学号：___M202374097___

密级：_____

华中科技大学

基于 LSM 的键值存储中的二级索引研究综述

院(系)名 称 ： 计算机学院

专 业 名 称 ： 计算机技术

学 生 姓 名 ： 潘怡珺

指 导 教 师 ： 施展 童薇

二〇二三年十二月

摘　　要

在当今数字化时代，数据规模的不断膨胀推动了存储和检索技术的进步。在大数据应用的背景下，KV 存储系统成为高效管理和处理海量数据的重要工具。而基于 LSM 的键值 KV 存储系统，以其出色的写性能和适应大规模数据操作的能力，逐渐成为学术界和工业界关注的焦点之一。本综述将聚焦于这一领域中的二级索引研究，旨在深入剖析在 LSM 架构下实现高效、灵活的键值存储系统所面临的挑战、当前的研究进展以及未来的发展方向。

关键词：LSM；二级索引；KV 存储

ABSTRACT

In today's digital era, the continuous expansion of data volume has driven advancements in storage and retrieval technologies. In the context of big data applications, KV storage systems have become crucial tools for efficiently managing and processing massive amounts of data. Within this landscape, LSM-based key-value (KV) storage systems, characterized by outstanding write performance and the ability to handle large-scale data operations, have gradually become a focal point of attention in both academia and industry. This review will focus on research related to secondary indexing in this domain, aiming to delve into the challenges faced in achieving efficient and flexible key-value storage systems under the LSM architecture, current research advancements, and future directions.

