LSM-KV 项目报告

赵楷越 522031910803

2024 年 6 月 5 日

1 背景介绍

通过仔细圈画阅读项目 pdf 以及知乎文章《深入浅出分析 LSM 树（日志结构合并树）》[1] 后、我对 LSM-KV 有了初步的理解。在这个实验中、我们需要完成 LSM 树的实现、并完成其中键值分离式地存储在硬盘上的实现。当我们将一个键值对插入 LSM 树中时、该键值对会先存储在内存中的 Memtable 中；当 Memtable 的大小超出了限制时、我们便需要将 Memtable 中的数据分离式地写入硬盘、其中、SSTable 用于存储值的索引信息、用于查询硬盘中 vLog 文件中存储的实际数值；又因为 LSM 的插入是通过覆盖的方式实现的、因此在进行大量数据操作之后、许多先前的操作已经被覆盖了、但其对应的数据仍然存在硬盘上的 vLog 文件中、因此我们又需要对 vLog 文件进行打洞、实现垃圾回收这一功能。

本实验基于给定的知识与资料、完成了 LSM-KV 树的构建，通过了给定的正确性测试、持久性测试、并编写了对应的性能测试，在本报告中对性能测试进行了主要分析。

2 测试

2.1 性能测试

2.1.1 预期结果

基于普通 LSM 树的 KV 系统的 Merge 操作会造成写放大的性能劣势。这种写放大并非写操作时马上就会发生写放大，而是写操作发生时会潜在的导致“未来会发生”写放大，所以这种写放大只会导致整体写代价提升，不会影响实时的延迟性能。LSM-KV 的键值分离方案优化了 LSM 系统的写放大问题。根据键值分离的 LSM，对下面的性能测试、先作出结果的预期：

1. **常规分析：**理论上来说、LSM 树的插入性能是 O（1）的、但是引入了合并操作之后、暂时无法确定其合并操作对 PUT 以及 DEL 操作的平均时延造成的影响。而又由于 LSM 树不擅长区间范围的读操作，因此预期 SCAN 操作会比 GET 操作的平均时延大很多。

2. **索引缓存与 Bloom Filter 的效果测试：**预期从硬盘中读取 SSTable 会最慢、因为硬盘的数据读写是很慢的、直接使用缓存的 SSTable 进行二分查找的效率处于中等、而使用 Bloom Filter 进行辅助查找的 GET 操作的平均时延会最小，因为使用了 Bloom Filter 对查找进行了预先过滤的辅助，能提升性能。

3. **Compaction 的影响：**预期合并操作会对 PUT 操作的时延产生巨大的影响、因为合并需要涉及到硬盘数据的大量读写、导致所耗时间变长、而常规的 PUT 只需要在内存中进行操作，因此所耗时间很短。

4. **Bloom Filter 大小配置的影响：**Bloom Filter 过大会使得一个 SSTable 中索引数据较少，进而导致 SSTable 合并操作频繁；Bloom Filter 过小又会导致其 false positive 的几率过大，辅助查找的效果不好。因此预期测试结果应该会在一个中间大小的 Bloom Filter 值中取到一个性能较优点。

2.1.2 常规分析

以下四张表格包含了对 Get、Put、Delete、Scan 操作的延迟和吞吐操作的性能测定。分别在 2^{13} 至 2^{17} 次的操作数下进行了对应的性能测定。PUT 进入 LSM-KV 的值恒定为长度为 1000 的字符串。观察下列数据可以发现，SCAN 操作的确比 GET 操作的平均时延大很多，符合预期。而 GET 和 DEL 操作涉及到了 LSM-KV 中的合并操作、发现合并操作的开销会对 PUT 和 DEL 操作的整体时延造成比较大的影响、甚至导致 PUT 和 DEL 操作的平均时延远大于 GET 操作。

PUT 操作数（次）	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
平均时延（微秒）	45	57	95	159	265
吞吐量（ops/sec）	22134	17348	10438	6277	3761

GET 操作数（次）	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
平均时延（微秒）	7	9	14	21	40
吞吐量（ops/sec）	127906	107015	70802	45507	24957

SCAN 操作数（次）	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
平均时延（微秒）	61	69	74	83	94
吞吐量（ops/sec）	16155	14337	13466	11980	10548

DEL 操作数（次）	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
平均时延（微秒）	60	113	229	423	854
吞吐量（ops/sec）	16530	8845	4366	2361	1170

