蚂蚁金服ZSearch在向量搜索上的探索

吕梁
通用搜索平台基础架构负责人， 蚂蚁金服

个人介绍

- 17年加入蚂蚁金服做基于ElasticSearch的通用搜索平台ZSearch

- 主要负责
  - 基础平台架构及集群调度
    - 从物理机到容器，再到K8s
  - ES性能研究及排序插件实现

目录：

1. ZSearch平台架构
2. 向量检索需求
3. 向量检索基本概念
4. KNN算法
5. ANN算法
6. 总结

1. ZSearch平台架构

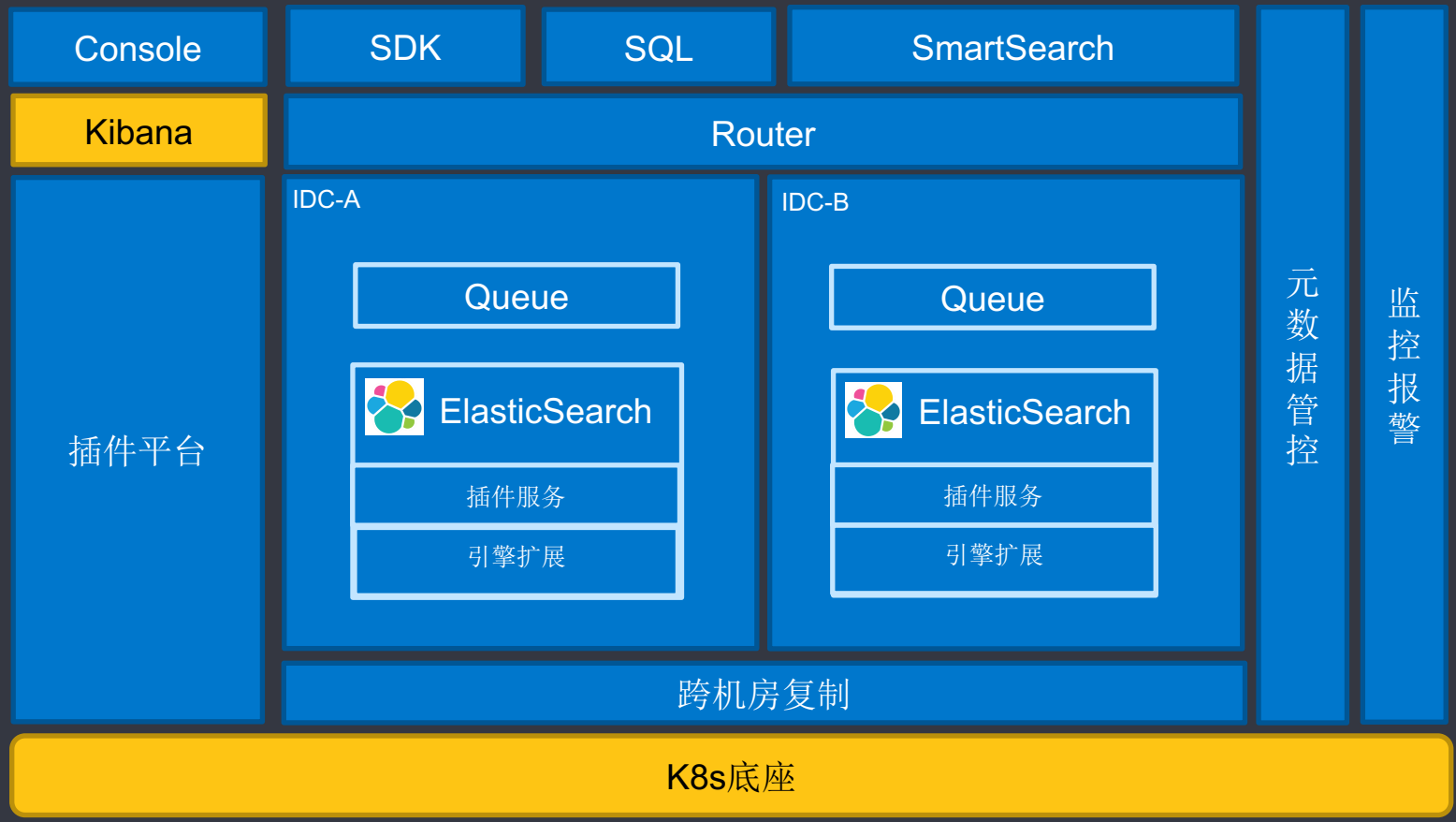
ZSearch 解决的痛点

- 当我们的ES集群越来越多，用户越来越丰富
  - 如何管理集群
  - 如何方便用户快捷接入
  - 如果支持用户丰富的场景

平台化

ZSearch平台架构

基于ES的金融级搜索平台



elastic

2. 向量检索需求

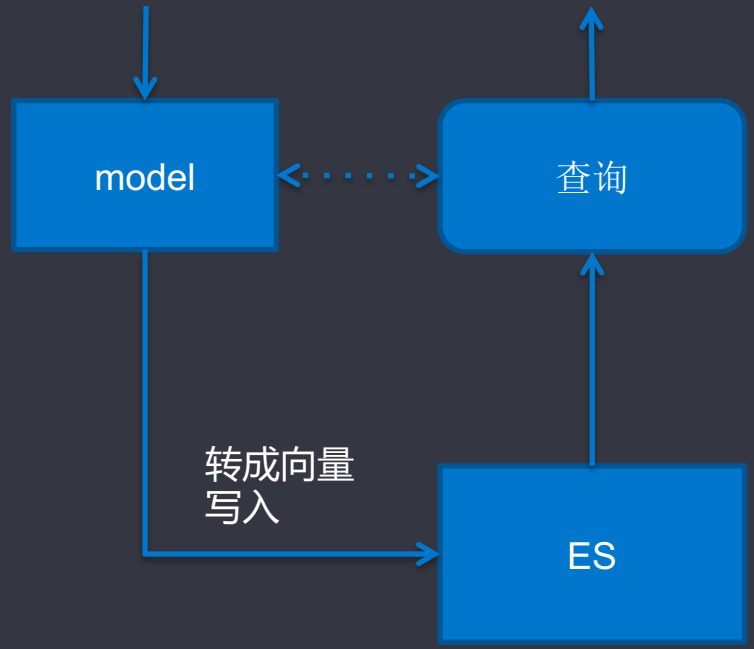
向量检索需求

- 如何查找相似图片
- 如何查找类似语音
- 问答系统如何查询问题的答案

将查询内容转换成向量，查找邻近

ZSearch向量场景

- 将人脸图片数据转换成向量，识别罪犯的信息
- 将文字使用TextCNN模型转换成向量，识别相似的敏感内容
- 把事件item的特征转为向量存储在ES中，把人的特征转为向量存储在HBASE上，进行u2i的推荐
- 海外轻量化人脸支付
- …



