up cost and iteration cost as described above, we can effectively utilize this abstraction to model the complete ANNS scan process before execution, and finally enable more informed and optimized decision-making in DBMS' planning stage.

Chapter 4 Implementation

In this section, we implement an actual cost model based on the proposed abstraction in PostgreSQL. The cost model is utilized in the optimization (query planning) subsystem, where the SQL has already been parsed into an execution tree but the execution plans have not yet been generated. DBMS relies on the cost model to select better plans to feed into the executor. An effective cost model can help reduce execution costs in this process. We evaluate our cost model by collecting actual execution times in Section 5.3.

4.1 Cost Model Implementation

4.1.1 Cardinality Estimation

Though various studies[48-51] have focused on improving the cardinality estimation of high-dimensional queries, our implementation employs uniform sampling as the cardinality estimator due to its simplicity, suitability for cost model demonstration, and reliable prediction performance as indicated in Section 5.2. Our abstraction is flexible and can readily incorporate cardinality estimation results from these studies as input, which allows for potential enhancements in cost model accuracy and adaptability to various high-dimensional query scenarios.

The selectivity is evaluated using Equation 4.1, where S_{exec} represents the actual selectivity of the range filter, and S_{esti} denotes the selectivity determined by applying the range filter to a sample set.

$$S_{\text{exec}} \approx S_{\text{esti}} \tag{4.1}$$

4.1.2 Parameter Collection

R_{start} is empirically determined in the HNSW start-up cost by collecting the number of distance calculations during the search. This process can be integrated and automated with *Analyze* in the DBMS.

As illustrated in Figure 4.1, the ratio of distance call times to iteration steps re-

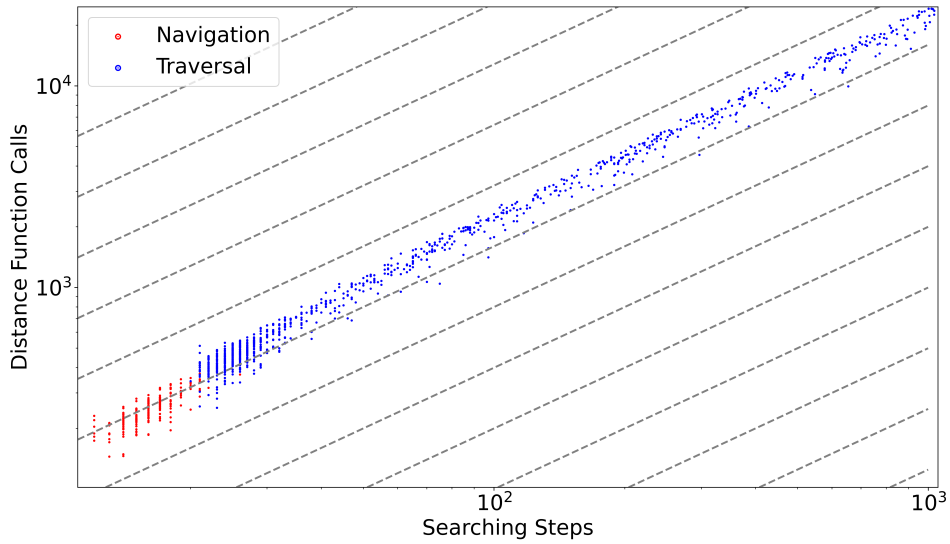

Figure 4.1 The scatter graph of distance function call times to iteration steps. Dashed lines denotes different ratio.

mains stable during the search. The distribution is not perfectly parallel to the dashed lines because, when the iterator scans further from the query, the ratio of visited vectors in the current node's hypersphere neighborhood decreases, resulting in more distance calculations. However, this effect has limited impact and can be safely ignored. For other indices, such as IVFFlat and IVFPQ, the scanning pattern is fixed, as clusters are always wholly scanned. In LSH, the calculation time of the hashing process is more predictable due to the use of fixed mapping algorithms.

Another cost parameter, C_{dist}, is proportional to the pre-defined cost constant in PostgreSQL and is addressed in Section 4.2.3.