Key words: LSM; Secondary Index; KV Storage

目　　录

1　绪论

本章我们将主要围绕基于 LSM 的键值存储以及二级索引进行展开介绍，详细介绍这两者的性质结构以及市场对它们的需求及应用。

1.1　基于 LSM 的键值存储

1.1.1　LSM 树的结构

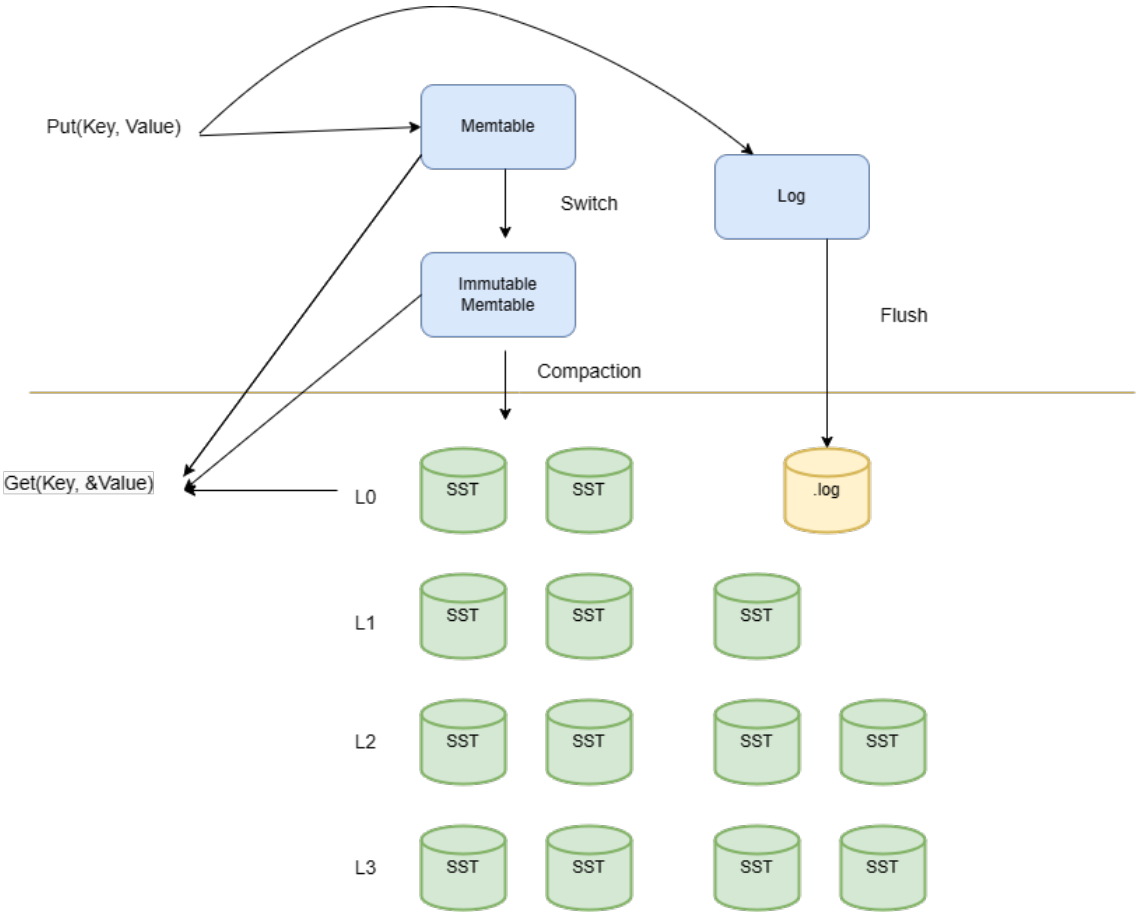

图 1.1　LSM 树的结构

LSM 树（Log-Structured Merge Tree）是一种高效的数据存储结构，通常用于实现高性能的键值存储系统，最初由 O'Neil 等人在 1996 年提出。LSM 树的核心特点是利用顺序写入来提高写性能，通过牺牲小部分读性能换来高性能写。

LSM 树由三个重要部分组成：MemTable、IM（Immutable MemTable）、SSTable。

其中，MemTable 是在内存中的数据，LSM 树会按照 Key 有序地组织这些数据，这些数据并不具有持久性，如果发生崩溃，MemTable 的数据也会消失。所以

LSM 树会使用 WAL（Write-ahead logging），即预写式日志来保证数据的可靠性。写入的数据会先在 WAL 中进行记录，用于实时落盘，以实现持久性。

当内存中的数据写满后，MemTable 会转化为 IM，即 Immutable MemTable。它是一种介于 MemTable 与 SSTable 之间的中间态，同样也存储在内存之中。在内存缓存达到一定阈值，LSM 树会把 IM 落盘到磁盘中的 L0 层。这步落盘的操作称为 minor merge，这步操作通常不对 IM 中的数据进行整理，仅进行落实。当 L0 也装满后，会触发 major merge 操作，将 L0 和 L1 的数据进行合并，进行清洗排序去重的 Compact 操作，全部整理为固定大小不可变的数据块，这些数据块即是 SSTable（Sorted String Table），其中的数据按键排序，SSTable 的大小可以在创建时自行设置。[1]

这样以来，LSM 树不同层次之间的数据是有序的，使用者可以在读取时使用合并排序算法进行快速搜索。

我们可以发现，由 MemTable 到 SSTable 的过程，是数据不断清理合并的过程，因此层次越往下，有效数据的密度就越高，数据的日期也越老旧，Compact 操作的开销也会越大。

1.1.2 LSM 树的操作

对 LSM 的操作主要为插入、查找和删除。

插入操作。插入时，数据会先在 WAL 中进行记录，再写入最顶层的内存缓存 MemTable 中。当 MemTable 存满，会转为 IM。IM 存储后则会进行 Compact 操作转为 SSTable。其中最需要注意的是 Compact 操作，它的实现高度影响 LSM 树的性能。在 Compact 操作中，多个数据文件会被合并为一个新的数据文件，这个过程存在读写的操作，也需要在新文件中进行空间分配；同时，多个数据文件中若有重复的数据，Compact 操作会删除老旧的数据文件；除此以外，Compact 操作还需要对重组的数据进行 Key 的排序，使得用户仍可以快速查询定位到目标数据；并且 Compact 操作需要对新的数据文件进行压缩从而减少存储空间的使用。由此观之，Compact 操作的优化与否紧密联系着 LSM 树的质量，通常可以通过流水线技术、复用组件、单独硬件执行等方式来实现 Compact 的优化。

