Smart LSM-tree with Persistence 报告

陈梓萌 523031910121

2025 年 5 月 2 日

1 背景介绍

为了避免在此阶段以及之后的阶段中频繁地进行 embedding 向量操作，所以我首先实现了将 embedding 的向量保存到磁盘的操作并且可以从磁盘中读取 embedding 的向量。其次，使得 LSM 支持 HNSW 索引结构的删除和修改操作。最后，我将 HNSW 结构持久化，主要是将构建好的索引保存到磁盘，以便在后续的查询中复用，避免每次启动时都重新构建索引。

2 测试

2.1 实验设置

2.1.1 实验平台（编译环境）

- C 编译器：GNU GCC 13.3.0
- C++ 编译器：GNU G++ 14.2.0
- CUDA 编译器：NVIDIA NVCC 12.6

2.1.2 开发环境

- 操作系统：Windows 10（版本 10.0.26100）
- 子系统：WSL2（Ubuntu 24.04 LTS）
- CUDA 工具包：CUDA Toolkit 12.6（支持 GPU 加速开发）
- IDE：Visual Studio Code（通过 Remote - WSL 插件远程开发）

2.1.3 测试量

分别测试了对于 embedding 向量的持久化以及对于 HNSW 结构的持久化。

1. 对于 embedding 向量的持久化：运行测试用例，测试会插入一些数据，然后保存到磁盘，然后重启数据库，重新从磁盘加载数据，然后查询数据。对于 HNSW 结构的持久化
2. 对于 HNSW 结构的删除操作：检查 lazy delete 是否正确实现。
3. 对于 HNSW 结构的持久化:
   (a) 测试 1：插入一些数据，然后删除一些数据，如果仍然可以查询到被删除的数据，则测试失败。
   (b) 测试 2：插入一些数据，然后保存到磁盘，然后重启数据库，重新从磁盘加载数据，然后查询数据。并且测试 accept rate。
4. 对于 HNSW 和 embedding 的插入与删除的持久化测试：插入一些数据，然后删除一些数据，然后再插入一些数据，进行查询，如果仍查询到被删除的旧数据，则测试失败。并且测试 accept rate。

此外，我修改了测试集，具体表现为在各测试阶段的 Phase 2 均加入了持久化加载操作：`store.load_embedding_from_disk("data/");` 并且修改了相对路径，能够正确运行，读取和写入数据。

2.2 预期结果

1. 实现了持久化操作后，任何测试都可以正确的将数据保存到磁盘并且重新从磁盘加载数据，成功查询到数据。

2.3 实验结果与分析

均通过测试

1. HNSW 结构持久化的 accept rate: 0.59375。
2. 插入与删除操作的持久化测试的 accept rate: 0.90625。

3 结论

本实验通过实现数据持久化优化机制，显著提升了程序在异常中断情况下的健壮性和数据完整性。实验结果表明：

1. 容错能力提升：系统在意外终止后能够完整恢复最近操作状态。
2. 实现了崩溃一致性 (crash consistency) 的 ACID 特性。

4 致谢

1. 感谢孙昕玥同学，我们共同讨论理解了 README 所给出的项目指导，以及一些可能会导致 bug 的实现思路
2. 感谢张同学给出的进一步测试删除和修改的测试集。让我发现一个潜在 bug，这个是在原本测试集当中没有测试出来的。