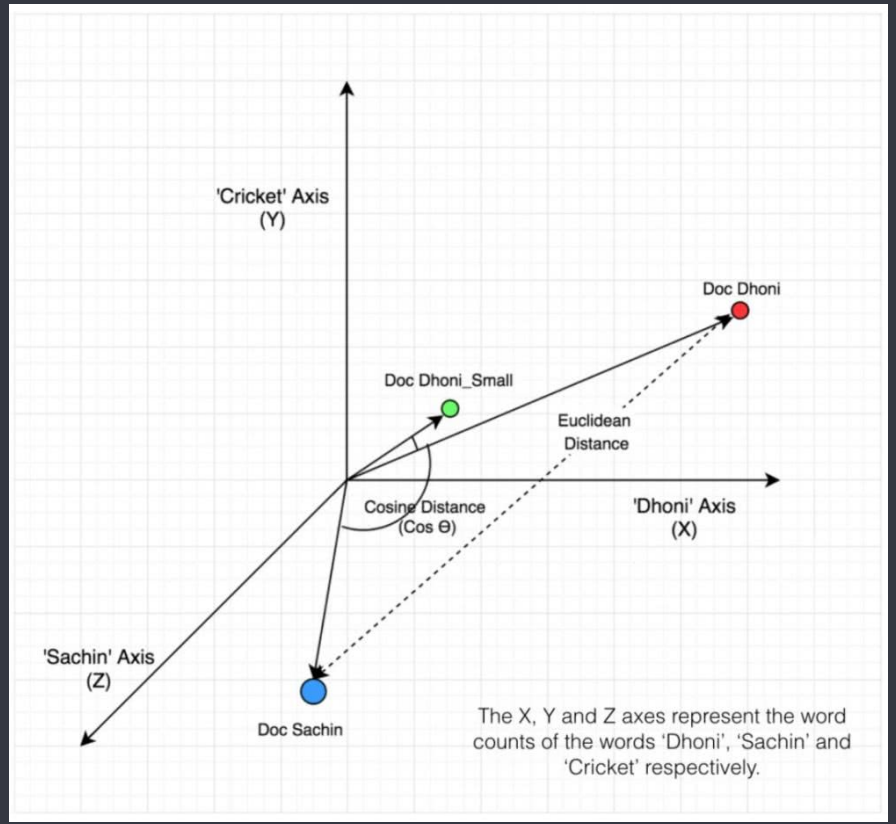
3. 向量检索基本概念

向量检索基本概念

- 向量的定义

$$\vec{a} = (x_1, x_2, x_3 \ldots)$$

- 相似性度量
  - 余弦距离
  - 欧拉距离
  - 汉明距离
  - …
- 应用场景
  - 推荐系统
  - 图片识别
  - 问答系统
  - …



elastic

相似度计算公式

欧式距离(Euclidean Distance)

$$d_{12} = \sqrt{\sum_{k=1}^{n}(x_{1k} - x_{2k})^2}$$

夹角余弦(Cosine)

$$\cos\theta = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2}}$$

汉明距离(Hamming distance)

$d(x,y) = \sum x[i] \oplus y[i]$

杰卡德相似系数(Jaccard similarity coefficient)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

4. KNN算法

KNN算法

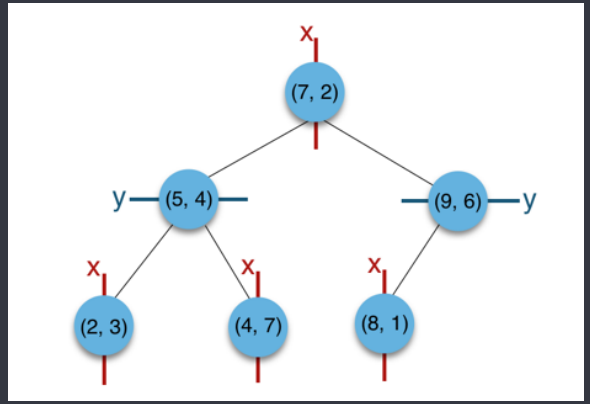
k-NearestNeighbor

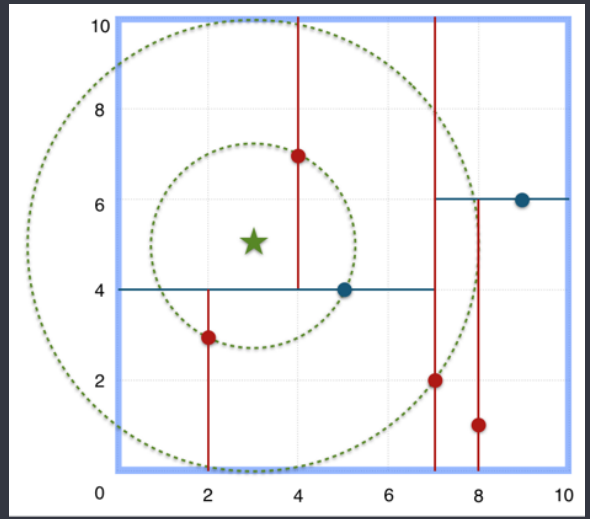
- 算法需要解决的就是：
  - 给定距离公式
  - 查找最邻近的K个数据

- 常用算法
  - KDTree
    - 一种平面划分树，可以通过剪枝减少查询次数
  - Brute Force
    - 顾名思义，就是暴力比对每一条数据，然后排序获得结果