查找操作。我们可以知道 LSM 树的数据新鲜度是越往上层数据越新鲜，并且 LSM 树中允许 Key 的重复存在。因此我们一般会从上到下递归地进行数据的检索

与查询。为了提高查询效率，一些 LSM 树的查询实现会使用布隆过滤器。

删除操作。LSM 树的删除操作通常不会直接删除，而是将将要删除的数据插入最顶层的内存缓存中，并将其标志为删除状态。即 LSM 树的删除操作实际上为插入操作的变种。

我们可以看到，LSM 树可以通过附加的方式大大简化写操作。LSM 树利用这种“写优先”的方式，可以实现高吞吐量的写入操作，极大地提高了写操作的效率。

1.1.3 基于 LSM 的键值存储

基于 LSM 的键值存储系统相对于传统 LSM 更注重灵活性和高效的键值查询。它在写入性能和大规模数据操作方面优越，在传统 LSM 写入操作的实现上进行了改进，引入更灵活的内存表管理和合并策略，以适应键值存储的需求，更适用于需要频繁写入和复杂查询操作的场景。同时，基于 LSM 的 KV 存储系统优化了存储结构和查询性能，提供了更为广泛的查询选项，实现了更高效的键值查询，更适应多样化的应用需求。

1.2 二级索引

二级索引的索引结构如图 1.2 右侧所示，左侧为该表的一级索引。二级索引主要由索引键和指针两个要素组成。索引键用于构建有序的索引，可以选择任意属性或字段，不限于主键。指针则是关联到实际数据行的位置，通过匹配索引键，迅速定位到对应的数据行，实现了高效的检索。[2]

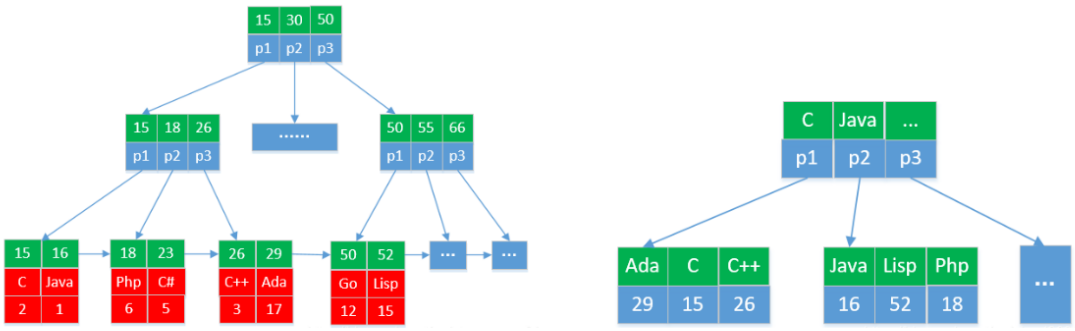

图 1.2　二级索引的结构

在操作方面，二级索引支持多种关键操作。插入操作时，系统需确保新的索引键被正确地插入到有序的索引结构中，并更新指针以指向新的数据行。删除操作通常采用标记删除的方式，将被删除数据行的索引键标记为已删除。这种延迟删除的策略有助于维持写入性能，避免频繁的物理删除带来的性能损失。

综合而言，二级索引在基于 LSM 的键值存储系统中通过其结构和操作特征，实现了对非主属性的高效查询，为系统提供了更灵活的数据检索方式。

1.3 基于 LSM 的键值存储中的二级索引的市场需求

在大数据应用的背景下，基于 LSM 的键值存储系统在应用中以卓越的写性能而脱颖而出。LSM 将写入的数据首先追加到内存表，通过后台的合并操作，将数据定期整理并写入磁盘，有效地减少了写入时的随机访问，实现了高吞吐的写性能。这使得基于 LSM 的键值存储系统在需要处理大量写入操作的场景下表现突出，如实时日志记录、传感器数据采集等。此外，由于其数据存储的方式，LSM 结构对于大量的、高密度的写入操作具有天然的适应性。这使得它成为应对互联网服务、实时分析等大规模数据应用挑战的理想选择。

