LSM Phase3 报告模板

陈梓萌 523031910121

2025 年 4 月 23 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。$LSM-tree\ Log-StructuredMergeTree$ 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。本阶段将基于上一阶段的工作,实现一个高效的近似最近邻搜索系统。我们将使用 $HNSW\ HierarchicalNavigableSmallWorld$ 算法来提高搜索效率。

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

对总数据量为 120 条,k=3(共 360 条数据) 的查询测试.

1. M (在插入过程中，被插入节点需要与图中其他节点建立的连接数)
2. m_L (来控制节点的层数分布)
3. $efConstruction$ (候选节点集合的数量)
4. $M_{\max}$(每个节点与图中其他节点建立的最大连接数。)
5. T (去除掉 $embedding$ 的运行时间)
6. α 相比于 KNN 算法的正确率 ($\alpha_{\text{HNSW}}/\alpha_{\text{KNN}}$)

2.2 预期结果

1. 鉴于 HNSW 是在 NSW 基础上，借助跳表的思想，设计的数据结构。HNSW 的分层结构（类似跳表）将显著减少搜索时的遍历范围，尤其是在高层级快速定位目标区域。相比 Phase2 的精确搜索时间性能上会有所提升；
2. 由于 HNSW 的近似搜索特性可能导致部分精确最近邻被遗漏，所以正确率会有所下降；
3. 利用了剪枝算法，可以提高时间性能，但是由于当 $efConstruction$ 增大会提升候选集覆盖范围，提高准确率，但搜索时间增加；
4. m_L 增大会加速高层级导航, 可以提升时间性能；
5. 由于需维护多层图结构（连接数限制 $M_{\max}$、动态调整边等），插入操作的耗时可能增加。但文档指出无需持久化向量，内存中缓存嵌入结果可避免重复计算，因此总体影响可控；

2.3 实验结果与分析

表 1: 探究 efConstruction 对于时间性能和正确率的影响

M	$M_{\max}$	efConstruction	m_L	T	α
10	15	22	6	197	303/324
10	15	23	6	201	300/324
10	15	24	6	237	300/324
10	15	25	6	252	318/324
10	15	30	6	323	318/324

表 2: 探究 $M_{\max}$ 对于时间性能和正确率的影响

M	$M_{\max}$	efConstruction	m_L	T	α
10	17	25	6	275	322/324
10	18	25	6	289	324/324
10	20	25	6	276	324/324
10	22	25	6	312	324/324
10	24	25	6	311	324/324

表 3: 探究 M 和 $M_{\max}$ 对于时间性能和正确率的影响

M	$M_{\max}$	efConstruction	m_L	T	α
6	12	25	6	257	315/324
7	14	25	6	263	308/324
8	16	25	6	246	314/324
9	18	25	6	262	321/324
10	20	25	6	276	324/324

表 4: 探究 m_L 对于时间性能和正确率的影响

M	$M_{\max}$	efConstruction	m_L	T	α
8	16	25	5	241	314/324
8	16	25	6	243	314/324
8	16	25	7	245	314/324
8	16	25	8	238	314/324
8	16	25	9	244	314/324
8	16	25	10	242	314/324

