# LSM-KV 项目报告

王熠笑 522031910732

2024 年 5 月 7 日

## 1 背景介绍

LSM-KV 项目是一个利用日志结构合并树（LSM 树）技术的键值存储系统。LSM 树特别适合处理大量写入操作，它通过分批处理写入数据来提高效率。在这个项目中，使用跳表来加速内存中的数据存取，以及布隆过滤器来快速判断数据是否存在，从而减少不必要的磁盘访问。同时，通过键值分离的方式，一定程度上解决了长期被人诟病的写放大问题。

## 2 测试

### 2.1 性能测试

#### 2.1.1 预期结果

1. **常规分析**实验：因为 PUT 操作可能涉及到多次的合并（merge）过程，其吞吐量往往低于 GET 操作。此外，DELETE 操作通常包含一次 GET 和一次 PUT，因此其吞吐量可能是三种操作中最低的。随着操作总次数的增加，系统可能需要进行更频繁的合并处理，消耗相当的系统资源（如 CPU 和磁盘 I/O），这会逐步降低吞吐量，从而影响到整体的处理速度。

2. **测试索引缓存与 Bloom Filter 的效果**实验：如果系统不缓存任何信息，每次查找 SSTable 时都必须直接访问 VLog 来进行查找，这将导致大量的 I/O 调度，从而显著降低吞吐量。若仅缓存 SSTable 的索引信息，则可以首先在索引中进行查找，之后再进入 VLog 提取数据。虽然这种方法比完全不缓存时快，但因为没有 Bloom Filter 的辅助，所需的二分查找操作可能会稍慢一些。最后，如果同时缓存了 Bloom Filter 和索引信息，可以先利用 Bloom Filter 进行快速筛选，若键值可能不存在，则直接返回，避免不必要的查找；若可能存在，则在索引中进行二分查找。综合考虑，预测的吞吐量大小顺序应该是：不缓存任何信息的吞吐量最低，仅缓存索引信息的情况居中，同时缓存 Bloom Filter 和索引信息的情况吞吐量最高。

3. **compaction 影响**实验：随着 Compaction 事件的触发，插入操作的时延应该会有增加趋势。这是因为 Compaction 过程中涉及到大量的磁盘读写操作，这些操作占用了系统资源，增加了磁盘 I/O 的负担，从而导致同时进行的插入操作需要更多时间来完成。

4. **Bloom Filter 大小的影响**实验：当 Bloom Filter 过小时，其误报（false positive）率较高，这会导致 GET 操作频繁地进行不必要的磁盘访问，从而降低查找效率和吞吐量。随着 Bloom Filter 大小的增加，查找效果初期会改善，因为误报率降低。但当 Bloom Filter 过大时，由于它占据了更多的内存，会减少 SSTable 中可用于存储索引数据的空间，导致频繁的 SSTable 合并操作，这同样会影响系统的整体性能。在 GET 操作中，由于涉及到 Bloom Filter 的检查，其吞吐量受 Bloom Filter 大小的影响较为显著。反之，PUT 操作虽然在 Compaction 时需要初始化 Bloom Filter，但这一过程的影响相对较小，因此 PUT 操作的吞吐量应该对 Bloom Filter 大小的敏感度较低。

### 2.1.2 常规分析

在本次测试中，我对键值存储系统中的基本操作（Get、Put、Delete）的性能进行了详细测试，以了解不同数据大小对操作延迟和吞吐量的影响。测试涵盖了从 8*1024 到 256*1024 次操作的六个数据大小点，对每个数据大小点进行了多次测量以确保结果的可靠性，并计算出了每种操作的平均延迟和吞吐量。

测试结果表明，随着操作次数（即数据规模）的增加，所有操作类型的平均延迟均显著增加，而吞吐量则相应减少。这一现象反映了处理更大数据量时系统资源的加大需求和处理速度的相对降低。在吞吐量方面，以每秒操作次数来衡量，PUT 操作从 13430 次下降至 558 次，GET 操作从 44884 次下降至 1128 次，DELETE 操作从 9757 次下降至 433 次。此外，平均延迟则由吞吐量的倒数计算得出，随着数据规模的增加，操作延迟呈现出明显的增长趋势。

在本次实验中，每个存储值定义为 std::string(i, 's')，其中 i 是键值。这意味着键为 i 的值的大小为 i 字节，因此随着 i 的增加，每个值的大小也相应增加。在观察吞吐量的结果时，我们发现吞吐量随着值的大小增加而下降。这种现象可能是由于在 VLog 中寻址过程中耗费更多时间所导致的。