而二级索引的引入则使基于 LSM 的键值存储系统的查询性能得到了较大的提高，同时增强其系统查询的灵活性，让在非主属性上进行高效的查询成为可能。这种灵活性对于满足多样化的应用需求至关重要，因为现在的很多场景下，单一主键的查询难以满足复杂的业务需求。二级索引的引入为基于 LSM 的键值存储系统提供了更灵活的数据检索方式，支持各种多条件、范围和排序查询，使其更具实用性和通用性，有助于其成为当今大数据应用需求的强大工具。

由此，诸多学者对于 LSM 的键值存储中的二级索引展开了研究，提出二级索引效率分析指标，旨在提高 LSM 存储结构的读写性能同时维护二级索引并保证崩溃一致性。

1.4 本章小结

本章我们主要介绍了 LSM 的键值存储以及二级索引的结构及其重要操作，并介绍了基于 LSM 的键值存储中的二级索引的大环境需求以及研发必要性。下一章节中我们将展示近年来针对该领域的优化策略。

2　基于 LSM 的键值存储中的二级索引

本章我们将聚焦于基于 LSM 的键值存储中的二级索引存在的问题与挑战做出阐述，并基于近年的研究成果展现学者们给出的解决方案。

2.1　问题与挑战

2.1.1　写性能问题

虽然基于 LSM 的键值存储拥有优越的写性能，然而现有流行的结构由于写放大和写停顿导致应用程序性能会周期性下降，从而导致不可预估的性能损失。[2]

虽然基于 LSM 的键值存储具有卓越的写性能，但由于写放大和写停顿的存在，导致应用程序性能定期下降，带来不可预测的性能损失。

在基于 LSM 的键值存储中，存在三种写停顿类型。首先是插入停顿，即当 MemTable 在 L0 刷新之前达到其填充限制时，所有 LSM 的插入操作都会停顿，导致写入性能的暂时下降。其次是刷新停顿，当 L0 的 SSTable 数量过多、尺寸过大时，在刷新到下一层时会受到阻塞。最后是合并停顿，过多待合并的字节会阻塞前台操作，这种速度不匹配对写性能产生级联影响。

在基于 LSM 的键值存储中存在三种可能的写停顿类型。如图所示。插入停顿：如果 MemTable 在 L0 刷新完成之前填充，则所有 LSM 的插入操作都会被停顿。刷新停顿：如果 L0 的 SSTable 过多，尺寸过大，则刷新到下一层时会被阻塞。合并停顿：过多的待合并字节会阻塞前台操作，这些速度间的不匹配会对写性能产生级联影响。

写停顿不仅会导致系统的低吞吐量，还会引起高的写延迟，从而导致长尾延迟的问题。

此外，写放大的概念被定义为写入存储设备的数据量与用户实际写入的数据量之间的比值。在基于 LSM 的键值存储中，由于频繁的压缩合并操作，写放大较高。尤其是随着数据集规模的增大，LSM 的层数越深，整体写放大也会增加。这种写放大的上升带来了对存储带宽的额外消耗，并与刷新操作相互竞争，最终减缓了应用程序的吞吐量。这一现象使得系统的性能在实际应用中变得难以预测和稳定。