4.2 Integration in PostgreSQL

4.2.1 Index Cost Estimation

PostgreSQL provides a set of APIs for index registration and processing, including query optimization and plan enumeration. The implementation of the cost model can focus on the amcostestimate API[20].

```
amcostestimate (PlannerInfo *root,
                IndexPath *path,
                double loop_count,
```

```
    Cost *indexStartupCost,
    Cost *indexTotalCost,
    Selectivity *indexSelectivity,
    double *indexCorrelation,
    double *indexPages);
```

Listing 4.1 PostgreSQL's index cost estimation API

The three parameters at the beginning provide the necessary context information. Since index scanning is a node in the execution tree, this information includes the loop count of its upper level, the considered index access path (with cost and selectivity awaiting calculation), and planning tree information. The other parameters are outputs of the cost model, providing the necessary runtime stats to PostgreSQL's backend. indexStartupCost represents the interval between open and next in the volcano model, or the time before the first result is ready. This cost corresponds to the navigation phase in the index scan. indexTotalCost is the sum of indexStartupCost and index iteration cost, with the latter corresponding to the traversal phase in the Abstraction. indexSelectivity is the estimated cardinality of the filter applied to the index. For range-based queries, it is the ratio of vectors within the hypersphere with a radius of r; for kNN queries, it is the ratio of k to the number of vectors. indexCorrelation can be set to zero, as the cache effect is weak for high-dimensional tuples and NVMe[34] devices. indexPages can be inferred from the filter selectivity. Our cost model implementation follows Equation 4.2.

$$\begin{aligned} \text{indexStartupCost} &= C_{\text{start}} \\ \text{indexTotalCost} &= C_{\text{start}} + C_{\text{iter}} \\ \text{indexSelectivity} &= S_{\text{esti}} \\ \text{indexCorrelation} &= 0 \end{aligned} \tag{4.2}$$

indexPages is automatically estimated by PostgreSQL (there is a general cost estimation function genericcostestimate), with selectivity estimation provided by uniform sampling in 4.1.1.

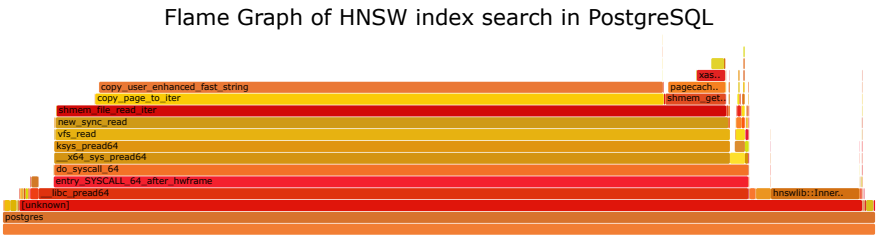

Figure 4.2 Flamegraph[56] of HNSW index search in PostgreSQL. Notice IO operations consume most of the CPU time. The coefficient constant values are set based on the execution time ratio of IO and CPU operations.

4.2.2 Distance Function Cost

PostgreSQL also provides API[53] for the distance function cost estimation. This general API is intended for function-related support, including selectivity estimation (for boolean functions, which are not relevant here), cost estimation, and more.

support_function(PG_FUNCTION_ARGS)

PG_FUNCTION_ARGS provides hints for support request types. Our implementation returns null for all requests except cost estimation. During cost estimation, the function arguments are provided. These may be a constant, an intermediate result not yet available, or a column. The length of the input vector is extracted here and scaled with a hardware-dependent constant c from Equation 3.4distance function cost, collected in Section 4.2.3.

4.2.3 PostgreSQL Fine-tuning

PostgreSQL has a few pre-defined cost constants originating from years ago, and it is recommended[54] to adjust them according to specific hardware configurations. For instance, seq_page_cost and random_page_cost should be equal[34] for data clusters on NVMe[34] SSD devices. Additionally, the ratio of seq_page_cost to cpu_operator_cost should be modified to account for the current state of modern CPUs. In practice, the values of seq_page_cost and cpu_operator_cost are adjusted according to profiling results[55].

Considering these constants, the cost of distance function calls, C_{dist}, can be determined by profiling vector search. All cost model parameters are available during the planning stage, and the model's output will guide PostgreSQL's execution plan selection.

Chapter 5 Evaluation

Despite a few early works[21-22] proposing cost models for high-dimensional indices like R-Tree, state-of-the-art high-dimensional databases do not employ cost evaluation for high-dimensional data. For instance, PASE[30], built as an extension to PostgreSQL, adopts the default cost estimate without filling the support function field. In this chapter, we empirically evaluate our abstraction by comparing cost and actual execution time in 5.2 and examining PostgreSQL's performance under our cost estimation in an end-to-end manner in 5.3. The latency introduced by our cost model is <1ms in all test cases, which is acceptable in all scenarios.

5.1 Experiment Setup

SIFT1M[45] is selected for its popularity among vector search benchmarks and sufficiency for cost model evaluation. All evaluations are conducted on an Azure VM, Standard_F64s_v2[57] (the same as VBase[17]), with 64 v-CPUs and 128 GiB memory running Linux Ubuntu 20.04 LTS.