KDTree

- 以二维平面点(x,y)的集合(2,3)，(5,4)，(9,6)，(4,7)，(8,1)，(7,2)为例
- 构建树
  - 按照x排序，确定中间值7，其他坐标分两边
  - 第二层按照y排序，第三层按照x排序
  - 并且在构建时维护每个节点中的x最大最小，y最大最小四个值
- 搜索(3,5)的最近邻
  - 到根节点距离为5，所以根节点的右子树都不需要查找了
  - 跟(5，4)节点比较，(7,2)被排除
  - 遍历完(2,3),(4,7)，最近点(5,4) 返回





elastic

KDTree

- 构建复杂度
  - O(nlogn)
- 查询复杂度
  - O(kn^((k-1)/k))
- Lucene BKDTree
  - BKD也是多个KDTREE，然后持续merge最终合并成一个。
  - 用于数字类型，GEO类型的查询
  - 在BKDWriter中，*MAX_DIMS* = 8
- 结论：如果向量维度较高，KDTree并不适合，因为查询复杂度变高，而且实现复杂

ZSearch实现Brute Force插件

- 实现ES的ScriptPlugin插件
- 直接使用es的binary类型
- 并且打开doc_value，直接读取lucene中的binaryDocValue
- 查询使用function_score，script_field，script_query 灵活组合向量相似度查询

```
// function_score中可以使用的script
implements ScoreScript.LeafFactory

// script field中可以使用的script
implements SearchScript.LeafFactory

// script query中可以使用的script
implements FilterScript.LeafFactory
```

SIMD加速

- SIMD
  - 单指令多数据流
  - 指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。
- Vector API
  - OpenJDK project Panama
  - 使用AVX2指令，向量查询RT性能提升6倍
  - 100w 256维 float数据，单分片，查询大概是260ms

```
import jdk.panama.vector.*;
import jdk.panama.vector.Vector.Shape;

Species = FloatVector.species(Shape.S_256_BIT);

//求平方和
vecSquare = Species.broadcast(0);
for (i = 0; i + (Species.length()) <= size; i += Species.length()) {
    // 可以直接从byte数组计算，减少一步byte[]转float[]
    vec = Species.fromByteArray(array, i >> 2);
    vecSquare = vec.mul(vec).add(vecSquare);
}
sum = vecSquare.addAll();
```

汉明距离优化

```
// 两个等长字符串之间的汉明距离是两个字符串对应位置的不同字符的个数。
// 对应到byte[]，就是每个byte异或以后，1的个数
// 使用java的unsafe加速计算
long tmp = unsafe.getLong(bytes, i*8L + offset + unsafe.ARRAY_BYTE_BASE_OFFSET );
res += Long.bitCount(tmp ^ longQueryVector[i]);
```

4. ANN算法

ANN算法

Approximate Nearest Neighbor

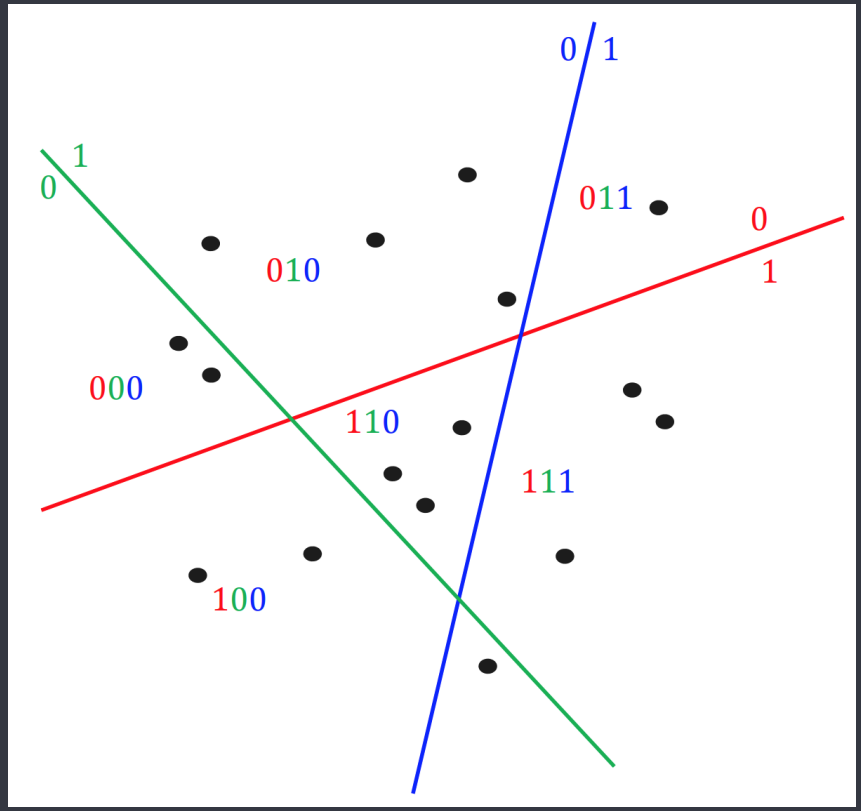
- 可以看到KNN的算法会随着数据的增长，时间复杂度也是线性增长
- 例如在人脸支付，推荐场景中，需要更快的响应时间，允许损失一些召回率
- ANN的意思就是近似K邻近，不一定会召回全部的最近点
- ANN的算法较多，接下来介绍开源的ES ANN插件
  - 基于hash的LSH
  - 基于编码的IVFPQ
  - 基于图的HNSW
- 然后介绍ZSearch开发的ANN插件(适配达摩院proxima向量检索引擎的HNSW算法)