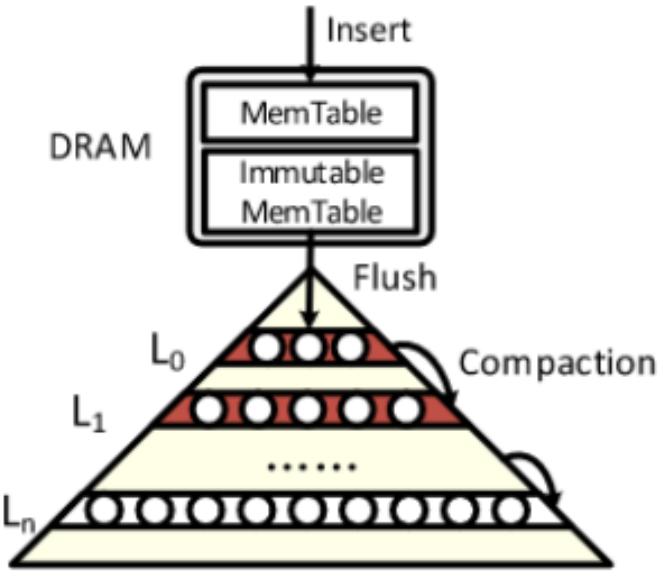

图 2.1　写性能问题

2.1.2　读性能问题

二级索引存储的值都为二级键到一级键的映射，这意味着进行二级索引查询主键前需要访问一级索引来获取记录，再从二级索引中检索主键，整个查询过程经历了从二级索引到一级索引的转变，称为索引导航。而 LSM 的键值存储本身便存在严重的读放大现象，索引导航使得这种读放大更加严重，因为一个副键可能与多个主键相关联，索引导航会对主索引进行大量的点查找，进一步恶化 LSM 的查询性能。

2.2　问题的解决与优化

2.2.1　写性能优化

传统的基于 LSM 的 KV 存储大多数都是建立在 SSD-DRAM 上，而非易失性内存技（NVM）因其可按字节寻址、具备持久性、低延迟高吞吐量特点在 LSM 的适配性上高于 SSD。学者们选择使用 NVM（例如 PM）来代替 SSD 或者与其进行混合来提高 LSM 的读写能力。然而与传统存储设备相比，PM 具有更小的原子单位，因此在做适配时需要更新数据结构来保证内存写顺序。[3][1][3]

华中科技大学的学者发现写停顿主要源于 L0 和 L1 的压缩数据量大，而写放大是 LSM 层数增加而增加，于是提出三种技术来减少写停顿和写放大的问题。

其一是引入矩阵容器，对 PM 对 LSM 的底层（L0 层）进行重新定位和管理。这一创新性的举措基于 LSM 与 PM 的高度匹配，通过利用 PM 替代 SSD，实现了对 MemTables 的数量和大小的增加。由于 PM 具有字节可寻址性和快速随机访问的特点，它能够降低 L0 到 L1 压缩的粒度，从而提升整体的写入性能。这种重新定位和管理策略为系统带来了更高的灵活性和性能优势。

其二是设计新的列压缩策略。通过每次仅压缩一个列，即选定特定的键值范围，并与压缩器中共享相同键值范围的柱进行压缩，系统在有限的数据量内能够更加精细地进行压实。这种列压缩策略不仅缩短了压缩所需的时间，也有效降低了写入操作的停顿时间。通过有选择性地对特定范围的数据进行紧凑压缩，系统能够更加高效地处理大规模的写入负载。

其三涉及增加每一层的宽度，以减少 LSM 树的深度，从而降低写放大。通过在保持相邻层之间写放大因子不变的情况下，增加每个层次的大小限制，系统实现了对层次深度的减少。这项优化措施旨在通过减少整体树的深度来减缓写放大的问题，从而提高存储系统的整体性能。[2]

