LSM-KV 项目报告

池昊 522031910095

2024 年 5 月 29 日

1 背景介绍

LSM 树（Log-Structured Merge-Tree）是一种专为写密集型应用设计的数据结构，常用于实现键值存储系统。与传统的平衡树结构（如 B 树）相比，LSM 树通过减少对磁盘的写操作次数和优化写入路径来提高写入性能。LSM 树主要由两部分组成：内存中的数据结构（通常是跳表或者红黑树）和硬盘上的多个有序的不可变文件。该项目增加了 Vlog 文件用于储存键值，SSTable 中不再储存值，而是储存键对应的值在 Vlog 文件中的偏移量，从而实现了键值分离的 LSM 树。本实验通过正确性测试和持久性测试验证了实现的正确性，并通过一系列性能测试验证其操作性能。

2 测试

2.1 性能测试

2.1.1 预期结果

1. 常规分析：

   对于 Get、Put、Delete、Scan 这四种操作，我们预期随着数据大小的增加，操作的延迟也将增加。这种延迟增加主要由于数据量增大导致操作访问 SSTable 数量增加，相应 IO 操作和 Compaction 操作数量增加。我们将通过对不同数据大小执行这些操作并计算平均延迟来验证这一预期。并且吞吐量会相应减小。

2. 索引缓存与 Bloom Filter 的效果测试：

   (a) 在没有任何缓存的情况下，预期 GET 操作将展示较高的延迟，因为每次操作都需要从磁盘中检索数据。

   (b) 当索引信息被缓存时，由于减少了对磁盘的访问，预期 GET 操作的延迟将有显著降低。

(c) 当添加 Bloom Filter 后，预期在存在大量非目标数据的查询时，GET 操作的延迟将进一步减少，因为 Bloom Filter 可以有效减少不必要的 SSTable 访问。

3. Compaction 的影响：

   随着数据的持续写入，系统将定期执行 Compaction 操作。预期在 Compaction 执行期间，系统的 PUT 请求处理吞吐量将受到影响，具体表现为吞吐量的下降。

4. Bloom Filter 大小配置的影响：

   通过设置不同大小的 Bloom Filter 并保持 SSTable 大小不变，预期可以观察到 Bloom Filter 大小对 Get 和 Put 操作性能的具体影响。一个过大的 Bloom Filter 可能会导致频繁的 SSTable 合并，而过小的 Bloom Filter 可能导致较高的误判率。这将帮助确定最优的 Bloom Filter 大小，以平衡性能和资源使用。

2.1.2 常规分析

该测试中，分别测试了键的数量在 512，12K 及 48K 三种情况下 Get、Put、Delete、Scan 操作的平均延迟和吞吐量。为了避免磁盘 IO 操作对延迟的影响，将插入 Value 大小固定为 64 个字节。得到结果如表 1和表 2所示。

键数量	Get	Put	Delete	Scan
512	6.007	2.815	7.471	1050.85
12K	13.099	14.490	23.666	249275
48K	16.967	30.996	65.308	2397400

表 1: 常规测试中操作平均延迟 (ms)

由结果可得，各操作随键数量增加均表现出平均延迟增大，吞吐量减小的情况。

键数量	Get	Put	Delete	Scan
512	166480	355244	133859	951.61
12K	76340.3	69012.6	42254.6	4.01164
48K	58938.7	32262.4	15312.1	0.417118

表 2: 常规测试中操作吞吐量 (ops/sec)

2.1.3 索引缓存与 Bloom Filter 的效果测试

该测试中测试了在键数量为 12K 时，在以下三种缓存策略下 Get 操作的平均时延和吞吐量：

1. 内存中没有缓存 SSTable 的任何信息，从磁盘中访问 SSTable 的索引，在找到 offset 之后读取数据
2. 内存中只缓存了 SSTable 的索引信息，通过二分查找从 SSTable 的索引中找到 offset，并在磁盘中读取对应的值
3. 内存中缓存 SSTable 的 Bloom Filter 和索引，先通过 Bloom Filter 判断一个键值是否可能在一个 SSTable 中，如果存在再利用二分查找，否则直接查看下一个 SSTable 的索引

结果如表 3所示

缓存策略	平均时延 (ms)	吞吐量 (ops/sec)
全缓存	41.356	24180
只缓存索引	46.019	51648.3
无缓存	3969.5	251.92

表 3: 不同缓存策略下 GET 操作平均时延和吞吐量

由结果可得，无缓存策略由于频繁的读写操作会严重影响 GET 操作的性能。使用 Bloom Filter 也可以进一步提高性能，但是可能由于 Bloom Filter 实现导致值检验性能不高，其优势并不明显。

2.1.4 Compaction 的影响

该测试中记录了不断插入数据的情况下，每一次操作的平均时延，绘制折线图如图 1所示。

由于每一个 SSTable 在给定的 SSTable 和 Bloom Filter 大小下最多能储存 408 个数据，并且在第一层 SSTable 数量为 3 时触发一次 Compaction，即每插入 1224 个数据会触发一次 Compaction。由折线图可以看出，在每一次键值为 1224 倍数时有一次明显的峰值。因为本实验中 PUT 操作的实现包含了 SSTable 写入磁盘的操作，在数据量为 408 的倍数时，Latency 由于 IO 操作也会出现峰值。但没有触发 Compaction 时时延大。