5.2 Cardinality Estimation

For the mixed query, PostgreSQL calculates the cardinality of the scalar and vector filters individually and then merges the estimations. Thus, we only need to test the accuracy of cardinality estimation for the range filter solely. The sample set and test queries are generated using uniform sampling, and cardinality estimation is performed by executing the query on the sample set and collecting the number of results.

```
SELECT *
FROM sift1m
WHERE embedding <*> q < r;
```

Listing 5.1 Range query for CE evaluation

The accuracy of sampling is measured by Q-Error[58].

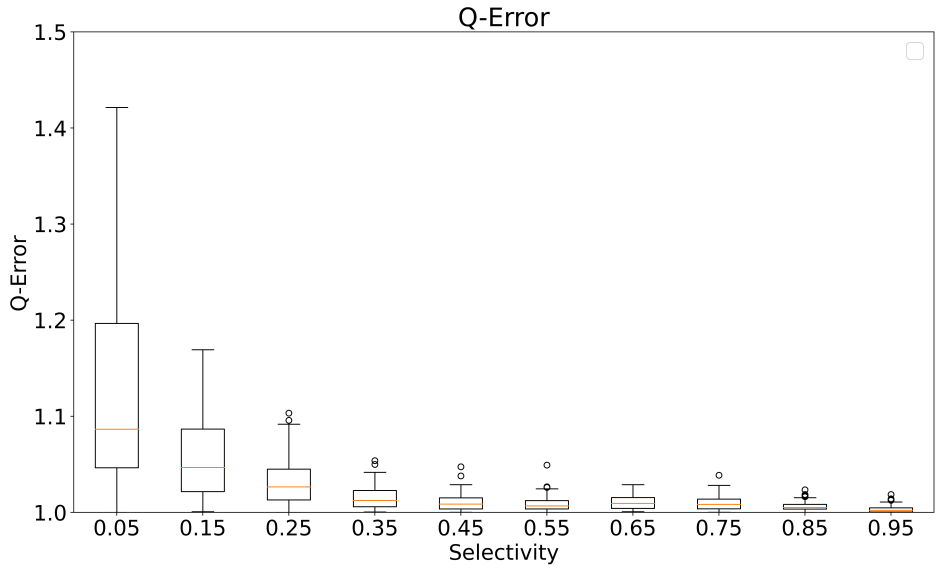

Figure 5.1 The box plot for Q-Error of selectivity estimation via basic sampling.

$$Q_{\text{err}} = \max\left(\frac{\text{Sel}_{\text{real}}}{\text{Sel}_{\text{esti}}}, \frac{\text{Sel}_{\text{esti}}}{\text{Sel}_{\text{real}}}\right) \tag{5.1}$$

We set the sample rate to 0.001 to accommodate online service scenarios, resulting in latencies of <1ms for 330k data in VBase[17]'s tests. Since the selectivity of range filters cannot be accurately controlled by the radius r of a hypersphere, different queries are split into buckets based on their actual selectivity.

5.3 Cost Model

Due to time constraints, we did not conduct a cost model evaluation on SIFT1M. Instead, we used the estimation data from VBase, which was also produced by a model with the same design. The cost model was tested on the Recipe table described below.

```
Recipe(recipe_id  BIGINT  SERIAL,
       images_embedding  FLOAT8[],
       popularity        FLOAT8)
```

In the table Recipe, the popularity column contains uniformly distributed random data. The cost model test is performed on two identical copies of the Recipe table. One copy has only a B-Tree index on the popularity column, while the other has an HNSW index[25] on the images_embedding column. Sequence table scans and parallel

