优化 LSM 树键值存储以降低云存储延迟峰值的综述

陈飞宇[1)]

1)(华中科技大学计算机科学与技术学院 武汉市 中国 430074)

摘 要 随着时代的发展和科技的进步，海量的数据喷涌而出，越来越多的企业选择将数据转移到云上，云存储越来越受欢迎，因为它的即用即付(pay-as-you-go)等特性显著降低了存储成本，但是云存储社区对其契约模型和 IO 延迟(contract model and latency characteristics)的探索还不够充分。随着基于 LSM 树的键值存储(LSM stores)成为众多云应用程序的构建块，云存储对键值存储的影响已经不可忽视，基于 LSM 数的键值存储的云存储可能在不同的 I/O 下存在着显著的延迟差异。本文主要对一种用于云存储的合约感知 LSM 存储（Calcspar）并结合近三年有关云存储和 LSM 键值存储的论文进行综述，梳理其研究进展。

关键词 LSM 树 键值存储 云存储 低延迟 延迟尖峰

A review of optimizing LSM-tree key-value storage to reduce cloud storage latency spikes

Feiyu Chen[1)]

1) (Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074)

Abstract With the development of the times and the progress of technology, massive data has emerged in an endless stream. More and more enterprises choose to transfer data to the cloud. Cloud storage is becoming more and more popular because of its pay-as-you-go characteristics, which significantly reduces storage costs. However, the cloud storage community has not fully explored its contract model and IO latency. As LSM-tree-based key-value stores (LSM stores) have become the building blocks of many cloud applications, the impact of cloud storage on key-value storage cannot be ignored. Cloud storage based on LSM trees may have significant latency differences under different I/Os. This paper mainly reviews a contract-aware LSM storage (Calcspar) for cloud storage and combines papers on cloud storage and LSM key-value storage in the past three years to sort out its research progress.

Key words LSM tree key-value storage cloud storage low latency latency spikes

1 引言

随着云计算技术的快速发展，越来越多的企业和组织将数据存储迁移到云端。云存储提供了可伸缩性、高可用性和灵活性等优势，使得用户可以根据需求轻松扩展存储容量，并随时随地访问数据。然而，云存储也面临着性能挑战，特别是在高并发和大规模数据访问的情况下，可能会出现延迟峰值问题。延迟峰值会导致应用程序的响应时间变慢，影响用户体验。在云存储中，延迟峰值可能由多种因素引起，如网络拥塞、磁盘 I/O 瓶颈、数据分布不均衡等。为了降低延迟峰值，提高云存储的性能和可靠性，需要对存储系统进行优化。

LSM 树（Log-Structured Merge Tree）键值存储是一种常见的数据存储结构，它具有高效的写入性能和良好的空间效率。LSM 树将数据以日志的形式顺序写入磁盘，然后通过合并操作将日志数据转换为有序的数据结构。这种结构在处理大规模写入操作时表现出色，但在高并发写入和读取操作时，可能会出现性能下降和延迟峰值问题。

因此，优化 LSM 树键值存储以降低云存储的延迟峰值成为了一个重要的研究领域。本综述旨在总结现有针对 LSM 树的优化方法，如 Calcspar[1]、ALLD 方法[2]、blk-switch 架构[3]、ALG 方法[4]、Cruisedb[5]，并探讨如何在云存储环境中降低延迟峰值[6]，提高存储系统的性能和可靠性。通过对相关研究工作的综合分析，我们可以了解当前的研究现状和发展趋势，为进一步的研究提供指导。下文将选取部分 LSM 树的优化方法，针对它们的原理和优势、研究进展依次展开

2 背景

2.1 LSM-Tree

LSM-Tree（Log-Structured Merge Tree，如图 1 所示）是一种用于存储大规模数据的数据结构，它通过将数据写入磁盘上的顺序日志文件，并定期进行合并操作，以实现高效的数据写入和读取。

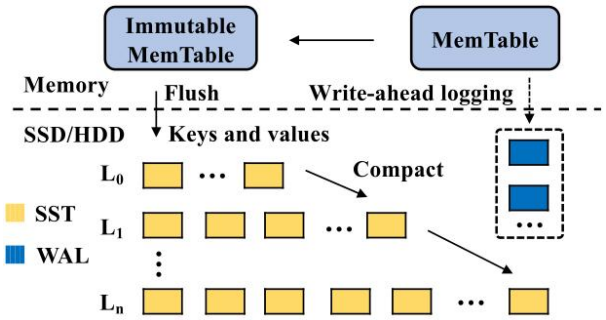

图 1 LSM-Tree 架构

LSM-Tree 的核心思想是将数据分为层级结构，每个层级都代表了不同时间范围内的数据。最底层是存储了最新数据的内存层（MemTable），当内存层达到一定大小后，会将其刷写到磁盘上的磁盘层（SSTable）。随着时间的推移，磁盘层中的 SSTable 会越来越多，为了提高查询性能，LSM-Tree 会定期对这些 SSTable 进行合并操作，将多个 SSTable 合并成一个更大的 SSTable。

LSM-Tree 的优点是可以实现高效的写入性能，因为它采用了顺序写入的方式，可以大大提高写入速度。同时，LSM-Tree 还可以通过合并操作来优化查询性能，因为合并后的 SSTable 具有更少的碎片和更高的压缩率。然而，LSM-Tree 的缺点是可能会导致读取性能下降，因为在查询时需要访问多个 SSTable。