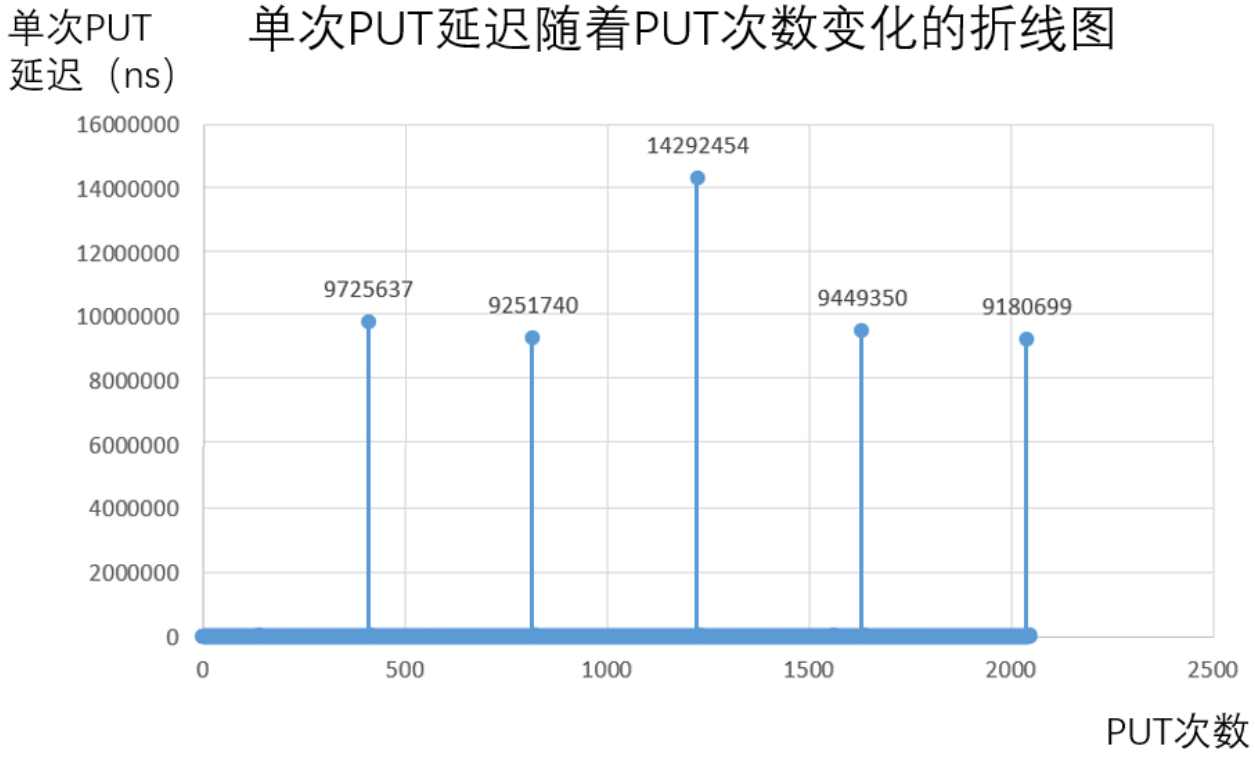
2.1.3 索引缓存与 Bloom Filter 的效果测试

测试得到要求的三种情况下 GET 操作的平均时延如下表所示。发现结果与预期近似但不完全一致。从表格中可以看出、从磁盘中读取 SSTable 的确是最慢的，但是在布隆过滤器大小为 8KB 的时候、不使用布隆过滤器的平均时延低于使用布隆过滤器的时延。分析可能原因是因为未达到过滤器的大小相较于 SSTable 大小的最优点、因此、过滤器在此数值设定的情况下进行辅助查找的效果并不理想。同时、四种操作的平均时延均随着数据总量的提升而提高、符合预期设想。

	从磁盘中读取	不使用布隆过滤器	使用布隆过滤器
平均时延（ns）	8550397	5463	5617
吞吐量（ops/sec）	117	183031	178029

2.1.4 Compaction 的影响

在不断插入数据（数据均为长度为 1000 的字符串）的情况下，本部分测试统计出了 PUT 操作的吞吐量为 35102.2 ops/sec，绘制了单次 PUT 的时延随 PUT 次数变化的折线图如下图所示。能够发现在第 409，817，1225，1633，2041 次插入的时候、单次 PUT 的延迟有了显著的增高。说明在这几次插入操作时、发生了合并操作。而其他情况下、PUT 的时延均较低、因此 PUT 操作会受合并操作的影响有巨大的性能影响、不过是当且仅当在有规律的几次插入后，不会影响大部分情况下 PUT 操作的实时性能。此部分测试结果符合预期。



2.1.5 Bloom Filter 大小配置的影响

尝试不同大小的 Bloom Filter 并保持 SSTable 大小为 16KB 不变，测得系统 Get、Put 操作的平均时延和吞吐量如下图所示。发现当过滤器从大到小变化时、PUT 操作的时延符合预期地降低了、因为所需要进行的合并操作减少了；而 GET 时延在 2KB 左右的情况会取到一个较优值、此时 Bloom Filter 的辅助查找效果最好。此部分测试结果符合预期。

过滤器大小	PUT 时延（ns）	吞吐量（ops/sec）	GET 时延（ns）	吞吐量（ops/sec）
12KB	109906	9099	8232	121468
8KB	39129	25556	6967	143523
4KB	25068	39891	6194	161433
2KB	21013	47587	6063	164909
1KB	19550	51150	6082	164393

3 结论

在本次 Project 的过程中、我感受下来最困难但也是给我收获最大的方面就是关于频繁操作文件读写的持久化实现、以及对应的垃圾回收功能的处理。从一开始只能在内存里利用 memtable 实现基本操作、到后来一步步能从内存中的 sstables 与硬盘中的 vlog 进行基本操作、再到最后实现了垃圾回收以及持久化读取的操作、到了程序能够通过全部的测试的那一刻、实验带给我的成长与收获还是很大的。

部分实验结果显示、本次实验中实现的 LSM-KV 树的性能还能有许多提高空间、因为为了保证程序大部分的正确性、在代码中尚未来得及对部分逻辑进行优化和处理、导致实验性能与理论并不完全一致。

4 致谢

1. 感谢在 Lab 制作过程中、我的舍友和我相互讨论理解要求 pdf 中的含义、比如在一开始制作的时候、理解了 offset 是从文件开始的偏移量、而不是从 tail 开始的偏移量、并分享互相的 debug 中的问题。

2. 感谢这篇文章让我清晰认识了 LSM 树《深入浅出分析 LSM 树（日志结构合并树)》

参考文献

[1] 康乐为. 深入浅出分析 *LSM* 树（日志结构合并树）. 知乎, 2021.