Elasticsearch Service

向量检索指南

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

向量检索指南
ES 向量集群配置评估

最近更新时间：2026-02-04 20:19:32

实现一个理想的向量检索业务应用，首先是合理的配置规划。相关评估请参见 集群规格和容量配置评估 > 向量搜索场景 。

ES 向量检索性能调优

最近更新时间：2026-02-04 20:19:32

在 AI 快速发展的时代，ES 被广泛用于向量搜索、多模态搜索和知识库构建，它特有的文本和向量混合搜索能力，兼顾召回率和搜索准确度，受到广大开发者的喜爱。那么如何更好的发挥 ES 向量搜索的性能优势，本文将重点介绍我们在实践中总结的一些调优经验。

提前做好配置规划

1. 合理的集群配置评估

为了确保 ES 向量搜索的性能，首先需要一个合理的集群配置，**重点是确保足够大的内存**，如果内存不够而导致经常性的搜索读盘，性能可能有十倍以上的下降。向量搜索的集群配置估算参见《 ES 向量集群配置评估 》。

2. 合理的索引规划

你需要提前评估业务数据和未来规模增长，提前进行索引和分片规划。如果数据规模很大，尽量避免都放在一个索引而导致每次检索查找整个索引。建议按分类如部门、商品类型等**切分索引**。分片过大或数量过多都会影响读写性能，如下是分片设置的实践建议：

- 单个分片大小建议在 **20GB – 50GB**，你可以据此初步确定索引的分片数量。
- **分片数尽量等于数据节点数**，若分片数较多，建议分片数为**数据节点的整数倍**，方便分片在数据节点均匀分布。
- 单节点所有索引的累计分片数**不要超过 1000 个**，集群总分片数控制在 **3 万个以内**。

增加副本可以提高查询吞吐量（QPS），因为搜索可以在主分片和副本分片上并行执行。

- 建议：在写入压力不大且需要高查询并发的场景下，适当增加副本数。

```
// 索引设置
PUT /my_index
{
  "settings": {
    "index": {
      "number_of_shards": 10, // 分片设置，根据未来规模增长合理评估分片数量
      "number_of_replicas": 1 // 副本设置，根据高可用和并发要求设置
    }
  },
  "mappings": {
    "properties": {
      ...
  }
}
```

3. 准备测试集与迭代调优

向量检索的调优是一个反复尝试优化的过程，你可以准备一个高质量的测试集，包含查询词、已知的相关文档（正样本）和不相关文档（负样本），明确召回率 Recall（召回正样本占全部正样本的比例）和准确率 Precision（查询结果中包含正样本的比例）的目标要求，通过“假设–验证–迭代”的过程，逐步找到适合自己业务的最佳配置。

向量索引 Mapping 设置

如下是一个常见的向量索引 mapping 设置举例：

```
// Mappping 设置
PUT /my-index
{
  "mappings": {
    "properties": {
      "category": {
        "type": "keyword"
      },
      "title": {
        "type": "text"
      },
      "title_vector": {
        "type": "dense_vector",
        "dims": 768, // 向量维度
        "similarity": "cosine",   // 向量相似度算法
        "index_options": {
          "type": "hnsw",// 索引算法
          "m": 16,  // HNSW 图中每个节点的最大连接数
          "ef_construction": 100  // 构建HNSW时，为每个新节点考察的候选邻居数量，影响构建索引的质量和速度
        }
      }
    }
  }
}
```

- **dims 向量维度：**越高则包含的信息越丰富，检索精度越高，但存储和计算成本也越高。你可以先从 384维、768维开始测试召回率/准确率，不达标则升维，达标可尝试降维。
- **index 设置：**默认值 true，如果设置index为false，将不会进行 KNN 搜索，只能进行暴力扫描（script_score）

- **type：**不同版本的默认值（8.13为hnsw，8.16为int8_hnsw，9.1.3为bbq_hnsw），你可以根据向量规模参考以下信息设置：
  - hnsw：建议向量数量在1亿以内采用
  - int8_hnsw：建议向量数量在1亿-20亿采用
  - bbq_hnsw：建议向量数量在十亿到百亿采用
  - bbq_disk：建议向量数量在十亿到千亿采用（diskbbq在 ES 9.2以上支持，云上版本预计Q1上线）
- **m：**默认值16，是每个节点在 HNSW 图中的最大连接数。较大的 m 会提高召回率和查询速度，但会增加索引时间和内存占用。大多数场景从 16 开始，如果对召回率要求极高，且可以接受更长索引时间和更大索引体积，可尝试增加到32或更高。
- **efConstruction：**默认值100，是HNSW 索引构建时，为每个新节点寻找连接时考察的候选邻居数量。增大此值会提升索引质量和召回率，但会延长索引构建时间。可以从 100 开始，如果数据分布复杂或对精度要求很高可以设置200或更高。