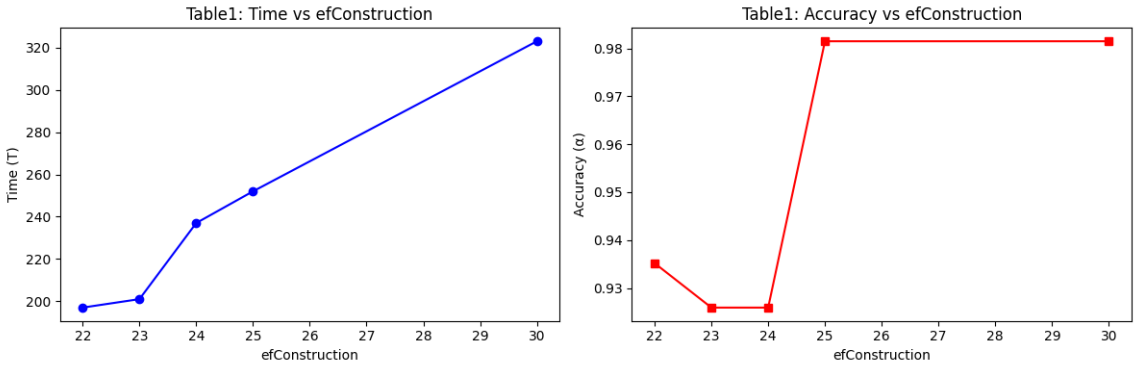

图 1: 探究 efConstruction 对于时间性能和正确率的影响

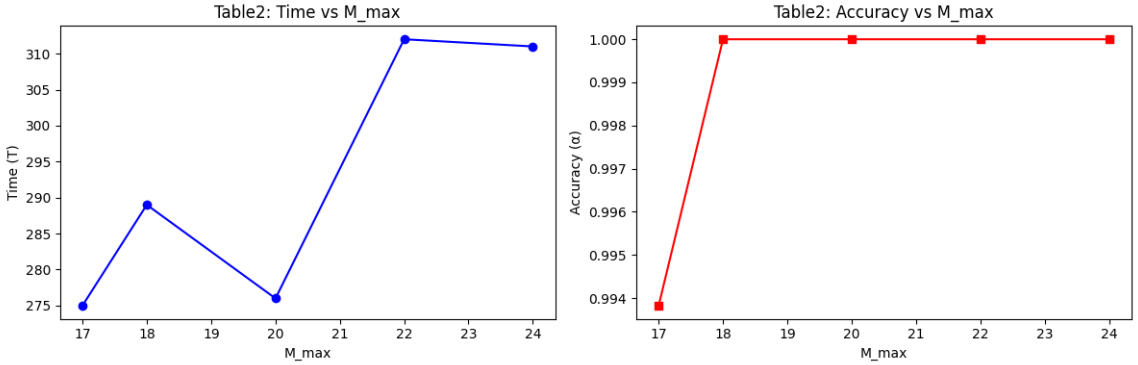

图 2: 探究 $M_{\max}$ 对于时间性能和正确率的影响

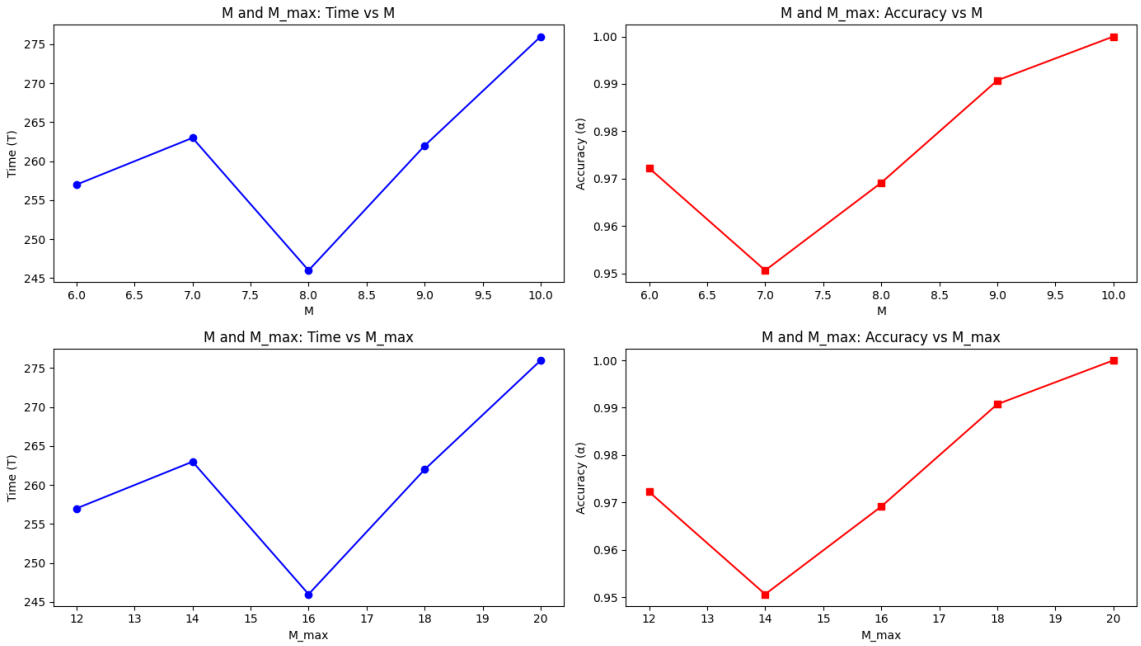

图 3: 探究 M 和 $M_{\max}$ 对于时间性能和正确率的影响

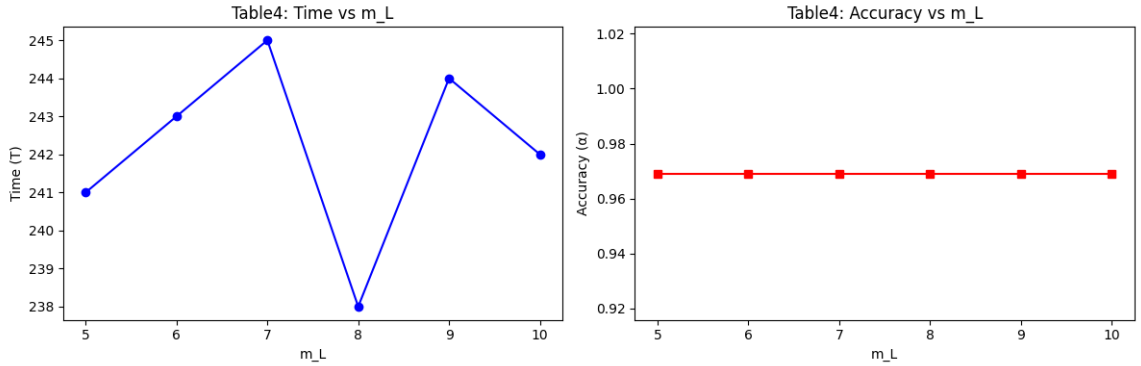

图 4: 探究 m_L 对于时间性能和正确率的影响

根据表格 1-4 的数据绘制出图 1-4, 通过观察和分析图像, 可以得出以下结论:

1. 由图 1可以看出当其他变量控制不变时, 随着 $efConstruction$ 增加, 时间性能逐渐上升, 正确率先小幅减小后大幅上升. 推测原因是由于搜索范围增加, 搜索时长自然会增加. 同时一些原本被忽略的点此时也被考虑进入, 从而正确率会有所增加.

2. 由图 2 随着 $M_{\max}$ 的增加时间性能先小幅波动后大幅下降, 正确率则快速提升后保持不

变. 推测是增加了图从链接边, 从而搜索范围增大, 并且到达了正确率的瓶颈. 其中 $M_{\max} = 2M$ 是时间性能和正确率的平衡点, 可以在两方面都有较好的表现.

3. 由图 3 基于上一步的观察, 我在 $M_{\max} = 2M$ 的前提下, 进行实验. 随着 $M_{\max}$ 的增加时间性能先小幅波动后快速阶跃下降, 正确率则快速上升后保持不变. 我们需严格控制 M 值，避免过度扩展邻居搜索范围