开源ANN算法介绍

LSH算法

Local Sensitive Hashing

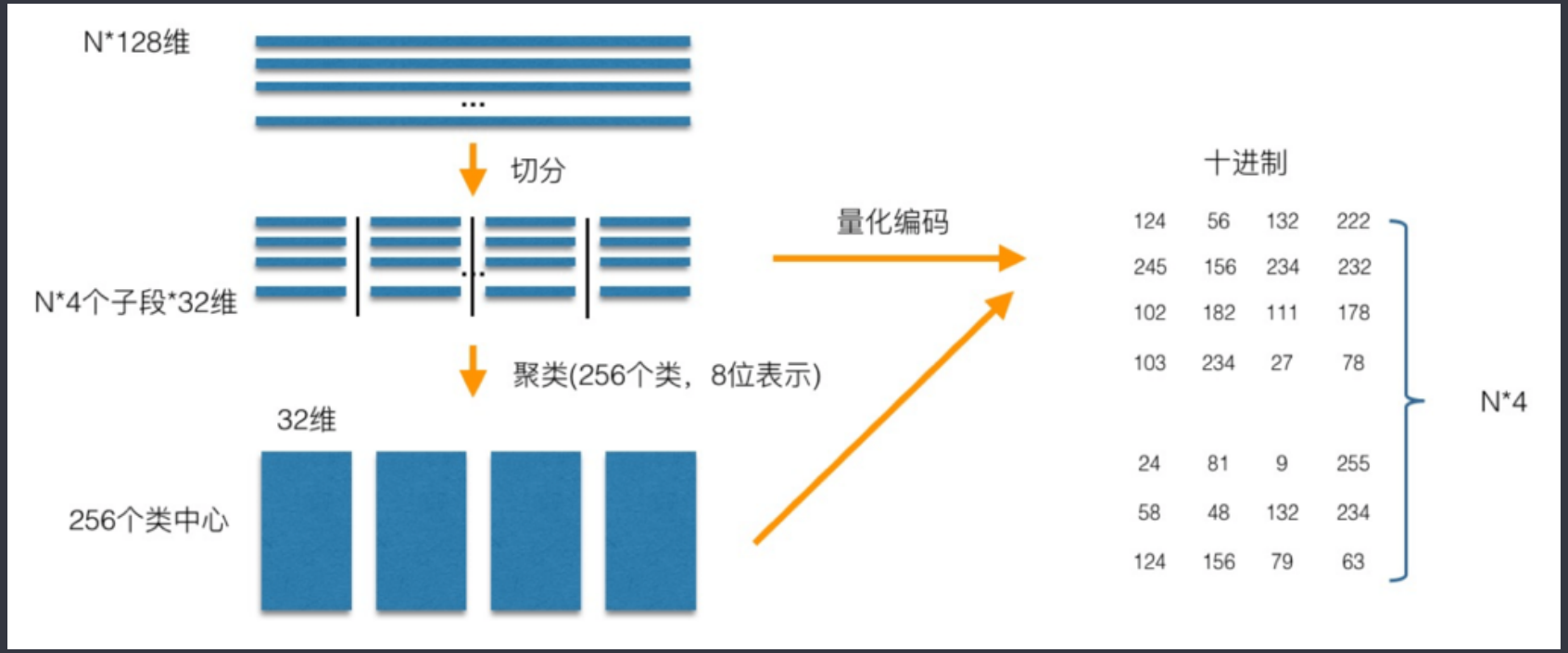
- 一个平面函数就是一个HASH函数
  - 两个点在越多的平面同侧，就说明这个两个点靠的越近
- 生成这个平面函数的方式有很多，最简单的是抽样一些点，取中位线
- 查找的时候，只要查找该点的邻近区域(图中数字重复度较高的一些)