批量写入向量

1. 原因说明

对于向量搜索场景而言，强烈建议采用批量写入。因为批量写入是ES高吞吐场景的核心优化手段，通过减少网络交互、优化磁盘I/O和降低索引开销，它能显著提升数据写入效率并保障集群稳定。

2. 批量写入设置

使用 Bulk 批量写入，可以从每批数据 500 -1000 条左右开始测试，观察集群的 CPU 和内存负载，逐步增加直到性能不再提升或出现拒绝请求。

```
// 批量写入
POST /_bulk
{ "index" : { "_index" : "my-index", "_id" : "1" } }
{ "category":"cat", "title": "慵懒的猫", "title_vector": [0.1, 0.2, ... ]
}
{ "index" : { "_index" : "my-index", "_id" : "2" } }
{ "category":"dog", "title": "遛弯的狗", "title_vector": [0.3, 0.4, ... ]
}
//... 更多数据 ...
```

3. 其他可调整参数（非必须）

- **调整刷新间隔**

  refresh_interval 默认为 1s，使新写入的数据对搜索可见，但频繁刷新会生成大量小 segment，影响向量索引构建和查询性能。因此建议在批量导入数据期间，可以将 refresh_interval 设置为一个较大的值（如 30s

或 60s ）或 -1，导入完成后再恢复。请注意，refresh_interval 为 -1 期间新写入的数据无法被搜索到，直到执行手动刷新或恢复自动刷新后下一次刷新发生。

- **临时关闭副本**

  写入大量数据时，也可以先设置 number_of_replicas 为 0，写入完成后再恢复副本数。

```
// 索引设置
PUT /my_index
{
  "settings": {
    "index": {
      "number_of_shards": 10, // 分片设置，根据未来规模增长合理评估分片数量
      "number_of_replicas": 0, // 初期批量写入时，临时关闭副本，写完再恢复
      "refresh_interval": -1 // 通过禁止自动刷新，来提升写入性能
    }
  },
  "mappings": {
    "properties": {
      ...
  }
}
```

向量搜索参数设置

这是一个简单的向量搜索语句：

```
// 向量搜索 - 只需召回少量非text字段
GET /my-index/_search
{
  "size" : 3,
  "knn": {
      "field": "title_vector",
      "query_vector": [-5, 9, -12, ...],
      "k": 10, // 返回的 top k 结果数量
      "num_candidates": 100  // 指定每个分片上要检索的候选向量数量
  }
  "fields": ["category"],
  "_source": false
}
```

```
// 向量搜索 - 召回字段较多或有text字段时
GET /my-index/_search
{
  "size" : 3,
  "knn": {
      "field": "title_vector",
      "query_vector": [-5, 9, -12, ...],
      "k": 10, // 返回的 top k 结果数量
      "num_candidates": 100  // 指定每个分片上要检索的候选向量数量
  }
  "_source": {
    "excludes": ["title_vector"] // 排除掉向量字段
    // "excludes": ["*_vector"] // 使用通配符，例如排除所有以 vector 开头的字段
    // "_source": ["doc_id", "title"] // 明确列出需要返回的业务字段
    // "_source": false // 完全不返回原始文档数据，拿到文档 _id 后，会去另一个数据库如 MySQL根据查询详情
  }
}
```

- **k**

  查询时，召回的 TOP 结果数量，根据需要设置。
- **num_candidates**

  指定每个分片上要检索的候选向量数量。这个值必须大于等于 k。增大 num_candidates 会提高召回率（特别是在量化、有过滤器的情况下），但会增加查询延迟。通常建议设置为 k 的 5 到 10 倍，在召回率和性能间平衡。
- **避免返回向量字段**

  向量字段通常很大，如果不需要返回向量数据，可参考如下建议：
  - 如果只需召回 id 等少量非text字段（如keyword、numeric等启用doc_values的字段类型）或显式存储字段store: true，可以采用fields明确指定召回字段，ES会直接查询doc values或stored存储值，避免解析 _source的开销，性能更好。
  - 如果召回字段较多或有 text 字段时，可以使用_source: false、_source: exclude 或 _source: ["field1", "field2"] 来排除向量字段，减少网络传输和序列化开销。