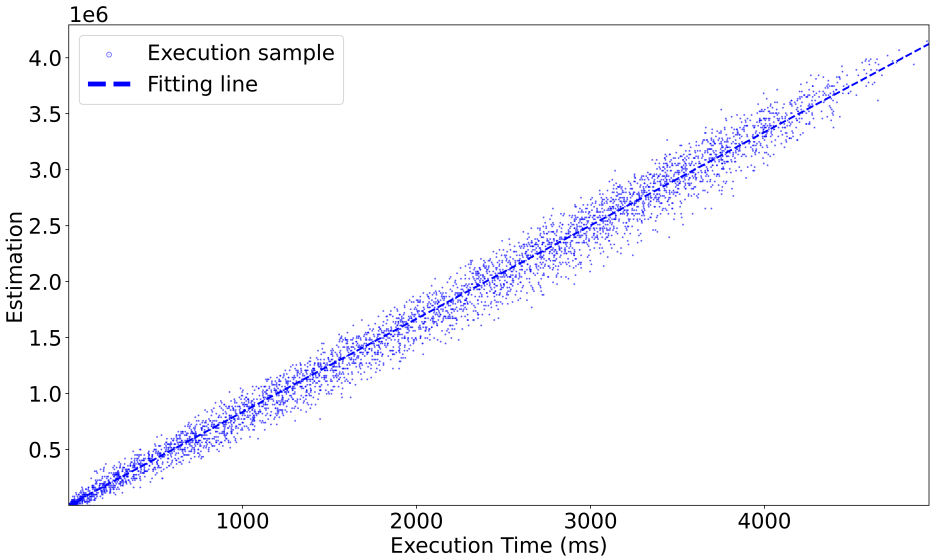

Figure 5.2 Comparing estimated cost and actual execution time, they fit well into a homogeneous line. This shows the cost model reflects end-to-end HNSW index scan cost well. The estimation value is relative to pre-defined constant in PostgreSQL.

table scans are disabled, and since seq_page_cost is set to equal random_page_cost (otherwise, Bitmap scans would be preferred for their better cache locality), PostgreSQL executes index scans on B-Tree and HNSW independently. The costs of pure B-Tree and HNSW index scans are collected in this setup. Cost estimation and execution time for each plan are collected using the EXPLAIN ANALYZE keyword in PostgreSQL.

```
SELECT recipe_id
FROM Recipe
WHERE distance(q, images_embedding) < r
AND popularity < p;
```

In PASE[30], the cost model is referred to as *default* since it does not implement a custom cost model. Instead, PostgreSQL reverts to the default estimation in this case, performing a B-Tree scan on the popularity column. Scanning the B-Tree first and applying a range filter on iteratively produced results is denoted as B-Tree(default), whereas scanning HNSW first and applying a scalar filter on results is labeled HNSW. The plan chosen by PostgreSQL (from the plans above) based on the cost model is marked as My Plan, and the optimal policy that consistently selects the faster scanning plan is labeled as Best Plan.

In 5.3, each point represents the average execution time of queries within the cor-

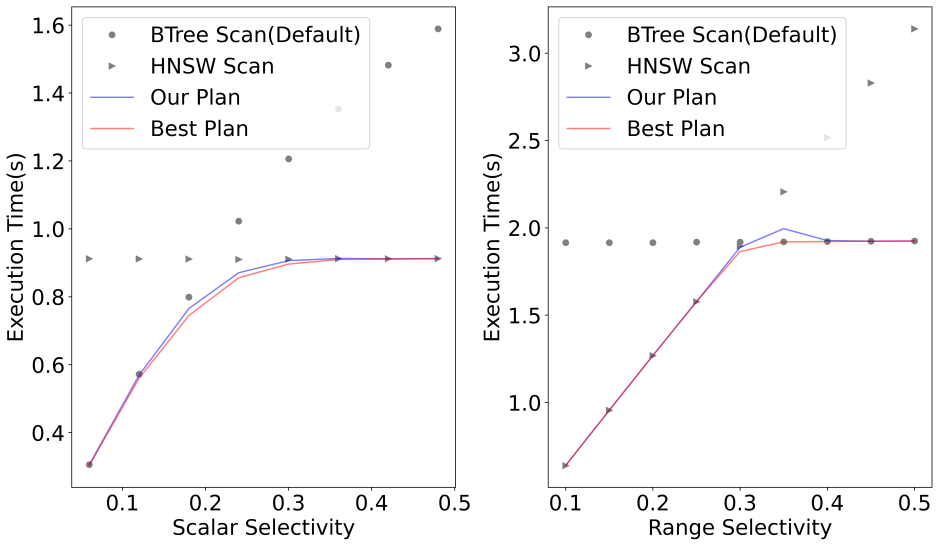

Figure 5.3 PostgreSQL's execution time with different planning. Because vector index is slower compared to scalar one with same selectivity, selectivity of range filter and scalar filter is set to 0.13 and 0.9 respectively to center the pivot in both graphs.

responding selectivity bucket. The best plan is computed afterward, as the faster option between the B-Tree and HNSW scans. Our plan is calculated during the query-planning stage (before execution) and selects the plan with the lower estimated cost. As seen in the figure, our planning approach under the abstraction is near-optimal. For queries near the intersection point and grouped into the same bucket, some may perform better under HNSW scan, while others perform better under B-Tree scan. Consequently, adhering to the same plan cannot always represent the best strategy for all queries in this bucket. Therefore, the optimal plan and our near-optimal planning might result in lower latency than the two basic scanning plans.

Chapter 6 Related Work

In this section, we provide an overview of related techniques in ANNS, DBMS cost models, and cost estimation of ANNS.