elastic

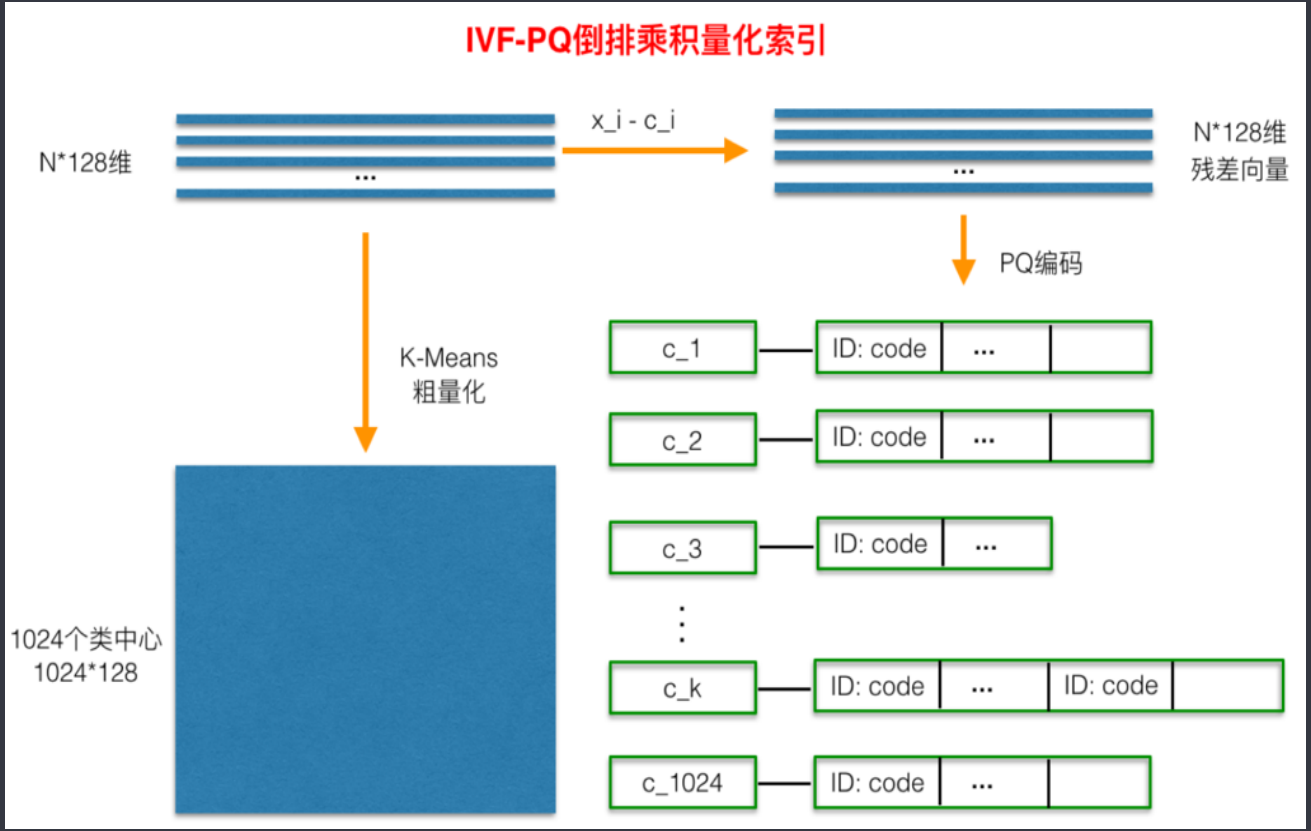
IVF-PQ算法

PQ编码



elastic

IVF-PQ算法

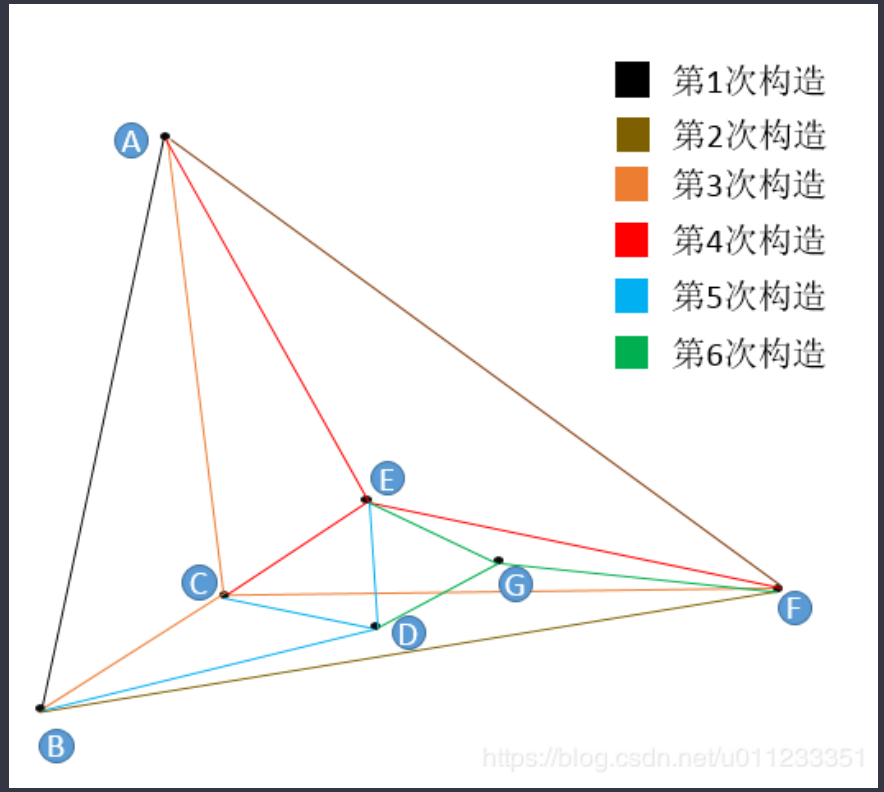


elastic

HNSW算法

NSW算法

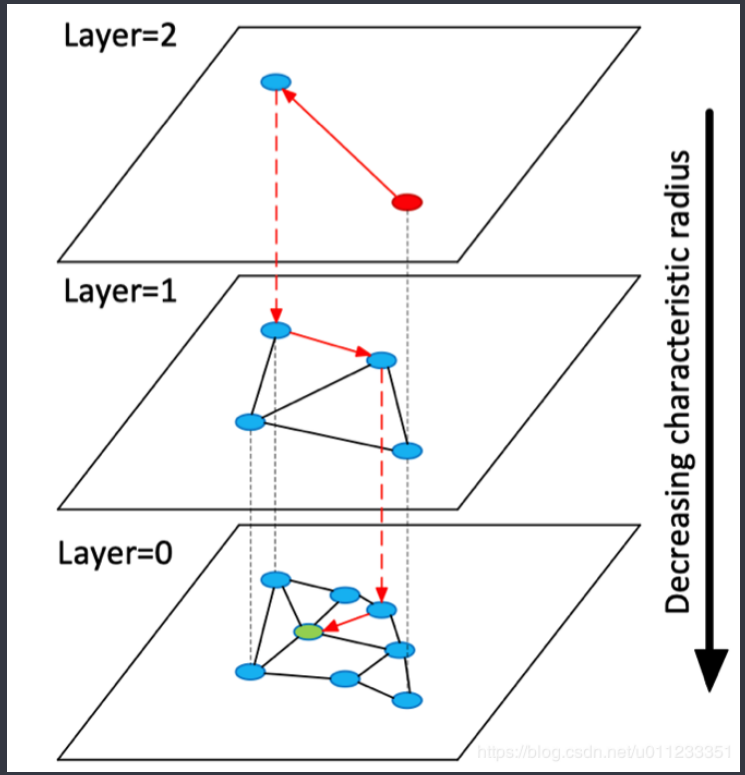
- 顺序构建，每个节点最多连边为3
- 插入流程
  - 例如第5次构造D点的流程
  - 随机一个节点，比如A，保存下与A的距离，然后沿着A的边遍历，E点最近，连边。然后再重新寻找，不能与之前重复，直到添加完3条边
- 查找流程
  - 查找流程包含在了插入流程中，一样的方式，只是不需要构建边，直接返回结果



elastic

HNSW算法

- HNSW是NSW的分层优化，借鉴了skiplist算法的思想，提升查询性能
- 开始先从稀疏的图上查找，逐渐深入到底层的图
- 查找的时候需要把索引全部load到内存中



elastic

算法总结

	LSH插件	IVFPQ插件	HNSW插件
概述 (具体源码地址见最后)	在创建index时传入抽样数据，计算出hash函数。写入时增加hash函数字段。 召回用minimum_should_match控制计算量	在创建索引时传入聚类点和码本，写入数据就增加聚类点和编码字段，召回先召回编码后距离近的再精确计算	扩展docvalue，在每次生成segment前，获取docvalue里的原始值，并调用开源hnsw库生成索引
优点	实现简单，性能较高 无需借助其他lib库 无需考虑内存	性能较高 召回率高 >90% 无需考虑内存	查询性能最高 召回率最高 >95%
缺点	1.召回率较其他两种算法较差，大概在85%左右 2.召回率受初始抽样数据影响 3.ES的metadata很大	1.需要提前使用faiss等工具预训练 2. ES的metadata很大	1.在构建的时候，segment合并操作会消耗巨大的cpu 2.多segment下查询性能会变差 3.全内存