向量量化和过采样

```
// Mapping 设置（向量采用 int8 量化）
PUT /my-quantized-index
```

```
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
      },
      "title_vector": {
        "type": "dense_vector",
        "dims": 768,
        "index": true,
        "similarity": "cosine",
        "index_options": {
          "type": "int8_hnsw", // 启用 8位标量量化 HNSW 索引
          "m": 16,
          "ef_construction": 100
        }
      }
    }
  }
}

// 向量搜索（过采样）
GET my-quantized-index/_search
{
  "knn": {
    "field": "title_vector",
    "query_vector": [0.15, 0.50, ..., 0.05], // 查询向量
    "k": 10,                 // 返回的 top k 结果数
    "num_candidates": 100,    // 需要 >= k
    "rescore_vector": {
      "oversample": 2.0 // 过采样系数
    }
  }
}
```

1. 什么是量化

当 ES 的向量搜索规模在**亿级甚至十亿级时**，我们面临两个挑战：内存占用高和计算速度慢。ES 的向量量化是此时的重要优化策略，它巧妙结合了量化的速度优势和原始向量的精度优势，在性能和准确率之间取得平衡。**量化**

（Quantization）本质是一种有损压缩技术，核心思想是将高精度的向量数据（通常是 32 位浮点数，float）转换为低精度表示（如 8 位整数，int8），它可以在召回率损失很少的情况下，大幅降低内存使用并提高查询速度。

2. 量化的内存和存储占用变化

如果你在索引 Mapping 设置中指定了量化，ES 在构建索引时除了生成原始精度数据存储外（假设为float32），还会额外生成一份对 float32 的量化精度数据存储。因此对于 float32 向量，int8, int4, bbq 量化分别可以将所需**内存减少** 4、8、32 倍，但也会降低向量精度并**增加磁盘空间**（分别增加 25%、12.5% 或 3.125%）。例如当使用 int8 量化 40GB 的浮点向量时，量化后的向量将额外占用 10GB 的磁盘空间，总磁盘使用量为 50GB，但内存使用量将减少到 10GB。

3. 量化的粗排与过采样

为了解决量化带来的精度损失问题，ES 的向量搜索分为了近似粗排和精确精排两个阶段：

- **粗排阶段：**ES 使用量化后的查询向量，在基于量化数据构建的 HNSW 图索引上搜索，在每个分片完成搜索后，会得到一个包含 num_candidates 个文档 ID 和近似得分的列表。
- **精排阶段：**通过**设置 rescore_vector 指定 oversample 过采样系数**，ES 将对每个分片返回的 num_candidates个文档中的前 k*oversample 个文档，从磁盘上读取原始的 float32 向量，使用原始查询向量与这些原始文档向量进行高精度相似度计算，从中选出 Top k 个结果。

显然，过采样机制结合了使用量化向量进行近似检索的性能和内存优势，和使用原始向量对最佳候选者进行重新评分的准确性。实际上int8通常不需要过采样，int4通过设置1.5-2倍可以获得更高的精确度和召回率，bbq通常需要过采样，3×-5× 过采样通常足够了。关于过采样的更详细说明请参见 量化向量的过采样和重评分 。

4. 量化的召回率评估

就我们的实践经验而言，**不量化情况下召回率99%的话，int8量化召回率约96-97%，bbq量化召回率约90-94%**，你需要特别留意，量化的召回率和数据规模有较高的相关性，数据集越大，量化带来的召回率损失越小，反之则越大。因此你应该在尽量大的数据集上来验证最终的召回率效果。

预热文件系统缓存

1. 背景

ES 依赖操作系统的 page cache 来加速磁盘索引文件的读取。在默认情况下，索引文件是在被访问时才被加载到缓存中。这可能导致一个问题，当主机操作系统重启后，Page Cache 会被清空，在缓存重新被“预热”期间，搜索性能可能显著下降，因为系统需要频繁地从磁盘读取数据。

2. 解决方案

可以通过设置 index.store.preload，在索引打开阶段，主动将指定的关键数据文件预先加载到内存中，从而避免在后续搜索中产生耗时的磁盘I/O操作，尤其适用于那些被频繁搜索的“热”索引。

