汽车之家基于 Milvus 的向量检索平台实践

作者：Zilliz

2022-12-20 北京

本文字数：3908 字 阅读时间约为 12 分钟

#01

背景

随着计算机技术及机器学习技术的发展，特征向量作为一种多媒体数据（文本、语音、图片、视频）的描述方式，逐渐成熟起来，而向量检索（向量相似计算）也逐渐成为一种通用的需求。

向量检索在之家拥有非常广泛的应用场景，如推荐在线业务非明文召回场景，相似视频/图片/音频去重场景等等。截止到 22 年初，业务方部署了 9 个向量检索引擎去检索向量数据。不过，随着向量检索需求增加，许多问题也接踵而来：

资源浪费

每个业务线都会搭建自己的向量检索引擎，资源没有统一管理，会出现不必要的资源浪费。

维护成本

维护向量检索引擎会有一些技术门槛，业务方无法专心于处理业务，且`Vearch`社区不活跃，基本上碰到问题都需要业务方自己去解决或者想办法绕过。

开发成本

为了适配新的召回需求，每次上线新的向量接入需求都需要业务方定制化开发。无法更敏捷地支持在线业务的需求变更。

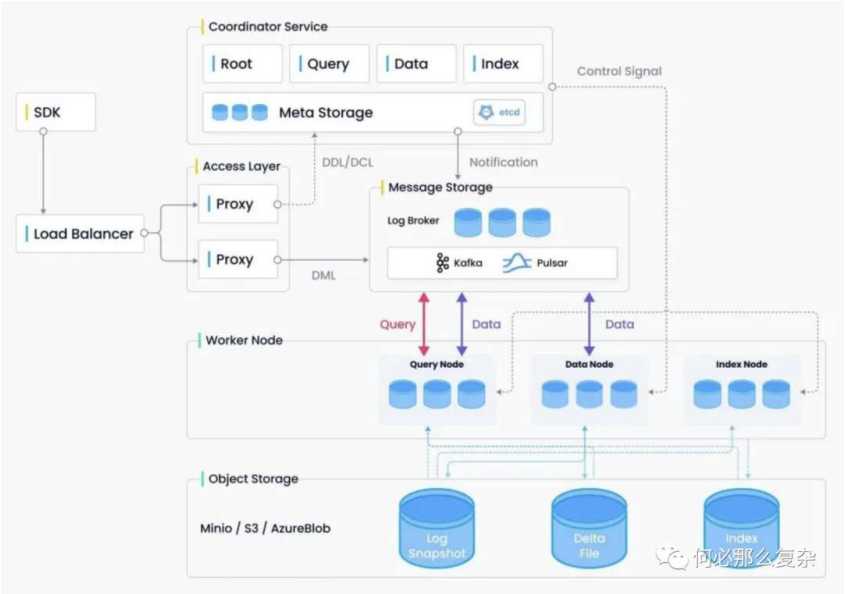
性能

`Vearch`的性能也越来越满足不了在线业务的需求，导致在线召回项目有较高的超时率，影响线上实验及模型效果。

#02

技术选型

非常不错的表现。



我们来看下 Milvus 2.x 版本的架构实现特点：

微服务

Milvus 将服务拆成多个角色，每个角色职责划分相对独立，这样每个角色的源码阅读起来非常容易。简单介绍下 Milvus 的角色职责：

`ETCD`:负责存储元数据

对象存储:负责存储向量数据

`Proxy`: Milvus 统一的访问层

`DataNode`/`DataCoord`: 负责向量的写入

`IndexNode`/`IndexCoord`:负责向量索引的构建

`QueryNode`/`QueryCoord`: 负责向量的查询

`RootCoord`: 负责处理 DDL 去协调其他 Coord，全局时间分发，维护当前元数据快照

其中`IndexNode`/`QueryNode`/`DataNode` 这些角色是实际工作的 Woker 节点，`IndexCoord`/`QueryCoord`/`DataCoord` 是负责协调 Woker 节点，是将任务 handoff 给其他角色的节点。