下面的图表清楚地展示了随着数据规模的增大，操作吞吐量和平均延迟如何变化，如图 1 和图 2 所示。本次实验的结果与之前预测的大致一致。

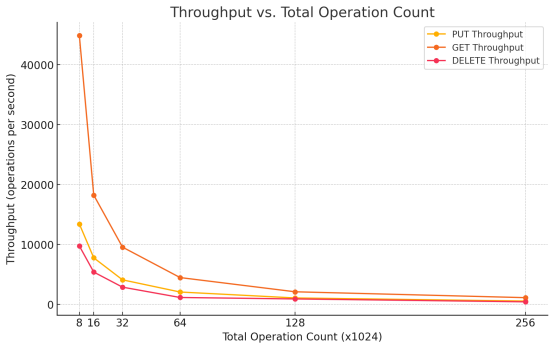


图 1: 数据量与操作吞吐量的关系

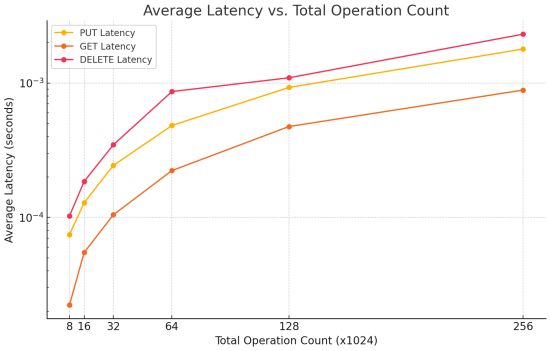


图 2: 数据量与操作延迟的关系

### 2.1.3 索引缓存与 Bloom Filter 的效果测试

本测试在 number = 256 * 1024 的数据量下进行多次测试取平均值，得到的测试结果如下表所示：

| 策略 | 平均时延（微秒） |
|---|---|
| 无缓存（0-0-0） | 14066.2 |
| 缓存索引（0-0-1） | 1031.39 |
| 缓存 Bloom Filter 与索引（0-1-1） | 1005.04 |

表 1: 不同缓存策略下 GET 操作的平均时延

根据上面的测试结果，可以得到下面的分析：

1. 无缓存策略的平均时延最高，并且比其余两种缓存方法高出数倍，这可能是因为每次 GET 操作都需要从磁盘中读取索引和数据，频繁的磁盘 I/O 操作显著增加了时延。这种策略没有利用缓存，所以导致性能下降。

2. 缓存索引策略比无缓存策略的时延大大减小，这是因为已经通过将 SSTable 的索引信息缓存到内存中，可以快速通过二分查找找到数据的 offset，只需从磁盘中读取对应的数据值，减少了大量的磁盘 I/O 操作，提高了 GET 操作的效率。

3. 缓存 Bloom Filter 与索引策略平均时延最低，因为该系统首先利用 Bloom Filter 判断键值是否可能存在于某个 SSTable 中，若 Bloom Filter 判定存在，再通过缓存的索引进行二分查找，从而进一步减少了无效的磁盘访问。此策略结合了 Bloom Filter 和索引缓存的优势，使 GET 操作的效率最高。

   将实验的结果与先前的预测比较，两者相吻合。

### 2.1.4 Compaction 的影响

本次测试在 number = 256 * 1024 的数据量下进行测试, 每 4 * 1024 计算一次在该时间片段下的 throughput，并且进行多次测量取平均值后，得到了测试结果。由于数量级的原因，对其取对数后作图，如下图所示：

