SMART LSM 键值存储报告

523031910121

2025 年 4 月 4 日

1 背景介绍

现代存储系统面临非结构化数据处理的重大挑战。LSM-tree（Log-Structured Merge 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。

在电商推荐系统等典型场景中，商品特征常以高维向量形式存在。传统 LSM-tree 能高效存储键值对，但面对" 查找与' 水果' 语义相似的商品" 这类查询时，存在局限：仅支持精确键匹配，无法处理语义相似性。

在这个阶段，主要是为 LSM-tree 添加语义检索功能。

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

2.1.2 开发环境

操作系统：Windows 10 (10.0.26100)
子系统：WSL2 (Ubuntu 24.04)
IDE：Visual Studio Code (通过 WSL 远程连接)

2.2 预期结果

我将测试代码模块分为了以下几部分：

1. PUT 阶段
2. GET 阶段
3. phase1 数据集，调用模型计算语义最相近的 k1 个元素
4. 依次记录了每一次调用 $search_KNN$ 的用时

预测结果：

1. PUT 阶段和 GET 阶段的用时接近，因为数据集的大小并没有涉及到大量的磁盘读写。
2. 总数据集越多，调用模型计算语义最接近的元素的用时也越长。
3. 单次调用的时间应该差距较小。

2.3 实验结果与分析

put KV Elapsed time: 285262519 microseconds.
get KV Elapsed time: 187 microseconds.
$search\_KNN$ PHASE1 Elapsed time: 282783500 microseconds.
由数据发现 PUT 阶段用时远多于 GET 阶段，经过分析发现，PUT 阶段涉及了

1. 数据预处理：在将 KV 对存储到 LSM 树之前，需要对 value 进行 embedding 处理。这一过程计算量较大，是主要的耗时点。
2. LSM 树存储：将处理后的数据存储到 LSM 树中，涉及磁盘写入操作，也会增加一定的耗时。

在 GET 阶段，耗时仅为 187 微秒。这一阶段的耗时较少，主要原因是：

1. 数据量较小：由于数据量较小，GET 操作主要涉及内存中的数据读取，磁盘读写操作较少。
2. 高效索引：LSM 树的索引结构使得数据查询效率较高，进一步减少了 GET 阶段的耗时。而由于数据量较小，不涉及或者只涉及少量的磁盘读写，所以 GET 阶段用时较少。

在 searchKNN 阶段, 耗时也相对较多，主要原因是涉及了：

1. 相似性计算：KNN 算法的核心在于计算查询点与数据集中每个点的相似性（如欧氏距离、余弦相似度等）。这一过程计算量较大，尤其是当数据量较大或维度较高时，相似性计算会成为主要的耗时点。

2. 结果排序与筛选：在计算出相似性后，需要对结果进行排序，以找到最近邻的点。排序操作的时间复杂度较高，尤其是在结果集较大时，这一步骤会显著增加耗时。

3. 数据分布与查询条件：数据的分布特性（如稀疏性、聚类性）以及查询条件的复杂性也会影响搜索效率。某些查询可能需要遍历大量数据点才能找到满足条件的结果，从而增加耗时。