支持云原生

Milvus 服务本身是没有状态的，数据存储在对象存储，元数据会存放在 ETCD。原生支持 K8s 部署集群部署，我们可以根据集群或者个别角色的负载去动态扩缩资源。

向量操作读/写/建索引之间进程级别隔离

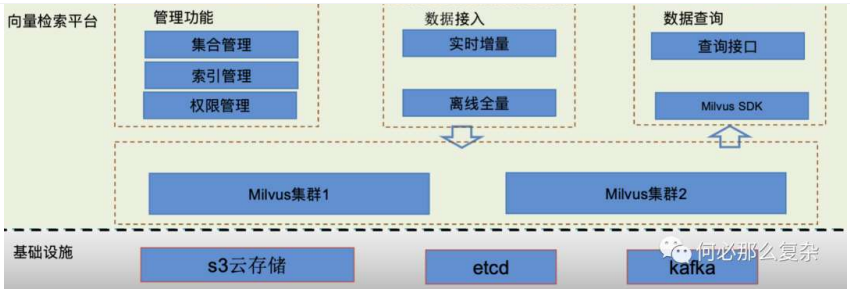
如上图，向量读/写/建索引都是通过不同的节点完成，这样操作之间都是通过进程之间隔离，不会抢占资源，相互影响。

此外，Milvus 还可以在查询的时候指定不同的一致性级别。在真实的业务场景中，一致性要求越强，查询对应的响应时间也会变长。用户可以根据自己的需求选择不同的一致性级别。**除了 Milvus 出色的架构能力之外，Milvus 非常活跃的社区及其背后优秀的商业公司 Zilliz 也是我们选择 Milvus 的重要原因。**

#03

向量检索平台介绍

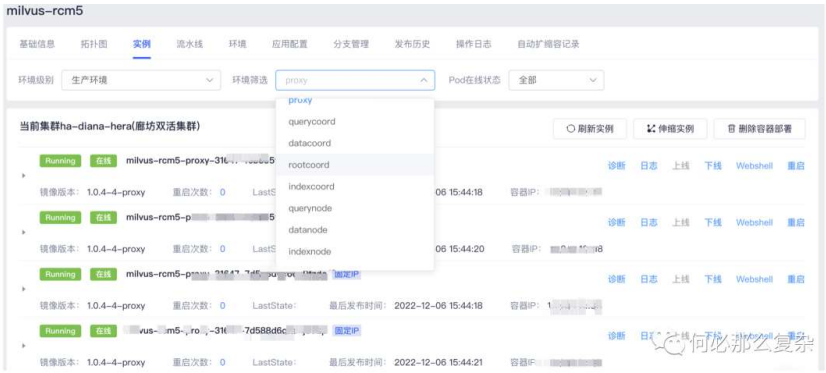
目前向量检索平台已经日益成熟，支持了 30 多个离/在线需求。在性能，稳定性，资源节约方面都非常不错的提升。



基础设施

部署

我们通过改造 Milvus 原生的部署方式，将 Milvus 集群部署在之家云 K8s 集群中。因为 Milvus 服务本身是无状态的，在 K8s 上，我们可以根据业务的查询写入需求，灵活地扩缩 Milvus 的服务节点，节约服务器成本。



监控/日志

我们将监控/历史日志采集到 Prometheus/ES ，可以非常方便地通过监控日志定位问题，配置报警。

索引的选择

IVF-FLAT 倒排索引

IVF-FLAT 适合数据量较小的集合，在我们的测试场景中，十万级别的数据使用 IVF-FLAT 索引可以得到很好的查询性能。通过调整构建索引参数 nlist 和查询参数 nprobe，在召回准确率和召回性能之间找到适合业务需求的平衡点。

HNSW 图索引

副本、分片的选择

不同的分片数和副本数，对于高并发下的查询性能有显著影响：

对于小数据量集合（十万数量级）推荐使用一个分片即可，可以通过扩展副本数，提高集合的并发能力，从而提高查询 QPS。如将副本数设置与 QueryNode 节点数量一致，可以充分利用每一个 QueryNode。

