基于 LSM Tree 键值存储报告

523031910121

2025 年 3 月 21 日

1 背景介绍

随着大数据时代的到来，数据存储和检索的需求日益增长，尤其是在高并发、大规模数据场景下，如何高效地处理海量数据的写入和读取成为了一个重要的挑战。传统的数据库系统在处理大量写操作时，往往会遇到性能瓶颈，尤其是在磁盘 I/O 操作频繁的情况下。为了解决这一问题，LSM Tree（Log-structured Merge Tree）作为一种高效的数据结构被提出，并广泛应用于现代存储系统中。

LSM Tree 最早由 Patrick O’ Neil 等人在 1996 年提出，其核心思想是通过将写操作先缓存在内存中，随后批量写入磁盘，从而减少磁盘 I/O 操作的频率，提升写操作的性能。LSM Tree 通过分层存储的方式，将数据从内存逐步合并到磁盘的不同层级中，确保数据的有序性和高效检索。由于其出色的写性能，LSM Tree 已经被广泛应用于多个知名的存储系统中，如 Google 的 LevelDB 和 Facebook 的 RocksDB。

在本项目中，我们基于 LSM Tree 设计并实现了一个简化的键值存储系统。该系统支持基本的键值操作，包括 PUT（插入或更新键值对）、GET（读取键值对）和 DEL（删除键值对）。通过 LSM Tree 的分层存储结构，系统能够高效地处理大规模数据的写入和读取操作。此外，我们还引入了 Bloom Filter 和 SSTable（Sorted Strings Table）等优化技术，进一步提升系统的查询性能。

本项目的目标是通过实现一个基于 LSM Tree 的键值存储系统，深入理解 LSM Tree 的工作原理及其在大规模数据存储中的应用。同时，通过性能测试和瓶颈分析，我们将进一步探讨如何优化系统的读写性能，为后续的近似最近邻搜索（ANN）模块打下坚实的基础。

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

随着数据量的增加，从原本的只涉及内存访问变为有磁盘的读写参与。整体的吞吐量会下降，延迟会上升。但是 put 操作的吞吐量会大于 delete 和 get 操作。

2.3 实验结果与分析

我分别进行了数据量为：1000、测试数据量：1000, 5000, 10000, 20000, 30000, 50000.
键范围：1,000,000
值大小：100 字节
测试操作：PUT、GET、DEL

2.3.1 平均延迟分析

由图 1 可以观察到随着数据量的增大，延迟增加。其中 PUT 操作的变化较小，但是 DEL 和 GET 操作在 data size 大于 10k 后急剧上升。推测原因是：

1. PUT 操作：主要涉及内存操作（写入 memtable）当 memtable 满时才会触发磁盘写入。由于 memtable 是内存中的数据结构，操作速度非常快，即使数据集变大，put 操作仍然主要在内存中进行。

2. GET 操作：需要从 memtable 开始查找, 如果 memtable 中没有，需要遍历所有 level 的 SSTable。每个 SSTable 都需要磁盘 I/O 操作，数据集越大，需要检查的 SSTable 越多，磁盘 I/O 越多，因此性能会随着数据集增大而显著下降。

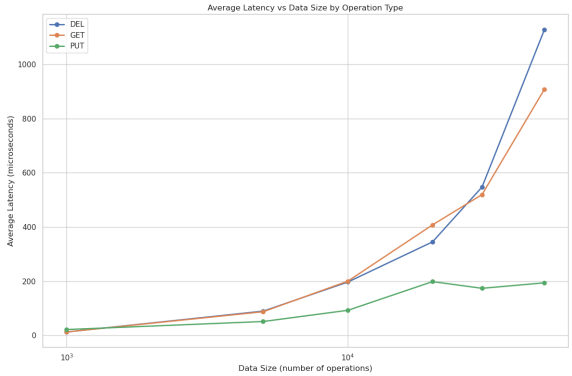

图 1: Average latency

3. DEL 操作：实际上是一个特殊的 put 操作（写入删除标记）但删除标记最终会触发 compaction。compaction 涉及大量的磁盘 I/O 操作，数据集越大，compaction 的开销越大，因此性能也会随着数据集增大而下降

4. 内存访问速度比磁盘访问快几个数量级,put 操作主要在内存中进行,get 和 del 操作涉及大量磁盘 I/O。数据集变大时，get 和 del 需要处理更多的磁盘数据，而 put 操作仍然主要在内存中进行，这就导致了性能差异的扩大。