4. 由图 4 随着 m_L 的上升时间性能先下降后上升, 但是正确率保持不变, 在 m_L=8 处能够获得最好的时间性能. 推测是由于我设定新的节点插入时候的随机层数是由与 m_L 正相关的公式计算得出.

5. 所有的 HNSW 算法得到的时间性能都有所下降, 这与预期的结果不同, 推测原因是涉及到很多相比于 KNN 遍历之外的额外操作, 如进行图操作和计算等. 而本次实验的数据量也相对较小, 无法发挥 HNSW 的查找优势, 所以与遍历的 KNN 算法相比时间性能反而有所降低.

6. 正确率的下降与预期相符.

3 结论

首先了解到一个在更大数据规模上会有更好表现的数据结构，通过调试参数来实现正确率和时间性能这两个指标的平衡。当前的测试效果没有明显的二者提升，推测原因是数据规模较小，不能体现 $HNSW$ 的部分搜索和剪枝法的效果。

4 致谢

1. 使用 CUDA 编译时，最初的指令为：

   ```
   cmake -B build -DGGML_CUDA=ON && cmake --build build --paralleld
   ```

   由于显存占用较高，在编译期间出现内存占用过高，程序被 killed。在郑宇轩同学的指导下，将指令修改为：

   ```
   cmake -B build -DGGML_CUDA=ON && cmake --build build -j 8
   ```

   顺利完成编译，感谢郑宇轩同学的帮助。

2. 对 README 文档的理解上有一些点存在疑问，感谢巩皓文同学分享的分析博客。

   (a) `https://www.xiemingzhao.com/posts/hnswAlgo.html`

3. 使用 CUDA 进行编译执行，环境配置和指令指导来自 CSDN 当中的一篇博客