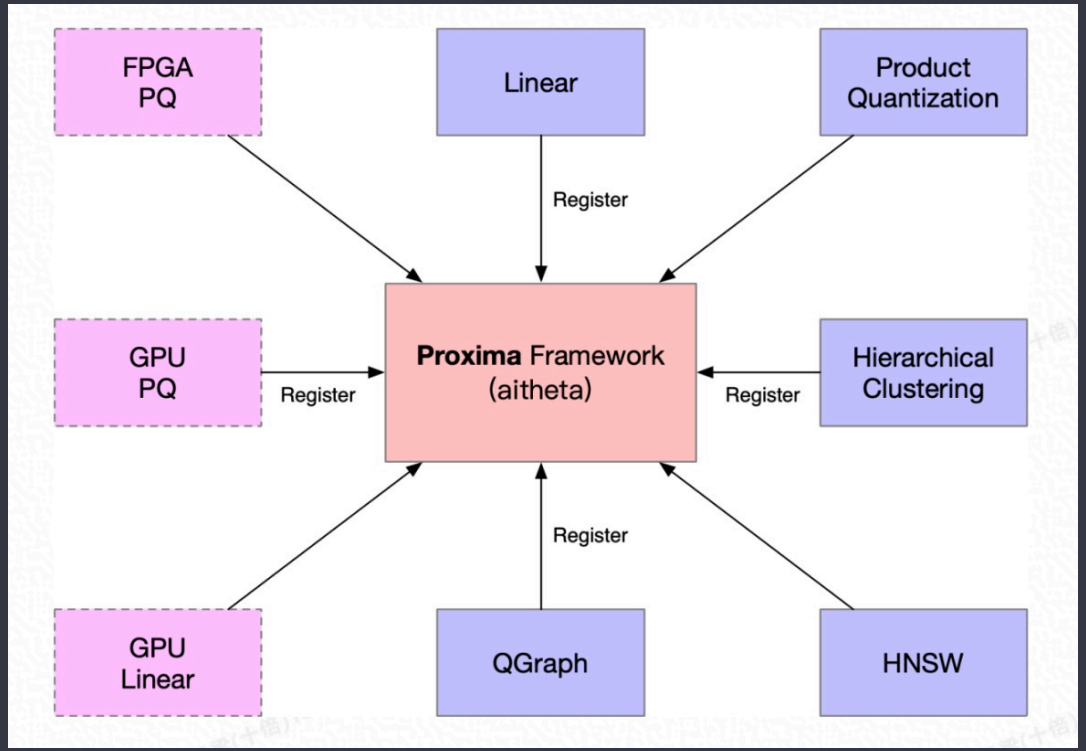
ZSearch ANN插件

elastic

ZSearch ANN的方案

Proxima向量检索引擎介绍

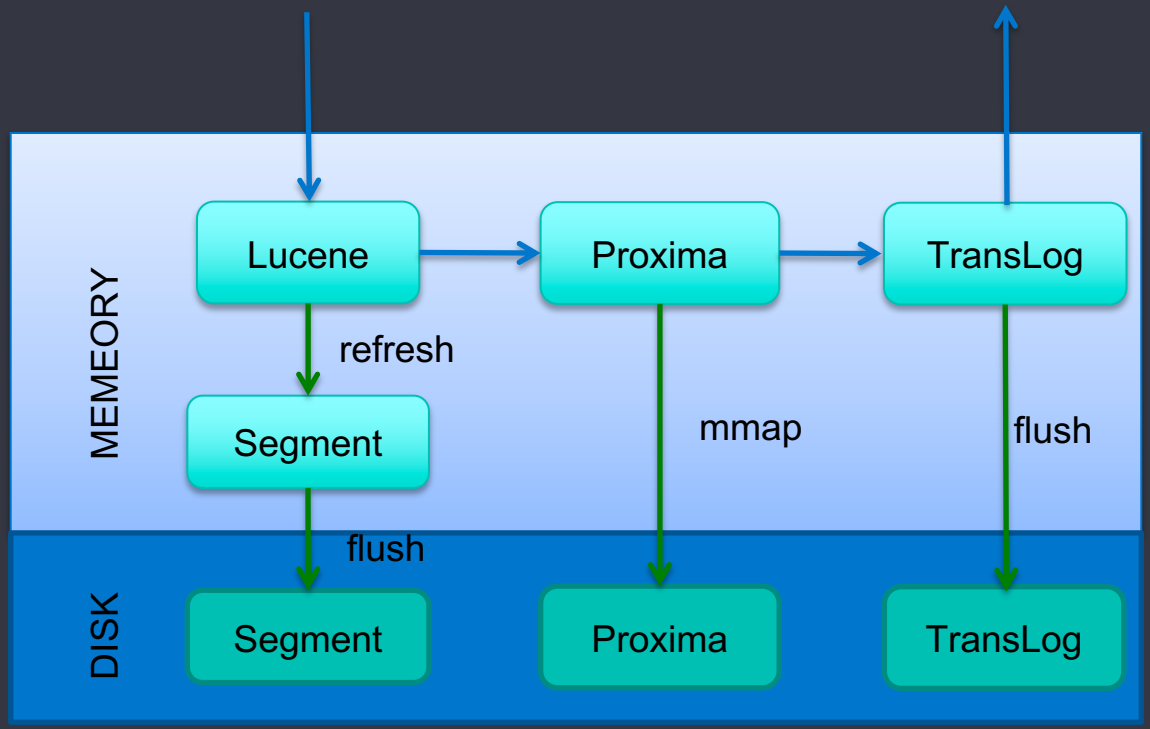
- 阿里巴巴Proxima向量检索引擎
  - 支持多种向量检索算法
  - 统一的方法和架构，方便使用方适配
  - 支持异构计算，GPU