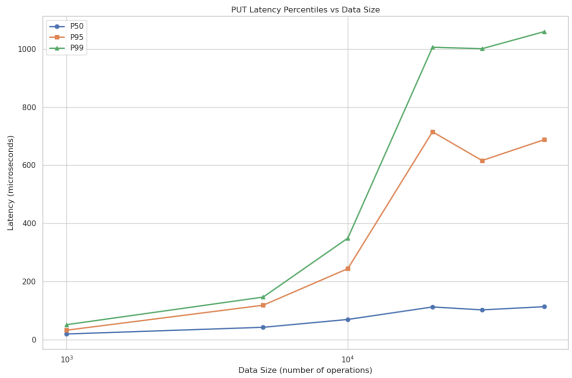

图 2: PUT percentiles

由图 2 、图 3、图 4 随着数据量的增加，P50 和 P99 延迟差距变大，反映了系统性能的波动性增加，这主要是由以下几个原因造成：

1. 磁盘 I/O 的影响：数据量增加时，更多的操作需要访问磁盘。

2. Compaction 的影响：数据量增加会触发更多的 compaction 操作。

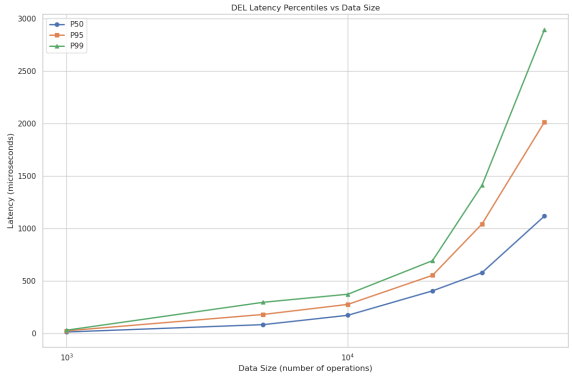

图 3: DEL percentiles

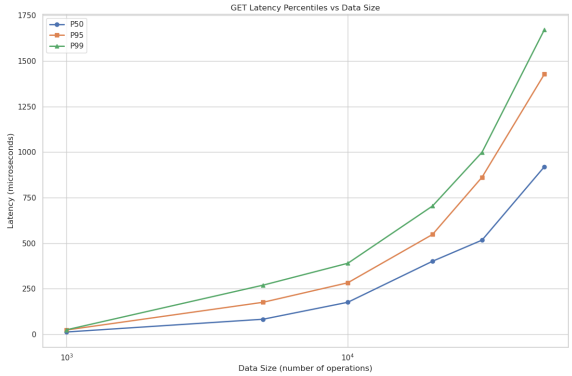

图 4: GET percentiles

3. 内存压力：数据量增加会导致更多的内存用于缓存、更频繁的内存换入换出、更频繁的 memtable 刷新。这些因素会导致性能波动。

2.3.2 Throughput

由图 5可以观察到随着数据量的增加 PUT,GET,DEL 的吞吐量都会下降，DEL 和 GET 的曲线几乎一致，变化幅度大于 PUT。推测原因是随着数据量的增加有更多的磁盘读写的参与，磁盘读写的速度较慢。而 DEL 和 GET 都是磁盘 I/O 密集型操作，都受 compaction 影响，都需要处理大量 SSTable。

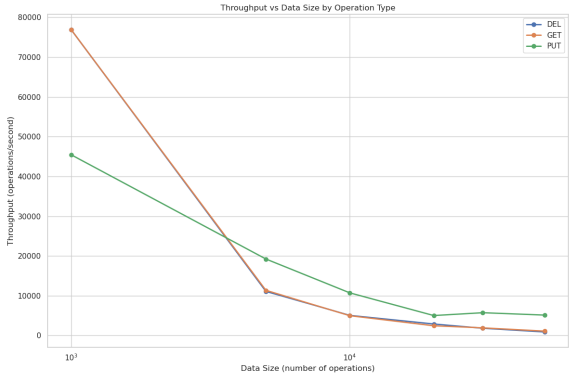

图 5: Throughput

3 结论

这个 LSM 树存储系统的实验实现了一个功能完整的键值存储系统，支持基本的 CRUD 操作，并采用了多级存储结构（memtable + 多级 SSTable）和 compaction 机制来维护数据一致性。系统架构上，使用跳表实现内存层的 memtable，提供快速的内存写入和查询；在磁盘层采用多级 SSTable 结构，通过布隆过滤器优化查询性能，并实现了 compaction 机制来维护数据一致性。性能测试结果显示，PUT 操作由于主要在内存中进行，性能最好且主要受内存大小限制；而 GET 和 DEL 操作由于涉及大量磁盘 I/O，性能相近且随数据量增加显著下降。系统的延迟分布也显示出 P50 和 P99 的差距随数据量增加而变大，这反映了系统在大数据量下的性能波动性，主要是由磁盘 I/O 和 compaction 操作导致。虽然系统在写入性能和范围查询方面表现优秀，且数据持久化可靠，但在读取性能和删除操作方面仍存在优化空间。总的来说，这个实验不仅成功实现了 LSM 树的核心特性，还通过性能测试清晰地展示了系统在不同数据量下的性能特征，为后续优化提供了明确的方向。实验结果符合预期，很好地展示了 LSM 树存储系统的特点和优势，同时也为理解多级存储系统设计和性能优化提供了宝贵的实践经验。更重要的是帮助我理解了磁盘 I/O 对系统性能的影响。但是受限于内存的影响，数据量大于 50000 时程序会被 kill，但是当前的数据量已经可以观察到明显的趋势。

4 致谢

首先是在 LSM 的基础知识理解上，深入浅出分析 LSM 树（日志结构合并树）- 康乐为的文章 - 知乎 https://zhuanlan.zhihu.com/p/415799237 此博客的动图演示极大程度上帮助我理解了这一数据结构。

其次，项目进行过程中我遇到了 WSL 崩溃的情况，查询到了 CSDN 当中的一篇博客，采取了其中的解决方案，成功解决了环境问题。

然后，此前并不了解持久性的问题，查询了网络资料以及和 DeepSeek 交互得知了几种潜在的会导致数据没有能及时写回磁盘从而读取失败的可能性。

此外，查询了一篇阿里云开发者社区的有关于软件测试中的性能瓶颈分析与优化策略的博客，了解到了如何进行软件测试和性能分析。

最后，特别感谢一起讨论的同学们，共同解决了困扰大家的 bug。还有朋友们的精神和情绪支持，让我没有过于情绪失控。

5 其他和建议

主要的困难在于跑大量的数据的时候程序执行时间很漫长，debug 一次就会耗费很长时间。而且大量的数据量让我设置了许多变量来查看过程中到底是哪里导致的 bug。并且起初没有太理解代码框架中的一些函数的作用，从而重复造轮子了。此外，数据量很大，调试的时候有一些不知道应该如何做。以及在 mergeSort 的时候不能只比较 key 和 time 还需要比较 level，否则会出现致命 bug。