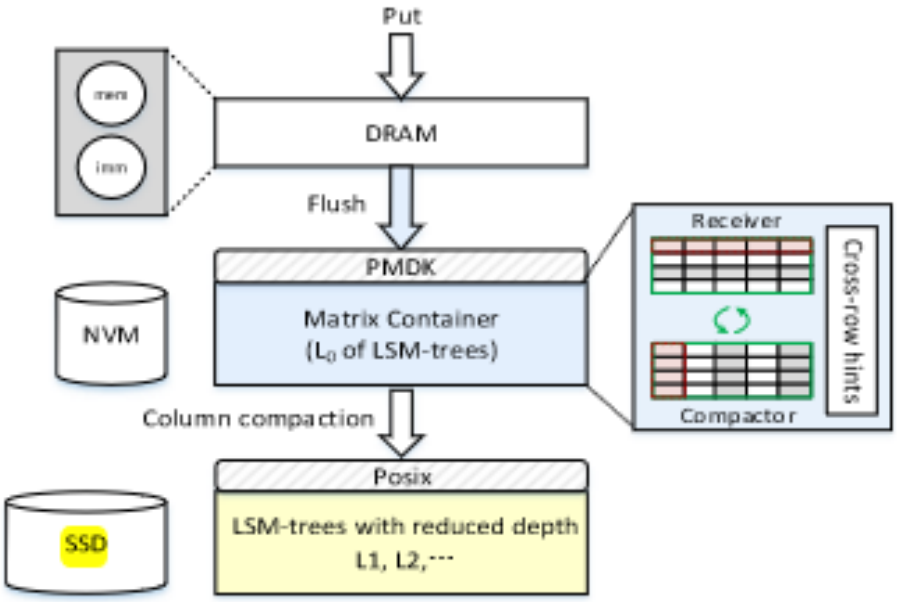

图 2.2　写性能优化

综合这三种技术，系统能够更好地应对 LSM 存储系统中常见的写放大、压缩效率和写入停顿等挑战，为存储系统的性能和效率带来了显著的提升。

2.2.2 读性能优化

由上文中的内容我们可以得知，传统的二级索引中的索引导航会带来严重的开销，即频繁对主索引进行点查找造成严重的读放大。因此来自清华的学者针对此问题设计一种独立 LSM 树的解耦二级索引结构。如图 2.3 所示。不仅一级索引存储记录地址，二级索引也存储记录地址。并且使用共享日志（SVLog）的日志结构存储记录值。所有索引共享对 SVLog 的读访问，但是只有主索引会发出写请求。这样，通过解耦的二级索引，使用者可以令二级索引单独服务于二级查询，从而减轻一级索引的查询压力并消除读放大。[4][5]

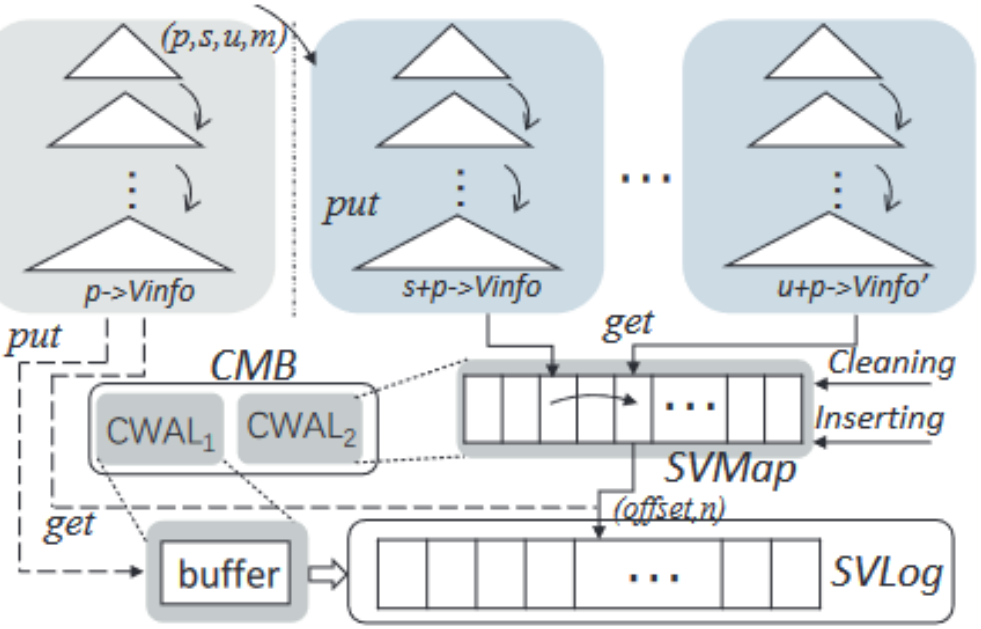

图 2.3　读性能优化

同时，他们提出种一种基于映射的懒惰索引维护机制来维护二级索引。在原有的急切更新策略下，无论二级索引是否被修改，系统都要更新所有的二级索引，每个二级索引都将执行一个 put 操作，从而导致额外的数据摄取，增加写放大。这种新型懒惰索引维护机制在更新过程中，系统会首先通过主索引获取旧的记录，然后将新纪录添加到 SVLog 中，得到 Vinfo，然后遍历所有的二级属性，与新记录值进行比较，再将复合密钥与新的 Vinfo 一起插入到二级索引中，将主密钥与 Vinfo 插入一级索引中，以延迟更新操作中未改变的次密钥的更新。