关于 KNN 搜索耗时数据的详细信息，请参见附录中的表格 1。

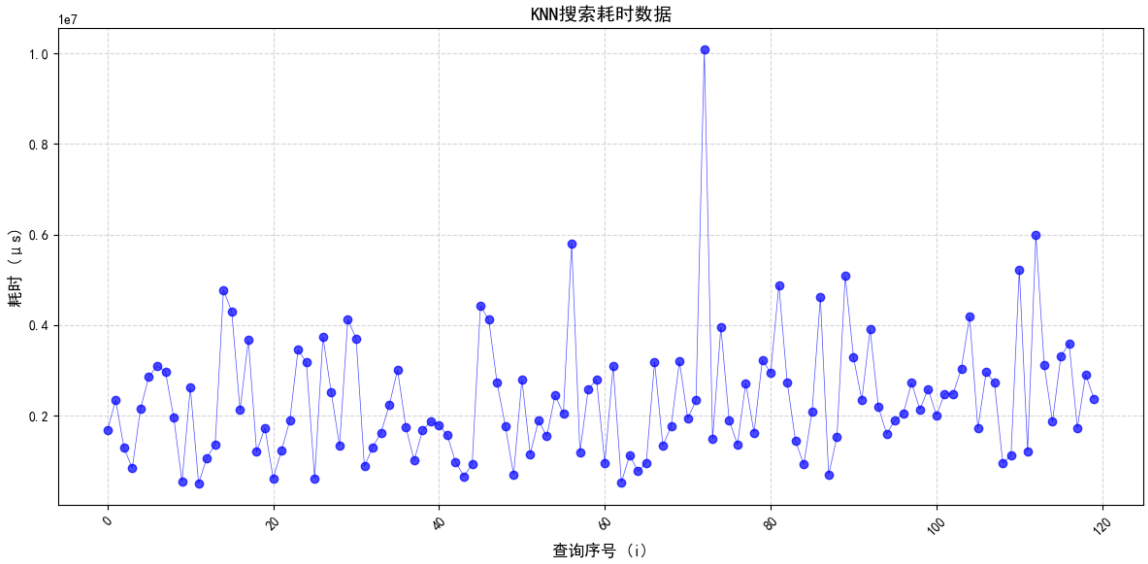

图 1: KNN 搜索耗时

由图 1 可以看出耗时数据在大部分查询序号上保持在较低的范围内（约 1e6 到 4e6 s），但在某些查询序号上出现了明显的峰值。在查询序号 72 附近，耗时达到了约 10,000,000 s，这是一个明显的异常值，远高于其他数据点。这种异常值可能由以下原因导致：

1. 数据集在该查询序号处的复杂性较高。

2. 查询条件或算法实现中存在某些特殊情况。

3. 系统资源在该查询时受到了限制（如 CPU 或内存瓶颈）。

3 结论

通过对 PUT、GET 和 SEARCH KNN 三个阶段的分析，可以看出：

PUT 阶段的耗时主要集中在 embedding 处理上，优化 embedding 算法或预处理数据可以显著减少耗时。

GET 阶段由于数据量较小且主要涉及内存读取，耗时较少，进一步优化空间有限。

SEARCH KNN 阶段的耗时主要集中在相似性计算和结果排序上，通过优化索引结构、使用近似最近邻搜索算法、并行计算和数据预处理等方法，可以有效提高搜索效率，减少耗时。

通过实施上述优化建议，可以进一步提高系统的整体性能，特别是在处理大规模数据集时，显著减少耗时。

4 致谢

感谢在编写 CMakelists.txt 时给出建议和共同沟通的同学。

5 其他和建议

第一次在 project 当中引入现实当中的大模型，在精度上会有所偏差。并且由于模型本身和系统能力不同，会有部分的测试集无法通过，需要进一步完善。

A 附录

表 1: KNN 搜索耗时数据 (i=0-119)

查询序号 (i)	耗时 (s)	查询序号 (i)	耗时 (s)	查询序号 (i)	耗时 (s)
0	1,675,731	40	1,793,873	80	2,940,902
1	2,344,264	41	1,564,272	81	4,879,544
2	1,285,122	42	971,903	82	2,734,978
3	838,217	43	658,848	83	1,440,053
4	2,156,022	44	925,476	84	918,402
5	2,860,404	45	4,426,036	85	2,094,268
6	3,097,214	46	4,117,482	86	4,609,969
7	2,975,189	47	2,724,844	87	693,165
8	1,954,307	48	1,768,482	88	1,523,622
9	544,712	49	692,615	89	5,083,684
10	2,622,680	50	2,801,069	90	3,281,363
11	502,953	51	1,148,878	91	2,345,931
12	1,067,504	52	1,887,886	92	3,902,018
13	1,361,441	53	1,544,087	93	2,188,178
14	4,777,088	54	2,450,062	94	1,596,011
15	4,292,150	55	2,034,412	95	1,899,052
16	2,120,324	56	5,802,270	96	2,043,514
17	3,675,377	57	1,191,653	97	2,723,044
18	1,200,825	58	2,582,078	98	2,134,609
19	1,731,748	59	2,797,294	99	2,585,757
60	945,294	100	2,008,150	100	2,008,150
61	3,087,156	101	2,481,449	101	2,481,449
62	526,391	102	2,476,937	102	2,476,937
63	1,125,994	103	3,035,686	103	3,035,686
64	767,577	104	4,192,632	104	4,192,632
65	956,869	105	1,713,747	105	1,713,747
66	3,189,362	106	2,957,834	106	2,957,834
67	1,336,533	107	2,721,284	107	2,721,284
68	1,759,904	108	949,251	108	949,251
69	3,196,901	109	1,121,249	109	1,121,249
70	1,936,051	110	5,226,141	110	5,226,141
71	2,334,144	111	1,217,901	111	1,217,901
72	10,084,731	112	5,986,519	112	5,986,519
73	1,483,404	113	3,115,788	113	3,115,788
74	3,950,146	114	1,863,420	114	1,863,420
75	1,898,303	115	3,302,707	115	3,302,707
76	1,359,089	116	3,598,633	116	3,598,633
77	2,716,873	117	1,719,992	117	1,719,992
78	1,619,128	118	2,912,613	118	2,912,613
79	3,231,197	119	2,372,563	119	2,372,563