（1）High-dimensional indices

Quite a few high-dimensional indices have emerged over the years. They can be roughly categorized into three groups:

Tree Index: This group includes R-Tree[27], RD-Tree[40], TV-Tree[59], KD-Tree[24] [60], BKD-Tree[61], BP-Tree[62], APD-Tree[44], and more. These indices focus on dividing the high-dimensional metric space in a tree-like manner to narrow down the search range. Tree-based indices have clear navigation and traversal phases and can be effectively represented using our cost abstraction.

Graph Index: This group consists of NSW[46], HNSW[25], and others. They exploit the potential of the small-world effect[63-64] and the Voronoi graph[65] along with its dual Delaunay triangulation form, performing a greedy search over the graph to narrow down search subspace. As our evaluation shows, the graph-based index can be modeled using this abstraction.

Locality-Sensitive Hashing[66] typically maps high-dimensional data to lower-dimensional representations. Key methods[38,67] encompass Hamming-based LSH[68], Minkowski-based LSH[69], Angular-based LSH[70] techniques, Jaccard-based LSH[71], and more. In general, they are faster than tree and graph-based indices but have lower recall. The underlying structure of LSH techniques varies[38] from tree, graph, to sequence. As a result, their cost models should be customized on a case-by-case basis, although they can maintain a consistent abstraction.

（2）ANNS in relational databases

Recent research has centered on integrating high-dimensional indices into traditional relational DBMSs, including Elasticsearch[31], Jingdong Vearch[72], AnalyticDB-V[15], PASE[30], Milvus[16], and VBase[17]. AnalyticDB-V and PASE utilize PostgreSQL as a backend but do not provide proper query planning; Elastic supports vector search along with scalar filters; Milvus implements an efficient workaround for mixed queries but continues to treat the vector index as a black box; VBase presents

an iterative model for vector index and the cost model proposed in this paper.

Recent work focuses on integrating high-dimensional indices into traditional relational DBMS, including Elasticsearch[31], Jingdong Vearch[72], AnalyticDB-V[15], PASE[30], Milvus[16], VBase[17]. AnalyticDB-V and PASE take PostgreSQL as backend but did not provide proper query planning; Elastic began to support vector search plus scalar filter; Milvus efficiently employed workaround for mixed query, but still treated the vector index as a black-box; VBase implement an iterative model for vector index and cost model proposed in this paper.

（3）Cost Models

There is extensive research on cost modeling[73-74] for relational databases. Earlier studies on high-dimensional indices[21-22] focused on accurately estimating range query cardinality for prevalent indices at the time, such as X-Tree[75] and R-Tree[27]. Achieving this required precise estimation of hypersphere intersection between search subspace and hypersphere around query point, which in turn enabled the prediction of the number of page fetches. However, high-dimensional indices in relational databases were no longer a popular research topic until the recent advances in machine learning and neural networks. Over the past decade, the bottleneck in vector search has shifted from I/O to computation (CPU/GPU) due to the rapid growth of fast storage devices[34], necessitating adaptations in cost models.

（4）Cardinality Estimation of ANNS

Cardinality estimation for high-dimensional queries has recently gained attention[10], but it remains an unsolved problem. Previous work can be broadly categorized into three groups: 1. Sampling-based methods[51,76] focus on enhancing sampling performance by partitioning the search space using LSH or tree-like indices. 2. Kernel density estimation methods[77-78] apply kernels to one-dimensional distance functions among metric objects. 3. Deep methods[79-81] leverage machine learning and deep models, including XGBoost[82], LightGBM[83], and deep lattice networks[84], among others. Estimated cardinality from these models can be applied as the input of our cost abstraction for a better performance over sampling baseline.

Chapter 7 Conclusion

This paper identified a shared search pattern among high-dimensional indices and proposed a two-level unified cost abstraction. This abstraction has been implemented as an actual cost model in PostgreSQL, with its correctness and accuracy validated through end-to-end testing. Utilizing this cost model, current high-dimensional databases can achieve significant performance improvements with near-optimal planning. In addition to enhancing query execution in DBMS, this cost abstraction offers guidance for future optimization in ANNS-integrated databases.

However, this paper only implement cost models under the abstraction for two popular indices: IVFFlat and HNSW. Future work can expand the application of this abstraction for more indices.

Bibliography

[1] SALAU A O, JAIN S. Feature extraction: A survey of the types, techniques, applications[C/OL]//2019 International Conference on Signal Processing and Communication (ICSC). 2019: 158-164. DOI: 10.1109/ICSC45622.2019.8938371.

[2] ISINKAYE F O, FOLAJIMI Y O, OJOKOH B A. Recommendation systems: Principles, methods and evaluation[J]. Egyptian informatics journal, 2015, 16(3): 261-273.

