LSM Tree 实验报告

林祺龙 519021910380

6 月 5 日 2021 年

1 背景介绍

LSM Tree 是一种优化了写操作的储存结构，它很好地支持了快速大量的写操作。

LSM Tree 储存系统分为内存储存与磁盘储存两个部分，内存部分被称为 MemTable，可以由性能较好的字典结构保存键值对。磁盘部分被称为 SSTable，它由一个储存基本信息的 Header、一个储存布隆过滤器数据的 Bloom Filter，一系列索引数据以及一系列数据组成。

每当内存部分的储存大小达到一定上限，我们将这个 MemTable 转化为 SSTable，并存入磁盘中。磁盘中的 SSTable 被储存在不同的层次中，每一层的 SSTable 数量有一定上限，当 SSTable 数量超过这个上限时，我们需要将这一层的一些 SSTable 与下一层的 SSTable 进行合并，产生新的 SSTable。

在这个项目的实现中，由 MemTable 转化产生的 SSTable 被放入了第 0 层，每次 SSTable 数量超过上限时，我们将这一层中超过上限数量的、时间戳最小的部分文件与下一层的文件进行合并（第 0 层将所有文件向下合并），由此每一层中 SSTable 的时间戳一定大于或等于下一层 SSTable 的时间戳，这一点是我们在查找与合并的过程中需要注意的。

2 挑战

做 LSM 过程中最大的挑战应该是来源于项目的 DEBUG。。。由于我使用了 Clion 作为 IDE，但是 Linux 版本的 Clion 似乎不太能直接根据 Makefile 文件自动配置好 DEBUG 相关设置，而我自己也不太会手动配置，最后只能用 cout 进行 DEBUG 这个样子 orz

3 测试

3.1 性能测试

3.1.1 预期结果

从复杂度上来说，PUT 操作的平均复杂度应该是接近常数的，GET 操作加上缓存索引后能达到 O(LogN) 的复杂度，而 DEL 操作相当于是进行了一次查找（GET 操作）之后再进行了一次插入（PUT 操作）。

但是实际上，因为磁盘的读取会比写入快比较多（特别是操作的数据大小比较大时），实际 PUT 操作需要的时间可能不太好与 GET、DEL 操作比较。而对于 GET 与 DEL 操作，由于 DEL 操作包含 GET 操作，所以平均上 DEL 操作用时较多。又因为我们插入的删除标签" DELETE " 并不大，DEL 操作应该不会比 GET 操作慢太多。

在索引缓存与布隆过滤器部分，由于在内存中的操作比文件 IO 快很多，所以应该是在内存中缓存索引更利于提高 GET 的速度，也就是在后面的测试中速度应该是缓存了索引的快于没缓存索引的。而布隆过滤器能够以常数时间筛选掉一些不存在于索引中的键，所以应该是缓存了布隆过滤器的更快一些，但是它产生的影响没有缓存索引明显，因为它对磁盘读写操作并没有产生太直接的影响。

最后的 Compaction 部分中，由于 Compaction 可能需要对大量 ssTable 进行读写操作，包含 compaction 部分的吞吐量应该会比正常状况下的小很多。

3.1.2 常规分析

Get、Put、Delete 操作的延迟与吞吐结果如表 1，表 2所示。测试所用的代码见提交文件中 testsForReport.cc 文件的 test1-1、test1-2 部分。

在延迟部分的测试中，我分别测试了数据大小为 500，1000，1500，2000，2500 bytes 情况下三种操作的延迟，并计算出了均值。在吞吐量测试中，我选取了数据大小为 10240 bytes 的数据进行测试。

可以看到，PUT 操作的时间会比 GET 操作和 DEL 操作慢较多，而 DEL 操作稍慢于 GET 操作。DEL 操作相当于一次 GET 操作接着一次 PUT 操作，因此它相较于 GET 操作会慢一些；又由于 DEL 中存入的数据很小（"~DELETE~"），因此它没有 PUT 那样大的延迟。PUT 操作的大延