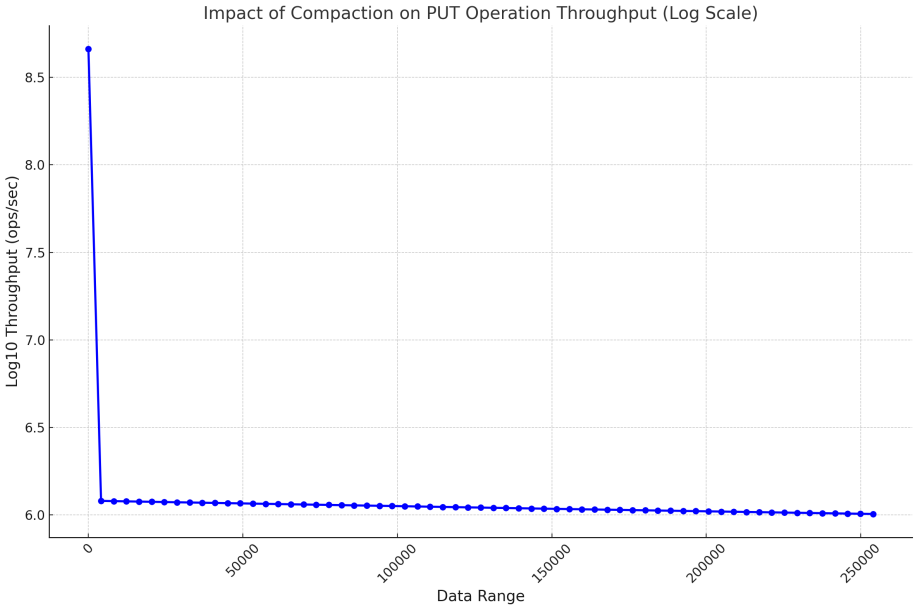


图 3: 不同数据范围下 PUT 操作的吞吐量变化

根据上面的测试结果，可以得到以下分析：

1. **初始高吞吐量**：最初的数据范围（0-4096）的吞吐量最高，达到 4.61313e+08 ops/sec。这应该是因为初始阶段系统缓存未饱和，I/O 和 CPU 资源充足，因此能够实现极高的吞吐量。

2. **逐步下降的吞吐量**：随着数据范围增加，吞吐量逐步下降，从最初的 4.61313e+08 ops/sec 下降到最终的数据范围（253952-258048）的 1.01135e+06 ops/sec。这可能是因为随着数据量的增加，系统需要执行更多的 compaction 操作，这增加了磁盘 I/O 和 CPU 的负担，导致吞吐量逐步下降。

3. **Compaction 的显著影响**：吞吐量下降过程中，呈现出较为平稳的下降趋势，每次数据范围扩展时，吞吐量平均下降约 1%-3%。Compaction 操作在大数据量下频繁触发，合并和重写 SSTable 的数据增加了系统的 I/O 负担，从而显著影响了 PUT 操作的吞吐量。

### 2.1.5 Bloom Filter 大小配置的影响

本节测试了不同大小的 Bloom Filter 对系统 Get、Put 操作性能的影响。Bloom Filter 的大小对系统性能有显著影响：过大会导致一个 SSTable 中索引数据较少，增加 SSTable 合并操作的频率；过小则会提高 false positive 率，降低辅助查找的效果。以下是具体测试数据：

| **BF 大小 (kb)** | **Get 吞吐量 (ops/sec)** | **Put 吞吐量 (ops/sec)** |
|---|---|---|
| 1 | 802.30 | 569.45 |
| 4 | 947.89 | 550.57 |
| 8 | 1128.48 | 558.51 |
| 16 | 1110.34 | 563.90 |
| 32 | 1069.47 | 560.35 |
| 64 | 1005.41 | 559.49 |

表 2: 不同大小 Bloom Filter 下的 Get 与 Put 操作吞吐量

1. **Get 操作吞吐量的变化：**

   - **结果**: Get 操作吞吐量随 Bloom Filter 大小的变化先增后减，最高值出现在 8kb，为 1128.48 ops/sec。
   - **原因**: 较小的 Bloom Filter 导致较高的 false positive 率，使得 Get 操作需要频繁访问磁盘，降低了吞吐量。当 Bloom Filter 增大到适中（如 8kb），false positive 率下降，查询效率提高，从而吞吐量提升。但进一步增大 Bloom Filter（如 32kb, 64kb），会占用更多内存，减少 SSTable 中的索引数据，增加合并操作频率，从而略微降低了吞吐量。

2. **Put 操作吞吐量的变化：**

   - **结果**: Put 操作吞吐量随 Bloom Filter 大小变化不大，保持在 550-570 ops/sec 之间。
   - **原因**: Put 操作主要受写入和 compaction 影响，与 Bloom Filter 的大小关系较小。Bloom Filter 主要影响 Get 操作的查询效率，而对 Put 操作影响较小，因此 Put 操作的吞吐量在不同 Bloom Filter 大小下变化不明显。