[3] AMATO G, FALCHI F. knn based image classification relying on local feature similarity[C]//Proceedings of the Third International Conference on Similarity Search and Applications. 2010: 101-108.

[4] KIM J, KIM B S, SAVARESE S. Comparing image classification methods: K-nearest-neighbor and support-vector-machines[C]//Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics. 2012: 133-138.

[5] COVER T, HART P. Nearest neighbor pattern classification[J]. IEEE transactions on information theory, 1967, 13(1): 21-27.

[6] CHEN M, WANG S, TIAN L. A high-precision duplicate image deduplication approach.[J]. J. Comput., 2013, 8(11): 2768-2775.

[7] HUANG J T, SHARMA A, SUN S, et al. Embedding-based retrieval in facebook search[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 2553-2561.

[8] YUVAL G. Finding near neighbours in k-dimensional space[J]. Information processing letters, 1975, 3(4): 113-114.

[9] WEBER R, SCHEK H J, BLOTT S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces[C]//VLDB: volume 98. 1998: 194-205.

[10] QIN J, WANG W, XIAO C, et al. Similarity query processing for high-dimensional data[J]. Proceedings of the VLDB Endowment, 2020.

[11] BHATIA N, et al. Survey of nearest neighbor techniques[A]. 2010.

[12] SYRIOPOULOS P K, KOTSIANTIS S B, VRAHATIS M N. Survey on knn methods in data science[C]//Learning and Intelligent Optimization: 16th International Conference, LION 16, Milos Island, Greece, June 5–10, 2022, Revised Selected Papers. Springer, 2023: 379-393.

[13] LI J, LIU H, GUI C, et al. The design and implementation of a real time visual search system on jd e-commerce platform[C]//Proceedings of the 19th International Middleware Conference Industry. 2018: 9-16.

[14] LI S, LV F, JIN T, et al. Embedding-based product retrieval in taobao search[C]// Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021: 3181-3189.

[15] WEI C, WU B, WANG S, et al. Analyticdb-v: a hybrid analytical engine towards query fusion for structured and unstructured data[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3152-3165.

[16] WANG J, YI X, GUO R, et al. Milvus: A purpose-built vector data management system[C]//Proceedings of the 2021 International Conference on Management of Data. 2021: 2614-2627.

[17] ZHANG Q, XU S, CHEN Q, et al. Vbase : Unifying online vector similarity search and relational queries via relaxed monotonicity[C]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). 2023.

[18] GRAEFE G. Volcano— an extensible and parallel query evaluation system: volume 6[M/OL]. IEEE Educational Activities Department, 1994: 120–135. https://doi.org/10.1109/69.273032.

[19] DUSCHKA O M, GENESERETH M R. Query planning in infomaster[C]// Proceedings of the 1997 ACM symposium on Applied computing. 1997: 109-111.

[20] GROUP T P G D. Index cost estimation functions[EB/OL]. 2023[11.5.2023]. https://www.postgresql.org/docs/15/index-cost-estimation.html.

[21] BERCHTOLD S, BÖHM C, KEIM D A, et al. A cost model for nearest neighbor search in high-dimensional data space[C]//Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. 1997: 78-86.

[22] BÖHM C. A cost model for query processing in high dimensional data spaces[J].

ACM Transactions on Database Systems (TODS), 2000, 25(2): 129-178.

[23] KOENIGSTEIN N, RAM P, SHAVITT Y. Efficient retrieval of recommendations in a matrix factorization framework[C]//Proceedings of the 21st ACM international conference on Information and knowledge management. 2012: 535-544.

[24] BENTLEY J L. Multidimensional binary search trees used for associative searching[J]. Communications of the ACM, 1975, 18(9): 509-517.

[25] MALKOV Y A, YASHUNIN D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 42(4): 824-836.

[26] JEGOU H, DOUZE M, SCHMID C. Product quantization for nearest neighbor search[J]. IEEE transactions on pattern analysis and machine intelligence, 2010, 33(1): 117-128.

[27] BECKMANN N, KRIEGEL H P, SCHNEIDER R, et al. The r*-tree: An efficient and robust access method for points and rectangles[C]//Proceedings of the 1990 ACM SIGMOD international conference on Management of data. 1990: 322-331.

[28] PATRICK E A, FISCHER III F P. A generalized k-nearest neighbor rule[J]. Information and control, 1970, 16(2): 128-152.

[29] ILYAS I F, BESKALES G, SOLIMAN M A. A survey of top-k query processing techniques in relational database systems[J]. ACM Computing Surveys (CSUR), 2008, 40(4): 1-58.

[30] YANG W, LI T, FANG G, et al. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension[C]//Proceedings of the 2020 ACM SIGMOD international conference on management of data. 2020: 2241-2253.