> ⚠ **注意：**

1. 如果文件系统缓存容量不足以容纳所有数据，则在过多索引或文件上过早地将数据加载到 Page Cache 中会降低搜索速度。请谨慎使用。
2. 如果你使用量化索引，只需预加载相关的量化值和索引结构，例如 HNSW 图。预加载原始向量（vec）是不必要的，而且可能适得其反，因为对原始向量进行预加载可能导致操作系统从缓存中移除重要的索引结构。

3. 向量搜索相关的文件类型

index.store.preload 具体加载哪些索引文件，请参考下面的文件扩展名说明：

文件扩展名	文件作用
vex	存储HNSW图结构文件。
vec	所有非量化向量值。包括所有元素类型：float、byte 和 bit。
veq	量化索引的量化向量：int4 或 int8。
veb	量化索引的二进制向量：bbq。
vem、vemf、vemq、vemb	元数据，通常很小，不需要预加载。

4. 预加载设置

index.store.preload 是一个静态设置，意味着它只能在索引创建时或在配置文件中指定，无法在索引创建后动态修改，对于非量化场景一般我们建议可以简单设置为“ve*”，对于量化场景可以逐个指定（比如指定屏蔽vex，vec）。
通常我们建议同时设置 mmapfs，它将索引文件直接映射到进程的虚拟内存地址空间。之后，ES 可以像访问普通内存一样读写文件（实际的数据加载由操作系统通过缺页中断机制完成），mmapfs 和 preload 结合使用，可以实现高效的查询和无冷启动的平滑体验，但是会增加节点重启时间。

```
// 索引创建时，设置预加载
PUT /my-preloaded-index
{
  "settings": {
    "index.store.preload": ["ve*"], // 预加载
    //"index.store.preload": ["vex", "vec"], // 预加载，也可以精细化指定
    "index.store.type": "mmapfs" // 将索引文件映射到进程虚拟内存地址空间
  },
  "mappings": {
    ...
```

```
  }
}
```

如果你想对集群上创建的所有新索引生效，可以在全局配置文件 config/elasticsearch.yml 中设置：

```
index.store.preload: ["ve*"] // 预加载特定数据文件到 Page Cache
index.store.type: mmapfs // 将索引文件映射到进程虚拟内存地址空间
```

对于已经存在的索引，必须先关闭它才能修改此设置：

```
// 1. 关闭索引
POST /my-existing-index/_close

// 2. 更新索引设置
PUT /my-existing-index/_settings
{
  "index.store.preload": ["ve*"] // 预加载特定数据文件到 Page Cache
  "index.store.type": "mmapfs" // 将索引文件映射到进程虚拟内存地址空间
}

// 3. 重新打开索引。打开过程中会触发预加载操作，打开速度会变慢。
POST /my-existing-index/_open
```

减少索引段的数量

ES 分片由段（segment）组成，段是索引中的内部存储元素。对于近似 KNN 搜索，ES 将每个段的向量值存储为单独的 HNSW 图，因此 knn 搜索必须检查每个段。虽然 knn 搜索并行化使得跨多个段的搜索速度大大提升，但如果段的数量较少，knn 搜索的速度仍然可以提升数倍。默认情况下，ES 通过后台合并定期将较小的段合并为较大的段。如果这还不够，你可以采取以下明确的步骤来减少索引段的数量。

1. 提高段的最大值

ES 提供了许多可调参数来控制合并过程,一个重要参数是 index.merge.policy.max_merged_segment，它控制合并过程中创建段的最大值，通过提高该值可以减少索引中段的数量。该值默认为 5 GB，这对于高维向量来说可能太小，建议将此值增加到 10 GB 或 20 GB，有助于减少段数。这是一个静态设置，因此如果对已经存在的索引，需要先关闭索引再设置：

```
// 1. 关闭索引
POST /my-index/_close
```

```
// 2. 更新索引设置
PUT /my-index/_settings
{
    "merge.policy.floor_segment": "300mb",    // 段的最小值
    "merge.policy.max_merged_segment": "10g" // 段的最大值
}

// 3. 重新打开索引
POST my-index/_open
```

