Smart-LSM-Tree 的结构持久化报告

523031910852 朱子墨

2025 年 5 月 8 日

1 背景介绍

LSM-Tree（Log-Structured Merge-Tree）是一种适用于高吞吐量写入的存储结构，由 Patrick O’Neil 等人于 1996 年在论文中首次提出。主要用于数据库（如 LevelDB、RocksDB）和键值存储系统（如 Cassandra）。它通过批量写入和分层合并（Compaction）来优化磁盘 I/O，从而提高写性能。

现代存储系统面临非结构化数据处理的重大挑战。LSM-Tree 作为高性能存储引擎，通过顺序写优化和层级合并机制，在键值存储领域占据重要地位。然而传统 LSM-tree 仅支持精确键值查询，难以应对语义相似性搜索需求。

在前面的阶段中，我使用了 HNSW（Hierarchical Navigable Small World）算法来提高搜索效率。在本阶段，我完成了 embedding 向量和 HNSW 算法搜索结构的持久化。

2 测试

2.1 实验设置

由于在 windows 系统上进行的环境配置遇到了很多困难，因此最终在 VMware Workstation 的 linux 系统虚拟机上完成实验。实验环境基本参数如下：

项目	参数
内存大小	13.1GB
硬盘大小	50.0GB
处理器数量	2
操作系统	Ubuntu 22.04.1 LTS
主要计算支持	CPU

表 1: 实验环境基本参数

在本阶段，持久化的完善过程包括以下几个步骤：

第一步：实现 vector 存储到磁盘和 vector 修改、删除的过程。在这个过程中，我增加了一个 bufferMap 用于缓冲，embedding 转化得到的向量会先存储到 bufferMap 中，然后再合并到 vectorMap 中。修改过程中，利用 std::unordered_map 的特性，利用插入操作的接口实现覆盖插入；删除操作中，设置被删除的键值对的向量值为 vector<float(dim,std::numeric_limits<float>::max())，然后替换 vectorMap 中的原有值；当 KVStore 析构、发生 compaction 时自动合并内存中的临时存储，插入到磁盘文件里。

第二步：实现 vector 的读取。利用流的方式从后往前读取数据、放入 vectorMap；对于已经删除或者读取过的数据不再读取以前的版本；按照引导增加测试文件，修改 CMakeLists，进行 Vector_Persistent_Test 测试，发现已经成功实现了加载。

第三步：实现 HNSW 的删除和修改。删除时，在该节点上加注 deleted 标签，通过修改 validness 值实现；修改时先删除原来的节点，再插入新的节点。由于这样会带来重复的 nodeKey，因此我增加了新的节点对象：nodeID 用于标注每个节点。nodeID 具有独特性，在后续的存储中更容易管理。增加测试文件，修改 CMakeLists，进行 HNSW_Delete_Test 测试，发现已经成功实现了删除。

第四步：实现 HNSW 算法查询结构的持久化和读取。存储时, 先记录 global 信息，再分节点记录各个节点的信息和邻接表。邻接表进行分层存储，每个文件存储对应层级与该节点连接的节点 nodeID；在读取时，先读取 global 信息，再读取每个节点的 head 信息，读取完所有节点的 head 信息后再进行每个节点的邻接表构建。增加测试文件，修改 CMakeLists 文件，先进行较为宽松的测试，再进行较为严格的测试。

2.2 预期结果

在测试中，由于 HNSW 算法会牺牲一定的正确性，因此可能在使用到 search_knn_hnsw 函数接口的 hnsw 持久化测试中出现一定的失败样例，但只需要保证正确率较高即可。

2.3 实验结果与分析

根据我从互联网上的资料得到，HNSW 算法参数设置和问题规模具有一定联系。出于平衡正确率和时间性能的考虑，我根据网络资料查找到的经验公式，经过初步的实验后把参数设定为 M = 15，M_max = 18，m_L = 10，efConstruction = 75。

先进行了 E2E 测试，发现正确率大约为 77.8%(作为对照，不使用 search_knn_hnsw 的直接索引正确率为 85%)。然后进行向量持久化测试和 HNSW 删除测试，均通过测试；对于 HNSW 持久化测试，先使用较为宽松的测试，采用不同大小的测试集多次测试，发现正确率大约 94% 左右。再次进行更为严格的测试，发现删除操作的检查和重新插入操作的检查均可以完全通过，而替换操作的检测中通过率为 96.88%，这基本符合正确率的要求。

3 结论

通过设置删除标记的方法可以实现 hnsw 算法的有效删除和替换操作，通过有效的文件存储结构设计可以很好地实现 hnsw 算法索引结构的持久化。

4 致谢

在完成此实验过程中，ChatGPT 和 deepseek 等大模型给了我很多帮助。同时，知乎专栏的介绍文章、助教和多位同学也给予我了很多指导和帮助。

5 其他和建议

在实验过程中，我遇到了一些挑战和困难：

1. 前期的设计中存在一定冗余，这可以使得存储的可靠性更高，但是在扩展功能、调试程序时会更加麻烦。为了不对原有的设计进行大规模的修改，我不得不对于文件中的推荐存储方案进行了修改。

2. 测试的时间较长，这在下一个阶段并行化之后可以得到缓解。