[31] ELASTICSEARCH B. Elasticsearch[J]. software], version, 2018, 6(1).

[32] LAN H, BAO Z, PENG Y. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration[J]. Data Science and Engineering, 2021, 6: 86-101.

[33] OUARED A, OUHAMMOU Y, BELLATRECHE L. Costdl: a cost models description language for performance metrics in database[C]//2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, 2016: 187-190.

[34] XU Q, SIYAMWALA H, GHOSH M, et al. Performance analysis of nvme ssds

and their implication on real world databases[C]//Proceedings of the 8th ACM International Systems and Storage Conference. 2015: 1-11.

[35] CHAUDHURI S. An overview of query optimization in relational systems[C]// Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. 1998: 34-43.

[36] BABU B, CHANDRA N, GOPAL V, et al. Clustering algorithms for high dimensional data–a survey of issues and existing approaches[J/OL]. Special Issue of International Journal of Computer Science & Informatics (IJCSI), 2013: 2231-5292. DOI: 10.47893/IJCSI.2013.1108.

[37] ARORA S, CHANA I. A survey of clustering techniques for big data analysis[C]// 2014 5th International Conference-Confluence The Next Generation Information Technology Summit (Confluence). IEEE, 2014: 59-65.

[38] WU W, LI B. Locality sensitive hashing for structured data: A survey[A]. 2022.

[39] KAMEL I, FALOUTSOS C. Hilbert r-tree: An improved r-tree using fractals[R]. 1993.

[40] HELLERSTEIN J M, PFEFFER A. The rd-tree: An index structure for sets[R]. University of Wisconsin-Madison Department of Computer Sciences, 1994.

[41] GRAY R. Vector quantization[J]. IEEE Assp Magazine, 1984, 1(2): 4-29.

[42] FERN X Z, BRODLEY C E. Random projection for high dimensional data clustering: A cluster ensemble approach[C]//Proceedings of the 20th international conference on machine learning (ICML-03). 2003: 186-193.

[43] DEEGALLA S, BOSTROM H. Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification[C]//2006 5th International Conference on Machine Learning and Applications (ICMLA'06). IEEE, 2006: 245-250.

[44] MCCARTIN-LIM M, MCGREGOR A, WANG R. Approximate principal direction trees[A]. 2012.

[45] JEGOU H, DOUZE M, SCHMID C. Product quantization for nearest neighbor search[J]. IEEE transactions on pattern analysis and machine intelligence, 2010, 33(1): 117-128.

[46] MALKOV Y, PONOMARENKO A, LOGVINOV A, et al. Approximate nearest neighbor algorithm based on navigable small world graphs[J]. Information Sys-

tems, 2014, 45: 61-68.

[47] JOHNSON J, DOUZE M, JÉGOU H. Billion-scale similarity search with GPUs [J]. IEEE Transactions on Big Data, 2019, 7(3): 535-547.

[48] WANG Y, XIAO C, QIN J, et al. Consistent and flexible selectivity estimation for high-dimensional data[C]//Proceedings of the 2021 International Conference on Management of Data. 2021: 2319-2327.

[49] AGGARWAL C C. An efficient subspace sampling framework for high-dimensional data reduction, selectivity estimation, and nearest-neighbor search[J]. IEEE transactions on knowledge and data engineering, 2004, 16(10): 1247-1262.

[50] SUN J, LI G, TANG N. Learned cardinality estimation for similarity queries[C]// Proceedings of the 2021 International Conference on Management of Data. 2021: 1745-1757.

[51] HIRATA K, AMAGATA D, HARA T. Cardinality estimation in inner product space[J]. IEEE Open Journal of the Computer Society, 2022, 3: 208-216.

[52] BAYER R, MCCREIGHT E. Organization and maintenance of large ordered indices[C]//Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control. 1970: 107-141.

[53] GROUP T P G D. Function optimization information[EB/OL]. 2023[11.5.2023]. https://www.postgresql.org/docs/current/xfunc-optimization.html.

[54] GROUP T P G D. Query planning[EB/OL]. 2023[11.5.2023]. https://www.postgresql.org/docs/15/runtime-config-query.html.

[55] GREGG B. The flame graph[J]. Communications of the ACM, 2016, 59(6): 48-57.

[56] BRENDANGREGG. Flamegraph[CP/OL]. 2023. https://github.com/brendangregg/FlameGraph.

[57] PRIYASHAN 19, IAMWILLIEW, V REGANDOWNER, et al. Fsv2-series [EB/OL]. 2022[12.21.2022]. https://learn.microsoft.com/en-us/azure/virtual-machines/fsv2-series.