elastic

ZSearch ANN的方案

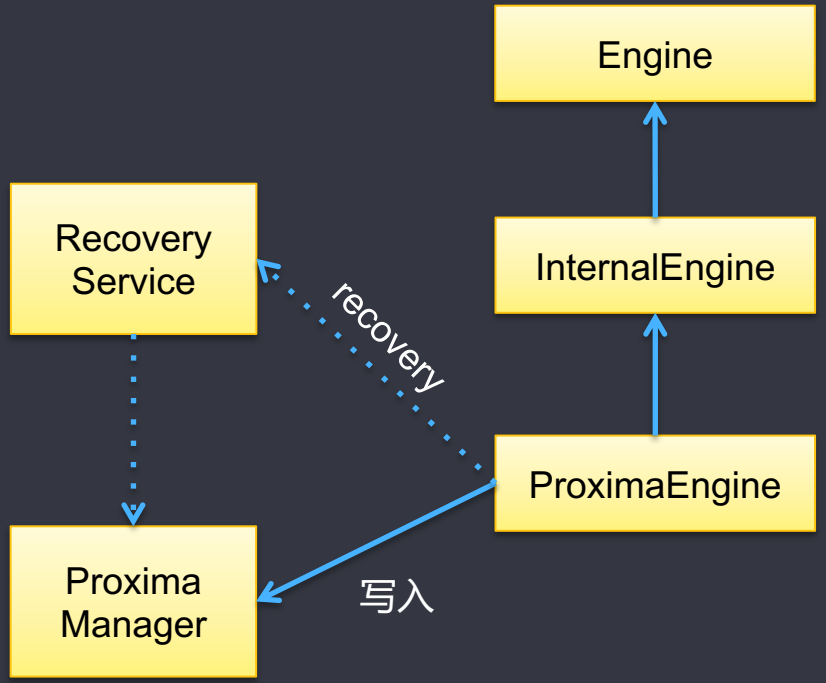
数据写入



elastic

ZSearch ANN的方案

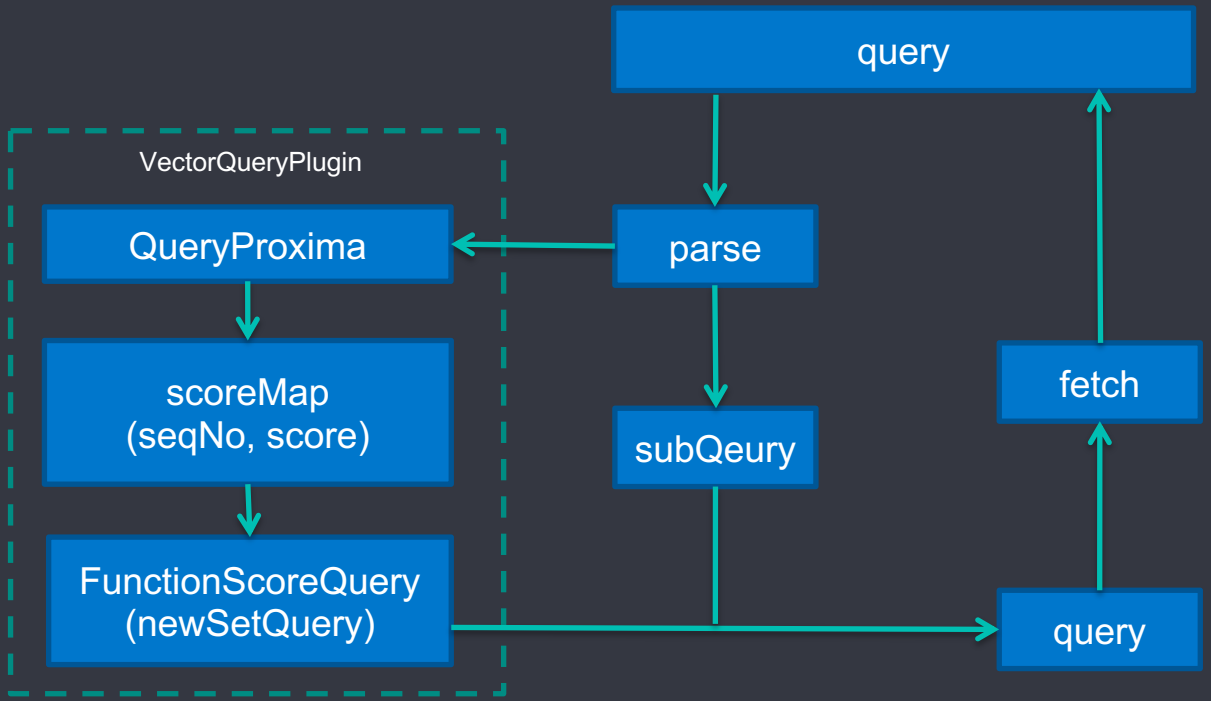
数据写入



elastic

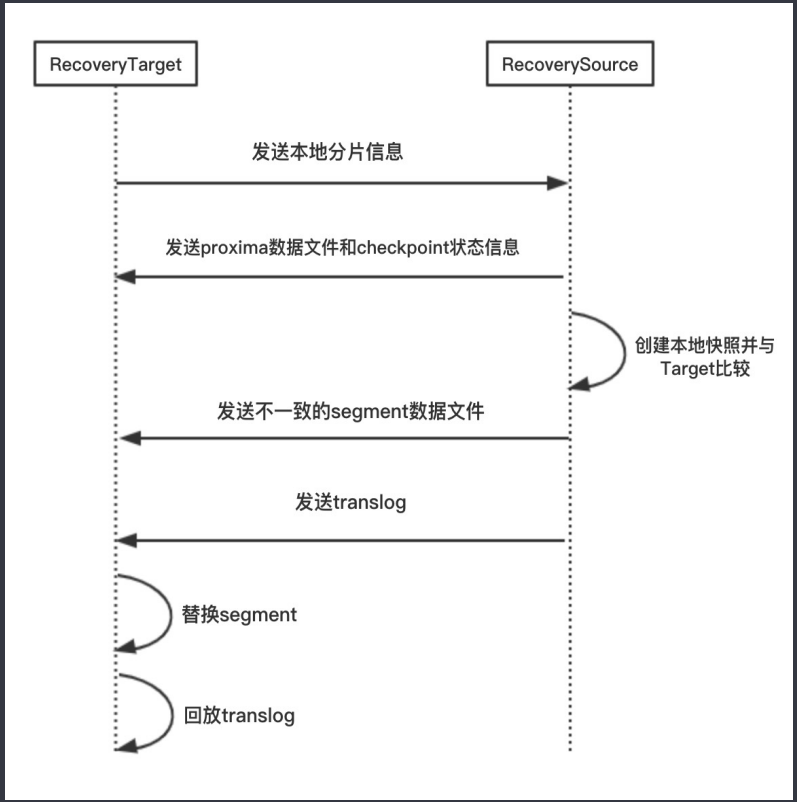
ZSearch ANN的方案

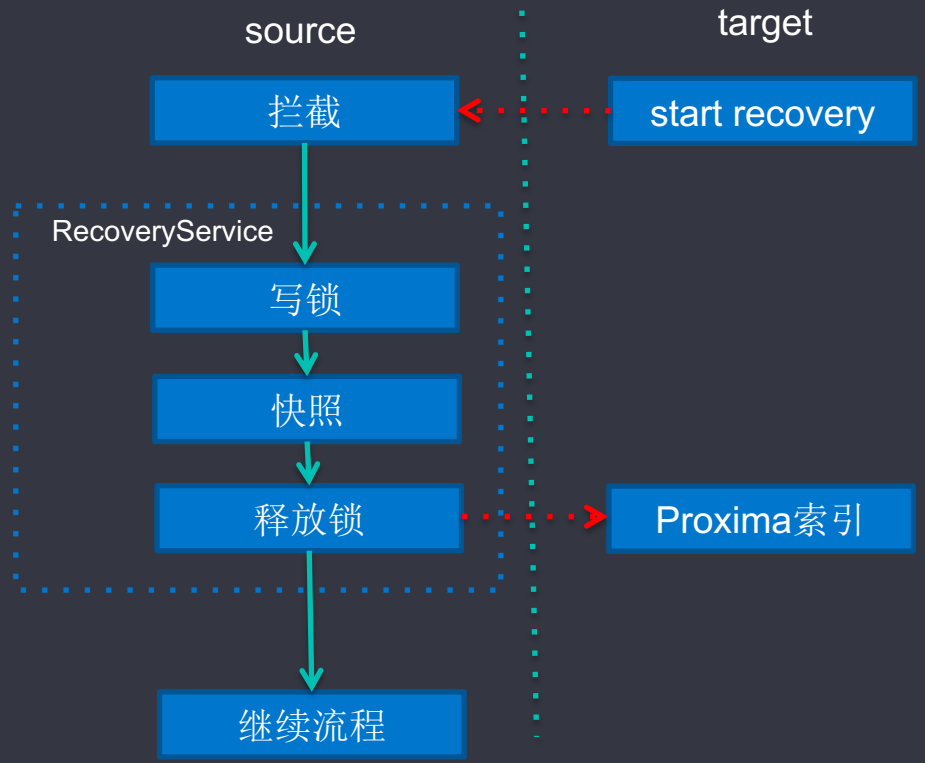
数据查询



elastic

插件Failover流程





elastic

性能对比

sift-128-euclidean 100w数据集(https://github.com/erikbern/ann-benchmarks)

	HNSW插件	ZSearch-HNSW插件
数据写入 （单线程1000个bulk写入）	1.初始写入 5min，25个segment，最大CPU 300% 2.合并成1个segment 5min，最大CPU 700%(本地最大)	构建索引 15min，因为单线程最大CPU 100%
查询	1. Top 10，召回率98% 2.rt 20ms，合并成1个segment后，5ms	1. Top 10，召回率98% 2. rt 6ms
优点	兼容failover	CPU消耗少，无额外存储
缺点	CPU消耗大，查询性能跟segment有关	主副分片全挂的情况下会有少量数据重复

6. 总结

ES参数配置最佳实践

- 100w 256维向量占用空间，大概是0.95GB，比较大
  - 所以更大的堆外内存分配给pagecache
  - 例如8C32G的机器，JVM设置8GB，其他24GB留给系统和pagecache
- 设置max_concurrent_shard_requests
  - 6.x默认为节点数*5，如果单节点CPU多，可以设置更大的shards，并且调大该参数