对于千万级别的大集合比较容易受到资源限制（内存占用），一般无法设置太多副本。可以先通过 Milvus 官方提供的计算工具（https://milvus.io/tools/sizing/）评估大概会占用的内存，再根据 quernNode 节点的实际情况确定分片数量。

容量规划

下图是我们的压测报告，经过一系列的压测我们得出结论：

（数据量：109780；索引：Ivf-flat；1 分片，10 副本，查询参数：nlist 1024，size 200，noprob 50）

1. 每个 querynode（12 核 16 GB）可支持 500 QPS
2. 每个 Proxy（4 核 8 G）可支持 1200 QPS
3. querynode 与 Proxy 比例建议为 2：1
4. 向量平台每个实例可以支持 3500 QPS
5. 小数据量（< 10 万） 场景下，建议 1 分片多副本。分片数过多会导致性能下降
6. 以上场景下，cpu 消耗均低于 60%，内存占用低于 10% 且没有持续增长的趋势

Milvus压测报告

	值	说明
集合数据量	109780	32维
索引	IVF-FLAT	nlist 1360
查询nprob	500	压测50-500，对性能没有明显影响
查询size	500	测试200-1000，查询条数变化对性能没有明显影响。
qps	3000+	
平响	9ms	
tp90	21ms	proxy 15毫秒
tp99	<50ms	
准确率	98%	可以继续提升

#04

平台对 Milvus 的一些优化

之前提到我们选择 Milvus 的一个重要原因就是 Milvus 非常活跃的社区和背后优秀的商业公司，我们反馈的问题都会有社区的运营跟进。在我们对 Milvus 不熟悉时，社区的同学会专门来之家为我们解惑，协助我们上线。**在享受社区的开源红利的同时，我们在熟悉 Milvus 的过程中还会向社区贡献我们对 Milvus 的改进：**

集成 kafka 的时候指定 kafka 配置

https://github.com/milvus-io/milvus/pull/18742

修复 QueryNode metric 相关问题

https://github.com/milvus-io/milvus/pull/18367

https://github.com/milvus-io/milvus/pull/19479

https://github.com/milvus-io/milvus/pull/20426

https://github.com/milvus-io/milvus/pull/20427

修复加载配置时未正确释放锁

https://github.com/milvus-io/milvus/pull/18773

优化 RootCoord show collection 操作延迟

https://github.com/milvus-io/milvus/pull/20124

修复小数据量的集合不能及时加载索引

https://github.com/milvus-io/birdwatcher/pull/55

此外在之家内部还有些通用性低或者抽象得不太完善的实现，后面完善后会和社区讨论是否可以贡献给社区，下面分享两处查询性能方面的优化。

在我们的测试场景下，在各组件 CPU 使用率正常的情况下，查询索引的性能比较稳定，但经过各组件之间的 RPC 通信后，TP99 就会变得比较高，基本在 100 ms 以上，在网络环境差的场景影响尤为明显。

以下两处优化本质都是减少 RPC 请求次数，避免因网络抖动导致 TP99 飙高。

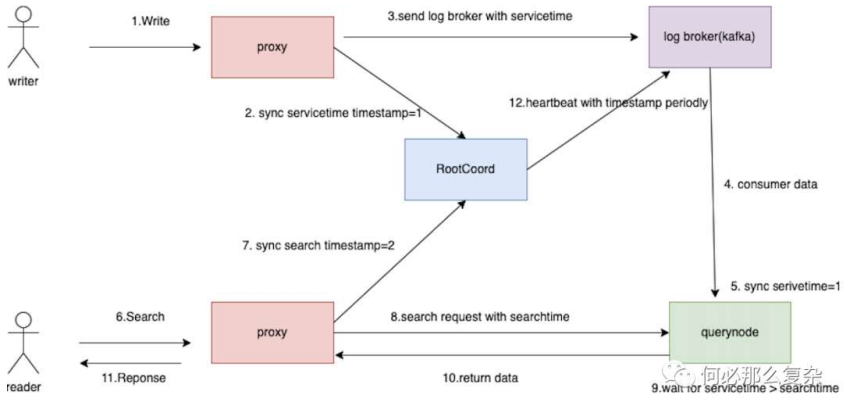
1.弱一致性查询不去访问 RootCoord 获取时间戳

背景:

Milvus 写入的数据和 RoodCoord 发送的心跳都会带有 RooCoord 的时间戳，会写到 logbroker 里面 ，QueryNode 会消费数据里的时间戳更新 servicetime t1。

在做强一致性查询时也会向 RootCoord 查询时间戳 t2。

QueryNode 只有在 t2>t1 后，才能确保要查询的数据都到齐了之后将查询到的数据返回给 Proxy。



优化:

目前在弱一致性的场景也会向 RootCoord 同步时间，这个时间并没有应用在 QueryNode，仅仅是用来做 msgID，在日志里追踪查询行为。因此在弱一致性场景我们在 Proxy 本地做了时间戳分发，不会再去请求 RootCoord。这样可以减少一次 RPC 请求，避免网络原因导致查询 tp99 较高的问题。

2.QueryCoord 分配 Segment 时优先分配给这个副本的 shardleader 节点

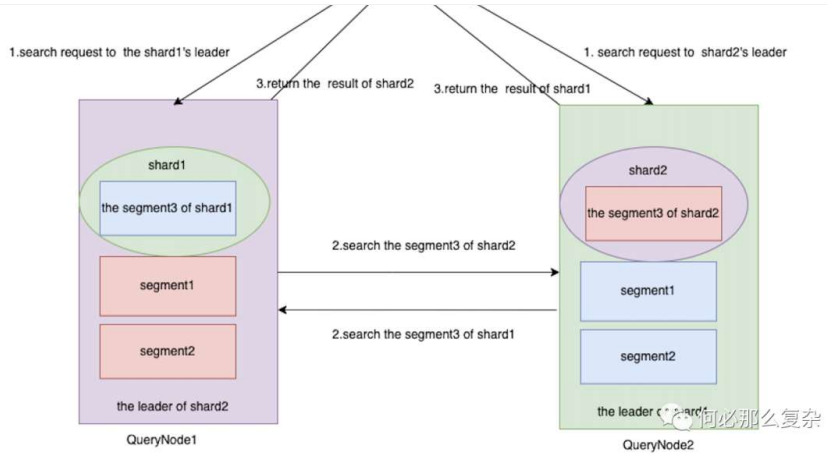
背景:

我们接着介绍下背景：如图我们看到的是一个集合某一个副本下两个分片的查询场景。

其中 Proxy 会分别去 QueryNode 请求两个分片的 leader

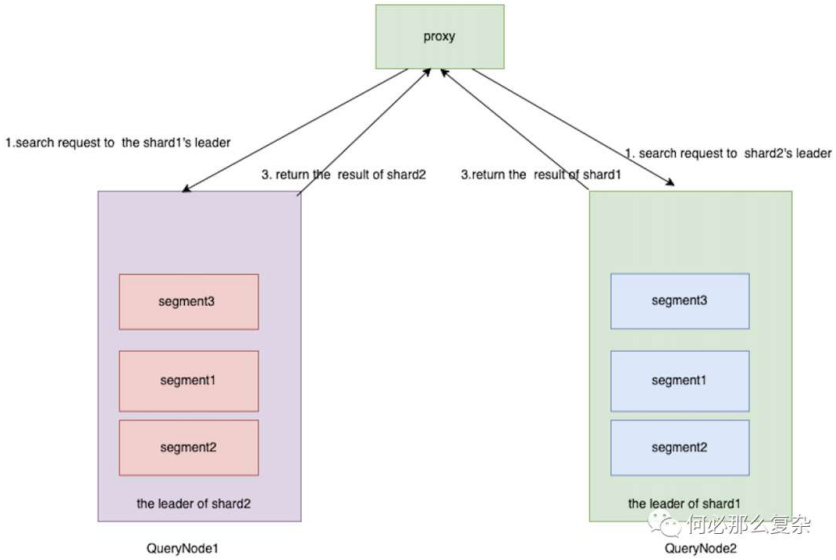
由于分配 Segment 是基于 QueryNode 持有向量的行数做均衡分配的，每个 shard 的 Segment 可能会被分配到不同的 QueryNode 上