2. 在批量索引期间创建大段

常见做法是先执行初始批量写入和创建索引，除了依赖后台定期段合并外，你还可以调整索引设置以鼓励 ES 创建更大的初始段：

- 确保批量上传期间不进行任何搜索，并将 index.refresh_interval 设置为 -1。这可以防止刷新操作，避免产生额外的段。
- 为 ES 分配一个较大的索引缓冲区，以便它在刷新之前可以接收更多文档。可以将 indices.memory.index_buffer_size 设置为堆大小的 10%，对于像 32 GB 这样较大的堆大小，这通常足够了。要允许使用完整的索引缓冲区，你还应该限制 index.translog.flush_threshold_size 小于 indices.memory.index_buffer_size。

3. 强制段合并(Force Merge)

如前所述，向量索引构建在每个段上，段越多查询时需要搜索的图就越多，性能越差。建议在批量写入数据后，或在数据不再频繁更新的索引上，执行强制合并。对于只读索引合并为 1 个 segment：POST /my-index/_forcemerge?max_num_segments=1。注意：强制合并是一个重资源操作，应在低峰期执行。

设置 preload 提升混合搜索性能

下面是一个典型的前置过滤搜索：

```
// 前置过滤搜索
GET  /my-index/_search
{
    "knn":{
        "field":"title_vector",
        "query_vector": [-5, 9, -12, ...],
        "k": 10,
        "num_candidates": 100,
        "filter":{
            "term":{
```

```
                "category":"cat"
            }
        }
    }
}
```

开源 ES 在前置过滤时，会先进行标量过滤，通过倒排链获取全部满足条件的 DocID，然后进行 KNN 查询，每找到一个相邻的向量就去标量过滤生成的DocID集合中检查是否存在，一直到获取 TOP N 个结果。这种情况下如果标量过滤结果集很大，比如查询性别为“男”的结果可能有亿级，将导致查询性能的显著下降。
针对混合搜索中前置过滤的性能问题，**腾讯云ES自研了预过滤优化**，你可以通过设置preload来启用，preload是动态参数，不需要重新打开索引。

```
// 预过滤优化
PUT /my-index/_settings
{
    "index.query.knn.optimize_prefilter_enable": "true"    // 启用预过滤优化，提升前置过滤的性能
}
```

使用 GPU 进行模型推理

向量的生成（Embedding）需要大量的计算，社区版 ES 支持机器学习节点专门用于向量模型推理，但是暂时还不支持 GPU 推理。
好消息是腾讯云 ES 从 8.16 版本开始已经支持了机器学习节点的 GPU 推理，这可以大大加速向量的推理速度，**推理性能较 CPU 提升 30 倍**，同时**性价比较 CPU 有超过 10 倍的提升**。因此在大规模向量生成的情况下，建议采用腾讯云 ES 机器学习节点进行 GPU 推理，方法是在购买ES集群时启用机器学习节点，并选择GPU机型，对于存量集群可以通过调整配置来启用机器学习节点。

特别提示：谨慎使用 script_score 执行向量搜索

```
// 采用 script_score 进行向量暴力搜索举例
GET /my_index/_search
{
  "query": {
    "script_score": {
      // 1. 定义扫描范围，这里是全表扫描
      "query": { "match_all": {} },
      // 2. 向量计算脚本
      "script": {
```

```
          // 调用内置函数计算余弦相似度
          // doc['title_vector'] 是文档中的向量
          // params.query_vector 是外部传入的查询向量
          // + 1.0 是为了保证得分为正数（余弦相似度范围是 -1 到 1）
          "source": "cosineSimilarity(params.query_vector,
doc['title_vector']) + 1.0",
          "params": {
            "query_vector": [0.12, 0.34, -0.55, ...]
          }
        }
      }
    },
    "size": 10
}
```

ES 的 script_score 查询是一项强大的高级功能，在文本搜索场景深受 ES 开发者喜爱，它允许你使用脚本来自定义文档的评分计算方式，可以结合文本相关性与业务指标来实现复杂的排序逻辑，比如你可以结合用户画像和点击行为数据来更好的优化排序，以实现更精准的搜索和推荐，这给我们的业务实现带来了很大的灵活性和实用性。
但是当我们把 script_score 用于向量搜索时，它将执行**暴力扫描**（即使你在索引 mapping 中将索引类型设置为 hnsw），而不是你在索引设置中定义的近似 KNN 搜索，脚本会对范围内的每一个文档，读取其存储的 title_vector 字段值，并调用内置的向量函数（如 cosineSimilarity）与传入的查询向量参数进行计算，向量搜索性能会急剧下降，因此，除非在小数据集 (<10万) 或必须精确的场景，我们一般不建议将 script_score 用于向量搜索。