2.1.5 Bloom Filter 大小配置的影响

该测试中测试了在键数量为 48K 时，在不同 Bloom Filter 大小下得到的 PUT 和 GET 操作吞吐量，结果如表 4

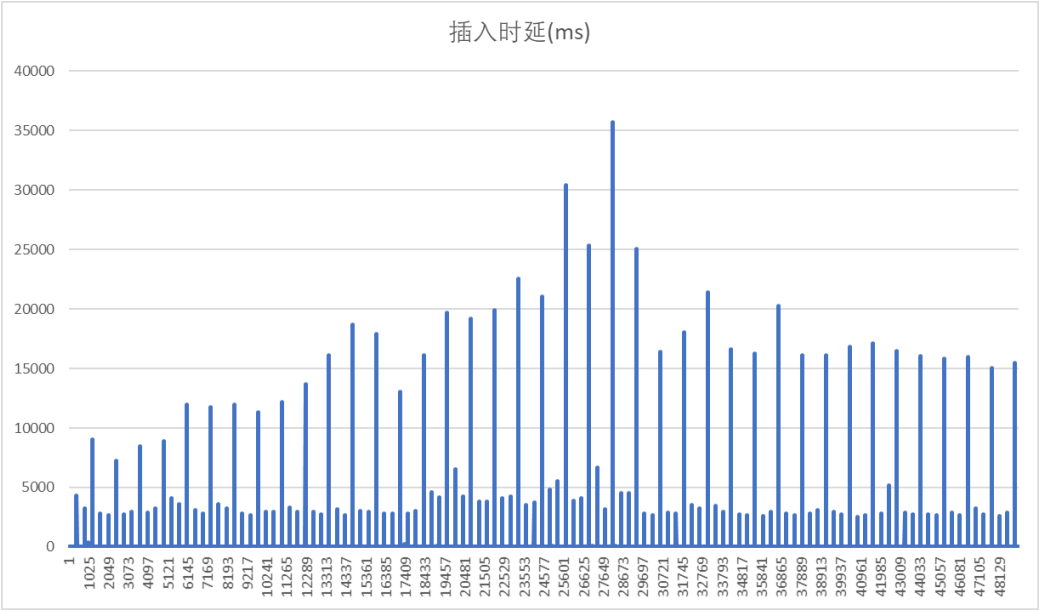

图 1: 插入操作时延 (ms)

从实际结果来看 PUT 操作有随 Bloom Filter 大小增加，吞吐量先增大后减小的趋势，而 GET 操作并不明显，在 Bloom Filter 大小达到最小 2K 字节时，吞吐量达到最大 7219.24ops/sec。这可能是因为 Bloom Filter 持续减小，虽然假阳性率增大，但被储存键值数量增加带来 SSTable 数量减少带来的影响所抵消。

3 结论

本项目通过一系列的测试验证了基于 LSM 树的键值存储系统的性能，并探索了不同的优化策略对系统性能的影响，测试结果基本符合预期。通过这些测试，我们可以得出结论，LSM 树的实现可以通过适当的优化（如索引缓存和 Bloom Filter 的使用）显著提高性能。Bloom Filter 选择合适的参数也可以提高性能。

4 致谢

感谢我的朋友侯明希在我写整个项目过程中叫我出去吃饭，缓解我 debug 带来的压力。感谢机核网的电台节目，在我写项目焦虑时带来了一点平静。感谢和山山老师的作品《为你着迷》和《女校之星》，在我项目实现的几天的闲暇时间里读完了这两部很好看的漫画。

Bloom Filter 大小	Get	Put
12K	2403.81	5656.57
10K	2530.16	6172.8
8K	2609.35	6791.74
6K	2550.76	6645.34
4K	2562.80	6962.29
2K	2523.77	7219.24

表 4: Bloom Filter 大小对操作吞吐量 (ops/sec) 的影响

5 其他和建议

在本人的拖延下，在项目发布到第十三周整个项目基本没有怎么推进。从第十四周周一开始才完成第二次迭代的要求，并且第一次通过测试。最后于周四完成了整个项目的代码实现。整个项目实现并没有花费太多时间，也没有遇到很大的困难，但是因为拖延还是给我带来了深深的焦虑感，也几度不是很想写了。最后一个 bug 是在 compaction 实现中，把整个结构中最大时间戳当作合并的 sstable 中最大时间戳赋给了新的 sstable，这一愚蠢的 bug。在做完所有项目和 lab 后，忽然想起所有 homework、lab 和 project 加起来只有整个课程分数 40% 的占比。或许三学分和 40% 并不匹配这门课的 workload 吧，我不认为减少 workload 是解决这个问题的最佳途径，或许未来可以调整学分或调整最终对分值比例构成。祝愿软院和 ADS 课程越办越好。