LSM-KV: KVStore using Log-structured Merge Tree
性能测试报告

罗雅元 523031910750

2025 年 3 月 24 日

1 背景介绍

LSM Tree （Log-structured Merge Tree）是一种可以高性能执行大量写操作的数据结构。本项目的键值存储系统分为内存存储和硬盘存储两部分。

1. 内存存储：内存存储结构被称为 MemTable，其通过跳表保存键值对。当内存中 MemTable 数据达到阈值（即转换成 SSTable 后大小超过 2MB）时，要将 MemTable 中的数据写入硬盘。
2. 硬盘存储：采用分层存储的方式进行存储，每一层中包括多个文件，每个文件被称为 SSTable（Sorted Strings Table），用于有序地存储多个键值对。本项目中 SSTable 的大小最大为 2MB。

每个 SSTable 文件的结构分为四个部分：Header、Bloom Filter、索引区、数据区。

2 测试

2.1 实验设置

本实验旨在测试 PUT、GET、和 DEL 三种操作吞吐量（每秒执行操作个数）和平均时延（每一个操作完成需要的时间）以表现出 Compaction 对性能的影响：

- PUT (K, V)：设置键 K 的值为 V。
- GET (K)：读取键 K 的值。
- DEL (K)：删除键 K 及其值。

2.2 预期结果

由于 LSM Tree 采用分层存储机制，随着数据量增加，系统会触发 Compaction（压缩）操作，以合并不同层级的 SSTable。Compaction 过程涉及大量磁盘 I/O 操作，对系统吞吐量和时延产生显著影响。

- **PUT 操作**：初始阶段主要写入 MemTable，速度较快，吞吐量较高。当数据量超过阈值，触发 Compaction 时，吞吐量会有所下降，时延增加。
- **GET 操作**：在数据量较小时，由于查询可以在 MemTable 或较少的 SSTable 文件中完成，吞吐量较高。随着数据增长，查询可能涉及多个层级，性能略有下降，但仍优于 PUT。

- **DEL 操作**：在本实现中，删除操作通常转换为“标记删除”并写入新的 SSTable，即 PUT（K，DEL）其性能类似于 PUT。

2.3 实验结果与分析

我们分别取 100 组数据（不包含 Compaction 操作）和 1,000,000 组数据（包含 Compaction 操作）做比较，我们得到如表 1的数据：

操作	数据量	总耗时 (s)	吞吐量 (ops/sec)	平均时延 (sec/ops)
PUT	100	0.000482376	207,307	4.82e-06
GET	100	0.000023554	4,245,560	2.36e-07
DEL	100	0.000118306	845,266	1.18e-06
PUT	1,000,000	10.1045	98,966	1.01e-05
GET	1,000,000	0.203673	4,909,830	2.04e-07
DEL	1,000,000	0.237376	4,212,730	2.37e-07

表 1: 实验数据表（三种操作的吞吐量和时延）

由实验数据我们可以绘出如图 1的实验图：

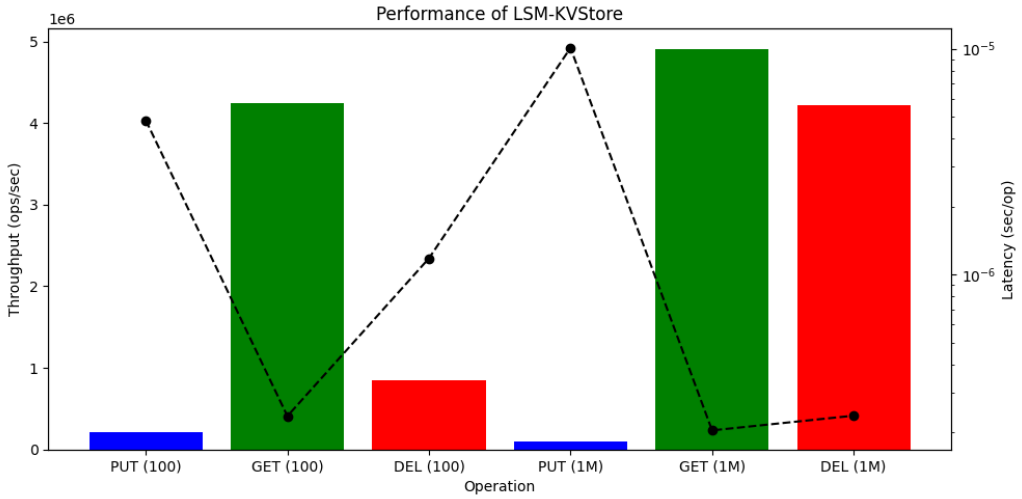

图 1: 实验数据图（三种操作的吞吐量和时延）

由实验结果可以得到：

1. PUT 操作性能分析：PUT 操作的吞吐量从 100 组测试的 207,307 ops/sec 降低到 1,000,000 组测试的 98,966 ops/sec，可能原因是数据规模变大，触发 Compaction 影响写入速度。在 100 组数据时，写入只涉及内存；而 1,000,000 组数据后，写入速度受到磁盘 I/O 限制。

2. GET 操作性能分析：GET 操作的吞吐量从 100 组测试的 4,245,560 ops/sec 提升到 1,000,000 组测试的 4,909,830 ops/sec，和预期相反，可能原因是索引和 Bloom Filter 的优化使得查询比跳表更快，合并后数据更集中，从而加快了查询速度。

3. DELETE 操作性能分析：DELETE 操作的吞吐量从 100 组测试的 845,266 ops/sec 提升到 1,000,000 组测试的 4,212,730 ops/sec，也与预期相反，可能原因 DELETE 操作只是在 GET 的基础上打上 DEL 标记，在 `Compaction` 过程中，标记删除的数据会被彻底清理。受益于 GET 查询效率的提升，删除操作也变快了。

3 结论

实验结果表明，PUT 操作在数据量增加后，吞吐量下降了约 52%，主要受到 `Compaction` 额外 I/O 限制，硬盘存储的速度比较慢；GET 操作吞吐量在数据规模扩大后反而提升了约 15%，表明 `Compaction` 通过索引和 Bloom Filter 优化，提升了查询效率。DELETE 操作吞吐量提升了近 5 倍，表明 `Compaction` 通过存入删除标记、减少查询负担，使得删除操作的效率明显提高。

`Compaction` 在提高查询和删除性能的同时，降低了写入吞吐量。因此，在实际应用中，LSM Tree 适用于写入相对较少，但查询和删除操作较频繁的场景，例如日志存储、搜索引擎索引等。

4 致谢

在本项目的过程中，我得到了许多人的帮助与支持，在此向他们表示诚挚的感谢。

我要感谢我的 AI 助手（ChatGPT），它为我提供了很多绘图代码上的帮助，帮助我高效地解决了问题。

5 其他和建议

5.1 其他优化

本 project 中的 `Compaction` 函数实现时使用循环次数较多，对性能损耗影响较大，可以进一步优化。

5.2 吐槽

一跑跑 15 分钟的代码总是让人怀疑是不是死循环了...