所以 shard leader 需要去其他 QueryNode Search Segment，会额外多一次 RPC 的开销



优化:

针对特别大的数据量集合的场景，Segment 在 QueryNode 之间负载均衡是非常有必要的。我们存在一些在线业务场景数据量很小，只会占据很少的资源。但是对查询延迟有极高的要求。**因此我们就在 QueryCoord 分配 Segment 时，关闭了 rebalance checker，将 Segment 分配到 ShardLeader 所在的机器。这样就不会有 QueryNode 之间的查询了。**



#05

应用案例

推荐非明文召回:

非明文召回服务良好地支撑了用户长短期兴趣召回模型、双塔召回模型、冷启动召回模型等 23 个算法向量模型数据生产及 24 路非明文召回。

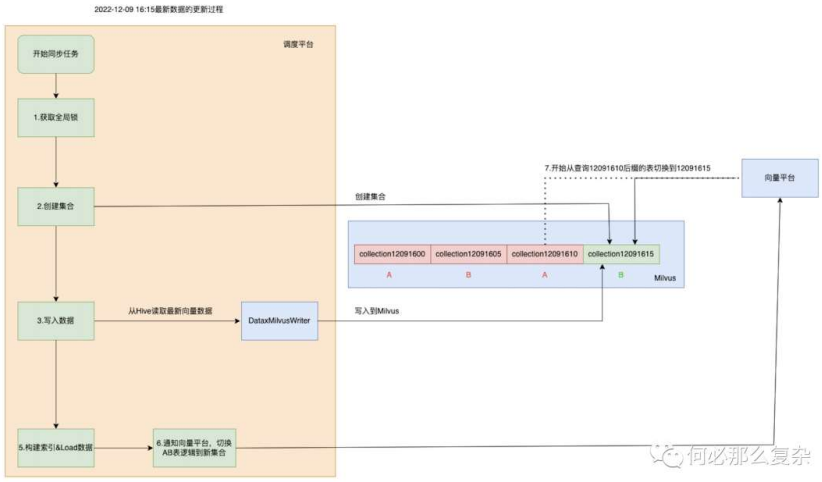
主要流程:

3. 在北斗系统定制召回及策略融合策略，就可以上线召回模型

最后说一下刚才说到的 AB 表功能，**向量平台平台目前提供两种数据更新方式，增量更新和用过 AB 表的方式全量更新。**

增量更新方式适用于业务数据不断增长，必须以全量数据作为基础数据为业务提供向量检索。比如，图片、文本、视频、音频去重业务。全量更新适用于算法小数据实验，可以快速看到实验效果，每次实验数据数据会自动隔离，不会相互影响。**全量更新可以精确到天、小时、分钟级别，可以满足算法不同需求。**

下图是 AB 表的流程，每 5 分钟调度任务会全量同步 Hive 中模型加工好的向量数据，待所有数据写完，索引构建成功，就会通知向量平台从查询旧表（图中 collection12091610）切换到新表（collection12091815）。



收益:

性能方面

召回超时率较 Vearch 下降了 3-7 倍

平响较 Vearch 下降了 55%

简化向量接入流程

接入全面配置化，取代了 Vearch 产品需要代码开发的发布方式，上线效率提升至少 1 倍

#06

后续规划

1. Milvus 的 Upsert 功能未 Release，目前有部分数据量特别大的业务会强依赖这个功能。
2. 部分去重场景需要强一致性查询，但是目前强一致的查询延迟较高，需要和社区一起探讨优化思路。

发布于: 2022-12-20 | 阅读数: 152

数据库 向量检索 Milvus

Zilliz

Data Infrastructure for AI Made Easy 2021-10-09 加入

还未添加个人简介

+关注

评论

快抢沙发！虚位以待

发布

暂无评论