基于 LSM-TREE 的键值存储的优化（Smart-LSM）报告

523031910852 朱子墨

2025 年 4 月 5 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。

LSM-Tree 尤其适合写密集型场景（如日志存储、物联网设备数据、时序数据库），其设计哲学体现了” 写优化” 与” 读优化” 的权衡，成为大数据时代高性能存储系统的基石之一。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。

本阶段将基于 LSM-tree 架构，通过集成语义嵌入模型（Embedding Model）与近似最近邻搜索（ANN）算法，赋予存储引擎语义检索能力。将现实中的大模型（如实验中所使用的 Embedding 模型）引入 LSM-Tree 结构，从而实现“语义最近查询”，查找与目标字符串词义最接近的 n 个字符串。

在 Embedding 模型中，字符串可以转化成一定维数的向量，即向量化；通过比较目的字符串与存储结构中的所有字符串的向量余弦相似度距离，依次比对找到最近的 n 个字符串。其中，余弦相似度距离满足：

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1}$$

其中，A 和 B 分别是两个字符串通过 Embedding Model 转化得到的向量；$A \cdot B$ 是向量 A 和 B 的数量积，$||A||$ 和 $||B||$ 是其模长。

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

表 1: 实验环境基本参数

在实验第一阶段的基础上进行代码的修改，考虑到本阶段不需要实现向量的持久性，且考虑到查找效率的因素，因此我使用了 std::unordered_map 存储管理键-向量的对应关系。向量应该存储在内存中，如果在每次查找时进行 Emedding 计算会造成时间性能较差。

在 PUT 操作时，通过 vector 存储对应的字符串和键值；当需要查找距离最近的字符串时，将 vector 中存储的字符串通过 Embedding Model 转化为对应的向量。然后和对应的键值一起存入 std::unordered_map 中。

进行 DEL 操作时，先查找对应的键值对对应的向量是否存储在了 std::unordered_map 中；若不存在则在 vector 中遍历查找删除。

对于查找距离最近、最相似的字符串的算法，由于本阶段并不过分要求时间性能，选择遍历 std::unordered_map 中的键值-向量进行计算查找。最后返回一个向量，其中存储了与目标字符串距离最近的若干个键值对。

在完成了基本的代码结构后，我编写了 timeCompare 文件用于测试各个阶段的用时。

2.2 预期结果

在测试文件中大体分为四个阶段：准备阶段、插入阶段、读取文件阶段和搜索阶段。我预测主要的时间开销在插入阶段和搜索阶段。同时，在 kvstore 中我将搜索阶段又进行了划分：合并字符串阶段，Embedding Model 向量化阶段，储存阶段，遍历搜索阶段。我预测这里的时间开销主要出现在向量化阶段和遍历搜索阶段。

2.3 实验结果与分析

设置参数为 80，进行 80 次循环；每次循环查找 5 个距离目标字符串最近的键值对。通过运行测试，我得到了各个时间段的占用时间，如下表所示：

操作阶段	占用时间 (ms)	占用时间比例
准备阶段	2	几乎忽略不计
插入阶段	1	几乎忽略不计
读取文件阶段	1	几乎忽略不计
搜索阶段	89270	100%

表 2: Smart LSM-Tree 键值存储系统性能测试结果

在测试文件的四个阶段中，可以看到搜索阶段的时间消耗远远大于其他阶段。由于在 PUT 操作中不会进行合并、向量化和储存等操作，且 LSM-Tree 的插入性能很好，因此这里 PUT 操作阶段占用的时间很少；搜索阶段则占用了绝大部分时间。

对搜索阶段再次进行划分，得到的平均时延性能如下：

操作阶段	平均每个循环占用时间 (ms)	占用时间比例
合并字符串阶段	0.01	几乎忽略不计
Embedding Model 向量化阶段	728.93	64.83%
插入存储阶段	0.02	几乎忽略不计
遍历搜索阶段	395.43	35.17%

表 3: 查找距离最近的键值对性能测试结果

可以看出，主要的时间消耗在 Embedding Model 向量化阶段和遍历搜索阶段。这是由于使用了 std::unordered_map 使得插入操作的时间复杂度为 O(1)。而合并操作的时间开销本身较小。

在测试过程中，发现第一次执行搜索操作时，Embedding Model 向量化操作的时间开销远远大于后续的操作；这是由于第一次进行搜索时所有的键值对均未进行向量化操作，需要花费大量时间进行操作；而后续的 Embedding Model 向量化操作只需要对于目标字符串本身进行向量化操作即可，其他字符串均可从内存中读取。若排除第一次操作，则实际上 Embedding Model 向量化操作在每个循环的平均时延为 141.56 毫秒，低于遍历搜索阶段。

相比较除第一次以外的向量化操作，遍历搜索阶段的时间开销存在较大方差，这是由于在排序过程中使用了 std::sort，其时间性能受到数据分布的影响。而除第一次以外的向量化操作，Embedding Model 向量化的操作时间性能较为平均。

3 结论

在 Smart LSM-Tree 中，主要的时间性能开销来源于 Embedding Model 的向量化过程（即将合并完成的字符串利用 Embedding Model 转化为向量的过程）和遍历搜索查找距离最近的若干个键值对的过程。在下一阶段这里存在优化空间。而且在不利条件下，若存储的数据量很大时，遍历搜索的时间开销将非常巨大。

PUT 操作占用的时间远小于前两者，甚至可以忽略不计，这体现了 LSM-Tree 写入性能良好的特点。

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也给予我了理论原理上的指导。

5 其他和建议

在实验过程中，我遇到了一些挑战和困难：

1. 实验环境的配置中遇到了很多困难。
2. E2E_test 中存在一些无法通过的样例，而且调节参数不同得到的正确率也存在差异。
3. 虚拟机的运行速度较慢，测试时花费的时间更多。

建议：

1. 同学和助教共同完善答疑解惑的微信文档，互帮互助解决在环境配置等方面的问题。
2. 习题课对已经完成的 lab 或 project 进行讲解，加深理解。