- BF算法使用支持AVX2的CPU，基本上阿里云的ECS都支持

总结

- KNN适合场景
  - 数据量小(单分片100w以下)
  - 先过滤其他条件，只剩少量数据，再向量召回的场景
  - 召回率100%
- ANN场景
  - 数据量大(千万级以上)
  - 先向量过滤再其他过滤
  - 召回率不需要100%
  - LSH算法： 召回率性能要求不高，少量增删
  - IVFPQ算法：召回率性能要求高，数据量大(千万级)，少量增删，需要提前构建
  - HNSW算法： 召回率性能要求搞，数据量适中(千万以下)，索引全存内存，内存够用

未来规划

- Proxima支持覆盖写，保证最终一致性
- 增加更多的向量检索算法适配不同的业务场景
- 将模型转化成向量的流程下沉到ZSearch插件平台，减少网络消耗

附件

- fast-cosine插件：https://github.com/StaySense/fast-cosine-similarity
- LSH插件 ：https://github.com/alexklibisz/elastik-nearest-neighbors
- IVFPQ插件：https://github.com/rixwew/elasticsearch-approximate-nearest-neighbor
- HNSW插件：https://github.com/opendistro-for-elasticsearch/k-NN
- 向量算法概述：https://yongyuan.name/blog/vector-ann-search.html
- HNSW算法：https://blog.csdn.net/u011233351/article/details/85116719
- ANN性能测试框架 ：https://github.com/erikbern/ann-benchmarks