迟可能与写入磁盘的数据较多有关。

另外需要说明的是，DEL 操作随着删除数据比例的增加，其吞吐量有明显的减少。在吞吐测试中我只删除了 1/4 的数据，若删除 1/2 数据，得到的平均吞吐量会减少很多。

同时，PUT 操作的吞吐量与总数据量相关。吞吐量表中 PUT 操作的吞吐量是经过 24s 操作后取平均得到的，但是在第 1s 内 PUT 操作吞吐量达到了 12000 左右，是均值的两倍多，之后随着时间增加 PUT 吞吐量逐渐减少，才最终得到均值的数据。

表 1: 不同数据大小下 PUT, GET, DEL 操作平均延迟

数据大小/byte	PUT 延迟/us	GET 延迟/us	DEL 延迟/us
500	10.2	6.26	7.87
1000	17.8	6.87	8.67
1500	23.2	7.99	8.54
2000	30.3	7.87	9.03
2500	37.3	8.23	10.9
平均	23.7	7.44	9.00

表 2: 数据大小为 10240 bytes 时 PUT, GET, DEL 操作的吞吐

数据大小/byte	PUT 吞吐/s^{-1}	GET 吞吐/s^{-1}	DEL 吞吐/s^{-1}
10240	5869	24239	9227

3.1.3 索引缓存与 Bloom Filter 的效果测试

不同缓存情况下 GET 操作的平均延迟结果如表 3。这部分测试代码见提交代码中 testsForReport.cc 文件的 test2 部分，是否使用索引与布隆过滤器的选项定义在 ssTableCacheCell.h 中。

这里同样测试了数据大小为 500，1000，1500，2000，2500 bytes 情况下三种缓存情况的延迟。需要注意的是在无缓存情况下，表中数据的单位是 ms。

这里的结果和前面的分析一样，没有缓存索引时 GET 操作的延迟远远大于缓存索引的延迟，在这个测试中相差了两个数量级左右；而缓存布隆过滤器会提高 GET 性能，但是提高的程度没有缓存索引带来的明显

表 3: 不同缓存情况下 GET 操作平均延迟

数据大小/byte	不缓存/ms	缓存索引/us	缓存索引与布隆过滤器/us
500	2.82	7.05	6.26
1000	1.24	6.93	6.87
1500	0.883	8.08	7.99
2000	0.675	8.39	7.87
2500	0.557	9.32	8.23
平均	1.22	7.95	7.44

3.1.4 Compaction 的影响

不断插入数据情况下 PUT 操作的吞吐量结果如图 1，测试部分代码见提交代码中 testsForReport.cc 文件的 test3 部分。

测试中为提高 compaction 的频率，插入的数据大小为 10KB，测试吞吐量的时间间隔缩短为 100ms，即对应图中的横坐标单位为 100ms。

图中可以看出，包含 compaction 的时间段中 PUT 操作的吞吐量会下降很多，这是符合预测的。

4 结论

一个好的写代码环境真的太重要了 qwq。建议方面，并没有太多的建议，码就完事了

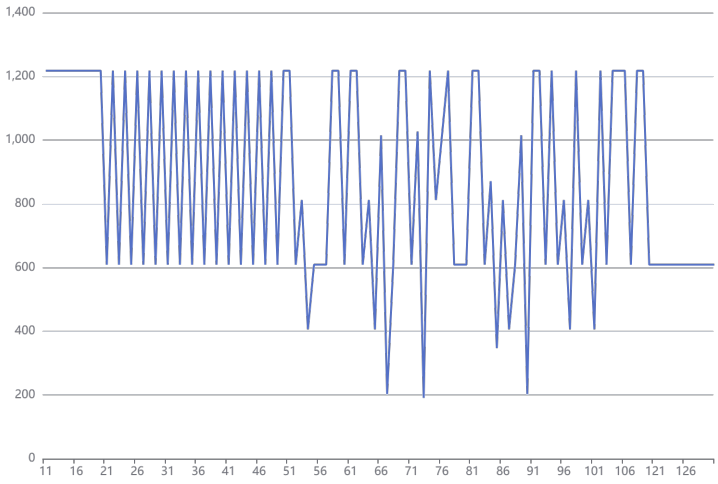

图 1: Compaction Test