[58] MOERKOTTE G, NEUMANN T, STEIDL G. Preventing bad plans by bounding the impact of cardinality estimation errors[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 982-993.

[59] LIN K I, JAGADISH H V, FALOUTSOS C. The tv-tree: An index structure for

high-dimensional data[J]. The VLDB Journal, 1994, 3: 517-542.

[60] BROWN R A. Building a balanced kd tree in o (kn log n) time[A]. 2014.

[61] PROCOPIUC O, AGARWAL P K, ARGE L, et al. Bkd-tree: A dynamic scalable kd-tree[C]//Advances in Spatial and Temporal Databases: 8th International Symposium, SSTD 2003, Santorini Island, Greece, July 2003. Proceedings 8. Springer, 2003: 46-65.

[62] ALMEIDA J, TORRES R D S, LEITE N J. Bp-tree: An efficient index for similarity search in high-dimensional metric spaces[C]//Proceedings of the 19th ACM international conference on Information and knowledge management. 2010: 1365-1368.

[63] KUPERMAN M, ABRAMSON G. Small world effect in an epidemiological model[J]. Physical review letters, 2001, 86(13): 2909.

[64] WANG X F, CHEN G. Complex networks: small-world, scale-free and beyond [J]. IEEE circuits and systems magazine, 2003, 3(1): 6-20.

[65] ERWIG M. The graph voronoi diagram with applications[J]. Networks: An International Journal, 2000, 36(3): 156-163.

[66] DATAR M, IMMORLICA N, INDYK P, et al. Locality-sensitive hashing scheme based on p-stable distributions[C]//Proceedings of the twentieth annual symposium on Computational geometry. 2004: 253-262.

[67] JAFARI O, MAURYA P, NAGARKAR P, et al. A survey on locality sensitive hashing algorithms and their applications[A]. 2021.

[68] INDYK P, MOTWANI R. Approximate nearest neighbors: towards removing the curse of dimensionality[C]//Proceedings of the thirtieth annual ACM symposium on Theory of computing. 1998: 604-613.

[69] DATAR M, IMMORLICA N, INDYK P, et al. Locality-sensitive hashing scheme based on p-stable distributions[C]//Proceedings of the twentieth annual symposium on Computational geometry. 2004: 253-262.

[70] JI J, LI J, YAN S, et al. Super-bit locality-sensitive hashing[J]. Advances in neural information processing systems, 2012, 25.

[71] BAWA M, CONDIE T, GANESAN P. Lsh forest: self-tuning indexes for similarity search[C]//Proceedings of the 14th international conference on World Wide Web. 2005: 651-660.

[72] The Vearch Authors. Vearch: A distributed system for embedding-based vector retrieval[CP/OL]. 2021. https://github.com/vearch/vearch.

[73] SOLIMAN M A, ANTOVA L, RAGHAVAN V, et al. Orca: a modular query optimizer architecture for big data[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014: 337-348.

[74] GRAEFE G. The cascades framework for query optimization[J]. IEEE Data Eng. Bull., 1995, 18(3): 19-29.

[75] BERCHTOLD S, KEIM D A, KRIEGEL H P. The x-tree: An index structure for high-dimensional data[C]//Very large data-bases. 1996: 28-39.

[76] WU X, CHARIKAR M, NATCHU V. Local density estimation in high dimensions [C]//International Conference on Machine Learning. PMLR, 2018: 5296-5305.

[77] MATTIG M, FOBER T, BEILSCHMIDT C, et al. Kernel-based cardinality estimation on metric data.[C]//EDBT. 2018: 349-360.

[78] HEIMEL M, KIEFER M, MARKL V. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015: 1477-1492.

[79] WANG Y, XIAO C, QIN J, et al. Monotonic cardinality estimation of similarity selection: A deep learning approach[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020: 1197-1212.

[80] WANG Y, XIAO C, QIN J, et al. Consistent and flexible selectivity estimation for high-dimensional data[C]//Proceedings of the 2021 International Conference on Management of Data. 2021: 2319-2327.

[81] WANG W, ZHANG M, CHEN G, et al. Database meets deep learning: Challenges and opportunities[J]. ACM SIGMOD Record, 2016, 45(2): 17-22.

[82] CHEN T, GUESTRIN C. Xgboost: A scalable tree boosting system[C]// Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[83] KE G, MENG Q, FINLEY T, et al. Lightgbm: A highly efficient gradient boosting decision tree[J]. Advances in neural information processing systems, 2017, 30.

[84] YOU S, DING D, CANINI K, et al. Deep lattice networks and partial monotonic functions[J]. Advances in neural information processing systems, 2017, 30.