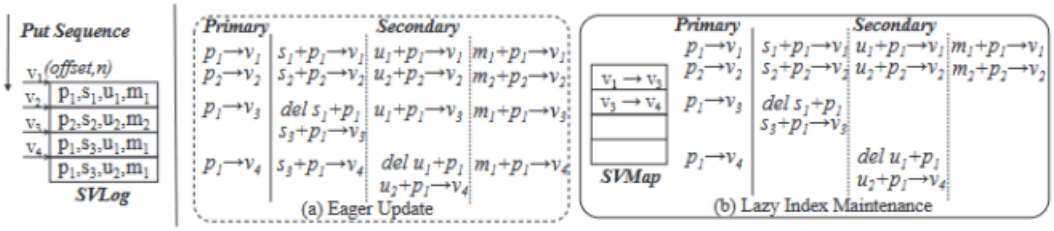

图 2.4　懒惰索引维护

2.3　本章小结

本章我们讨论了基于 LSM 的键值存储的二级索引的性能问题。三种写停顿类型包括插入停顿、刷新停顿和合并停顿。写停顿降低系统吞吐，导致高延迟。写放大高，尤其随数据规模和层数增加，影响存储带宽和吞吐。同时讨论了优化策略，包括使用 NVM、矩阵容器、新列压缩，减少层深度。对读性能，采用独立 LSM 树和懒惰索引维护机制。这些创新提升了 LSM 键值存储的性能和效率。

3 未来展望及研究方向

基于 LSM 的 KV 存储系统在二级索引方向的未来研究展望涵盖了多个关键方面，旨在不断提升系统的性能和应用广泛性。

其中，读写性能的研究还能继续发掘，在 PM 与 LSM 的键值存储的数据结构进行优化还能在读写能力上进一步提升，降低写放大和写停顿的问题，持续提高系统整体性能。同时还有泛用性，针对不同存储介质，特别是 NVM 等新兴技术，进行更深入的研究，以最大程度地发挥其对二级索引性能的潜在影响，多模型查询支持将成为未来研究的趋势，为系统提供更大的灵活性和适用性。此外，随着分布式计算的普及，关注在多节点和分布式环境下如何高效管理和查询二级索引，引入智能化的优化策略，使系统能够根据实时负载和查询模式动态调整索引结构，以提高自适应性。

今年以来，数据安全得到社会的广泛关注，关注二级索引的安全性和隐私保护，确保对敏感信息的访问得到有效控制。这些未来研究方向将推动基于 LSM 的 KV 存储系统中二级索引的发展，提高其在大规模数据管理和查询场景下的性能、灵活性和安全性。

4 总结

本文我们首先详细介绍了基于 LSM 的键值存储以及二级索引的基本概念以及市场需求，随后对基于 LSM 的键值存储的二级索引的读写性能问题进行了多方位的描述并给出了今年学者们研究出的解答。最后我们为相关发展方向进行了设想与展望。

参考文献

[1] KANNAN S, BHAT N, GAVRILOVSKA A, et al. Redesigning {LSMs} for Nonvolatile Memory with {NoveLSM}[A]. 2018 USENIX Annual Technical Conference (USENIX ATC 18)[C], 2018 : 993 – 1005.

[2] YAO T, ZHANG Y, WAN J, et al. MatrixKV: reducing write stalls and write amplification in LSM-tree based KV stores with a matrix container in NVM[A]. Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference[C], 2020 : 17 – 31.

[3] KAIYRAKHMET O, LEE S, NAM B, et al. {SLM-DB}:{Single-Level}{Key-Value} store with persistent memory[A]. 17th USENIX Conference on File and Storage Technologies (FAST 19)[C], 2019 : 191 – 205.

[4] LI F, LU Y, YANG Z, et al. Sinekv: Decoupled secondary indexing for lsm-based key-value stores[A]. 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)[C], 2020 : 1112 – 1122.

[5] QADER M A, CHENG S, HRISTIDIS V. A comparative study of secondary indexing techniques in LSM-based NoSQL databases[A]. Proceedings of the 2018 International Conference on Management of Data[C], 2018 : 551 – 566.