Lab1 LSM-Tree 报告

2025 年 3 月 9 日

1 背景介绍

LSM-Tree 是一种可以存储大量键值对的数据结构。通过采用一种延迟和分批更改索引的算法，LSM-Tree 从内存组件通过一个或多个磁盘组件级联更改索引，与 B 树等传统访问方法相比，该算法大大减少了磁盘臂的移动，提高成本性能 [1]。

本实验中的 LSM-Tree 分为 MemTable 和 SSTable 两部分，实现了 PUT、GET、DEL、RESET、SCAN 等操作，并测试了 PUT、GET、DEL 操作的性能。

2 测试

2.1 实验设置

实验运行在 Linux 环境中，使用 ctime 头文件中的 clock() 函数计算操作所用时间，通过测量吞吐量和平均时延来评估 PUT、GET 和 DEL 三种操作的性能。

为考虑各种情况，本实验对数据量为 2MB、8MB、32MB 的三种情况分别进行测试。

2.2 预期结果

数据量为 2MB 时，操作几乎全部在内存中进行，三种操作复杂度均为 O(logn)，且不涉及硬盘读写，速度最快。

数据量为 8MB 时，对内存的操作复杂度不变。PUT 操作可能导致（较少次数的）合并，涉及到 O(nlogn) 排序。GET 操作时，如果从硬盘中读取，会以较快速度定位到对应文件位置并读取字符串；如果 GET 失败，则没有从硬盘中读取的过程。DEL 操作可以看作一次 GET 操作加上一次 PUT 操作。如果 DEL 失败，则不执行 PUT 操作。

32MB 的情况与 8MB 类似，但数据量更大。

在数据类 2MB 时，由于 PUT 操作会导致跳表进行一次 search，所以 GET 快于 PUT；在数据量更大的情况下，由于 PUT 对硬盘的操作批量进行，所以 PUT 快于 GET。所有情况下，PUT、GET 都比 DEL 快，DEL 的平均时延约等于 PUT 和 GET 的平均时延相加。

2.3 实验结果与分析

数据量分别为 2MB、8MB 和 32MB 时，三种操作的吞吐量和平均时延分别如表 1、表 2、表 3。

表 1: 2MB 时的操作性能

操作名	PUT	GET	DEL
吞吐量	27057	70937	18105
平均时延（ms）	0.036958	0.014097	0.055233

表 2: 8MB 时的操作性能

操作名	PUT	GET	DEL
吞吐量	10040	7999	3578
平均时延（ms）	0.09960	0.12501	0.27947

表 3: 32MB 时的操作性能

操作名	PUT	GET	DEL
吞吐量	6755	4760	2282
平均时延（ms）	0.14803	0.21006	0.43821

实验数据与预期结果一致：数据量 2MB 时，GET 快于 PUT，其它情况 PUT 快于 GET；DEL 的平均时延约等于 PUT、GET 的平均时延相加。

3 结论

LSM-Tree 在面对大量数据时，采取对读性能的小部分牺牲提高了写性能，PUT、GET、DEL 操作的性能均较优，且 PUT 操作性能最好，DEL 性能最差。

4 致谢

感谢知乎、维基百科等博客、网站提供的参考；感谢 deepseek、kimi 等大模型提供的思路与帮助。

感谢提供支持的朋友们。

5 其他和建议

correctness test 或许可以设置得更复杂一些，例如将原来的

```
for (i = 0; i < max; ++i) {
    store.put(i, std::string(i + 1, 's'));
    EXPECT(std::string(i + 1, 's'), store.get(i));
}
```

改为

```
for (i = 0; i < max; ++i) {
    store.put(i, std::string(i + 1, (i % 26) + 97);
    EXPECT(std::string(i + 1, (i % 26) + 97), store.get(i));
}
```

原来的 correctness test 中所有字符都是 s，可能在错误的位置找到正确的字符串，导致 FAIL 信息和预期不一致，会提高 debug 难度（还是说这种设计有我没有考虑到的因素?）。

参考文献

[1] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 1996.