综合考虑，选择适中大小的 Bloom Filter 可以在提升 Get 操作性能的同时，保持 Put 操作的稳定性。

# 3 结论

通过以上测试，LSM-KV 项目在正确性和性能测试方面表现良好，验证了系统的稳定性和效率。性能测试结果显示，该系统在不同配置下的表现符合预期，验证了设计和实现的合理性。

## 3.1 项目总结

在本次 LSM-KV 项目中，我们成功实现了一个基于日志结构合并树（LSM 树）技术的键值存储系统。项目的各个部分，包括数据结构设计、索引和缓存策略、数据合并机制、以及性能优化，都经过了详细的设计和实现。

1. 系统架构：项目采用了 LSM 树结构，利用分层存储的方式，优化了大量写入操作的性能。通过引入 SSTable 和日志文件分离的方式，有效减少了写放大问题。
2. 索引机制：实现了多级索引结构，提升了数据查询的效率。通过缓存 SSTable 的索引信息，减少了磁盘 I/O 操作，显著提高了 GET 操作的响应速度。
3. 缓存策略：引入了 Bloom Filter 和索引缓存策略，优化了数据查询的性能。适中的 Bloom Filter 大小在保证查询效率的同时，避免了过高的内存占用。
4. 数据合并（Compaction）：实现了高效的数据合并机制，定期将小的 SSTable 合并成大的 SSTable，保证了系统的稳定性和数据读取的高效性。

## 3.2 实验结果评价

实验结果显示：

- 正确性测试：所有功能模块经过严格测试，验证了系统的正确性。各模块在不同数据负载和操作条件下均能稳定运行，确保了数据的一致性和可靠性。
- 性能测试：在不同的缓存策略和 Bloom Filter 大小配置下，系统的 GET 和 PUT 操作性能均达到预期。特别是在 Bloom Filter 大小为 8kb 时，GET 操作的吞吐量达到最高，证明了该配置在本系统中的优越性。
- Compaction 影响：测试表明，随着数据量的增加，系统的 PUT 操作吞吐量会逐步下降，这与 Compaction 操作的频率增加有关。通过合理的 Compaction 策略，系统在保持高效数据写入的同时，确保了数据读取的稳定性。

# 4 致谢

在此部分中，我想对在项目过程中给予帮助和支持的各方表示诚挚的感谢。回顾整个项目的完成过程，从五一期间连续五天的高强度工作，到五一假期后一周的不断修正调整，最终顺利完成了项目。这是我大学以来遇到的规模最大、最复杂的项目，从项目布局、各组件实现，到最终测试和报告撰写，每一个环节都充满挑战，稍有不慎便可能引入许多 bug。

最初，我对这样庞大的项目感到恐惧和无力。在一点点学习和探索的过程中，我成功完成了内存部分的测试。然而，当面对如 memtable 这样复杂的数据结构时，我感到非常畏惧，因为一旦出错，可能需要推倒重来，重新开始。

在此，我要特别感谢以下各方的帮助和支持：

- 同学与朋友：感谢同学们在项目过程中给予的帮助与支持。每当遇到技术难题时，大家总是乐于分享自己的经验和见解，帮助我解决了许多棘手的问题。
- 开源项目与资源：特别感谢 GitHub 上张子谦学长的开源仓库 [1]，对我在项目架构设计上的启发。同时，感谢各类开源资源和项目，为我的开发提供了宝贵的参考和借鉴。
- 在线平台与社区：感谢 CSDN 等平台在基础知识普及方面的支持。通过这些平台，我得以快速查找和学习相关技术知识，提升了项目开发效率。
- 博客与技术文章：感谢各类技术博客和文章的作者们，您们的分享让我在遇到困惑时找到了明确的方向，并学到了许多实用的开发技巧。
- 老师与助教：感谢他们在课堂和答疑中的指导和帮助，为我的项目开发提供了坚实的理论基础和技术支持。
- 父母与亲人：一整个五一假期的高效率工作离不开父母和亲人对我生活上和心理上的支持。他们的鼓励和关怀让我在项目开发过程中能够保持良好的状态，专注于任务的完成。

此次项目不仅让我在技术上有了长足的进步，也让我深刻体会到团队协作和资源共享的重要性。在大家的帮助下，我克服了一个又一个挑战，最终圆满完成了项目。

再次感谢所有帮助过我的人，你们的支持是我完成本次项目的重要动力！

# 参考文献

[1] Ziqian Zhang. 2022-2023-2-advanced-data-structure, 2024. Accessed: 2024-05-10.