然而，LSM-Tree 也存在一些挑战和问题，例如在高并发写入和读取操作时可能会出现性能下降和延迟峰值。为了降低延迟峰值，需要对 LSM-Tree 进行优化，这也是本综述的重点研究内容之一。

2.2 键值存储

键值存储（Key-Value Storage）是一种简单的数据存储模型，其中数据以键值对的形式进行存储和管理。在键值存储中，每个数据项都由一个唯一的键和与之关联的值组成。

键值存储的主要特点包括：简单的数据模型：键值存储只关注键值对的存储和检索，不涉及复杂的数据结构或关系；快速的读写性能：键值存储通常具有很高的读写性能，因为它们使用简单的数据结构和索引来快速定位和操作数据；灵活性：键值存储提供了很大的灵活性，可以存储各种类型的数据，无论是简单的数据类型还是复杂的对象；可扩展性：键值存储通常可以轻松扩展，通过水平扩展添加更多的存储节点来处理更多的数据和负载。

在云存储中，键值存储常用于存储元数据、配置信息、缓存数据等。一些常见的云存储提供商也提供了键值存储服务，如 Amazon[1]等。
总的来说，键值存储提供了一种简单、高效、灵活的方式来存储和管理数据，特别适用于需要快速读写和大规模扩展的应用场景。

2.3 云存储

云存储是一种将数据存储在云端的服务，通过互联网提供给用户访问和管理。云存储通常由云服务提供商运营，他们拥有大规模的数据中心和可靠的基础设施，用于存储用户的数据。

云存储的主要优势包括：可扩展性：云存储可以根据用户的需求轻松扩展存储容量，无需用户事先预测和购买大量的硬件设备；高可用性：云存储通常采用多副本存储和数据中心冗余等技术，确保数据的可靠性和可用性。即使某个数据中心发生故

障，数据仍然可以从其他数据中心恢复；便捷访问：用户可以通过互联网随时随地访问云存储中的数据，无需担心数据的物理位置和设备限制；数据备份和恢复：云存储提供了方便的数据备份和恢复功能，用户可以定期备份数据，并在需要时轻松恢复；成本效益：云存储通常采用按需付费的模式，其中最具有代表性的就是 pay as you go 的付费模式，用户只需为实际使用的存储空间和资源付费，避免了大规模的前期投资。

一些常见的云存储提供商包括 Amazon S3、Microsoft Azure Blob Storage、Google Cloud Storage 等。用户可以根据自己的需求选择适合的云存储服务，并根据使用情况进行计费。

总的来说，云存储为用户提供了一种便捷、可靠、可扩展和成本效益高的方式来存储和管理数据，适用于个人用户、企业和组织等各种场景。

2.4 延迟尖峰

延迟尖峰是指在数据传输或处理过程中，出现突然的、短暂的延迟增加的情况。这种延迟尖峰可能会导致数据传输速度突然下降，或者响应时间突然变长。

延迟尖峰可能由多种因素引起，包括网络拥塞、服务器负载过高、磁盘 I/O 瓶颈、内存不足等。在云存储中，延迟尖峰可能会影响数据的读写性能和响应时间，尤其是在高并发或大数据量的情况下。

为了减少延迟尖峰的影响，可以采取以下措施： 优化应用程序架构和代码：通过优化数据库查询、减少 I/O 操作、使用缓存等方式，提高应用程序的性能和响应时间；增加服务器资源：增加服务器的 CPU、内存、磁盘 I/O 等资源，以提高系统的处理能力和响应速度；优化网络架构：使用高速网络、优化网络带宽、使用内容分发网络（CDN）等方式，提高数据传输的速度和稳定性；监控和预警：实时监控系统的性能指标，如 CPU 利用率、内存使用情况、磁盘 I/O 等，及时发现并解决潜在的性能问题。总的来说，延迟尖峰是云存储中可能出现的性能问题之一，通过采取适当的措施可以减少其对系统性能的影响，提高数据的读写速度和响应时间。

3 研究进展

这一节将选取 Calcspar 和 ALLD 方法的原理进行介绍，并分析其实验结果。

3.1 Calcspar

为了理解云存储如何响应流量波动，作者研究了 AWS EBS 卷的延迟特性。结果表明，EBS 保证了一种称为服务水平协议(SLA)的服务协议，如果访问不超过付费 IOPS，则每个请求的处理延迟都在适当的阈值范围内。作者观察到，当一个时间窗口内的需求超过 IOPS 协议时，每个连续请求的处理延迟会急剧增加，如图 2 所示。

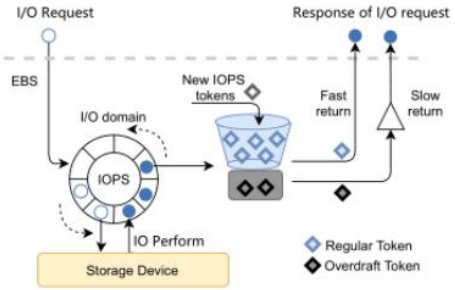

图 2 预估 EBS IOPS 阻塞机制

云存储中有限的 IOPS 导致的延迟峰值严重影响 LSM 存储上对延迟敏感的应用程序的性能。作者以 RocksDB 为例。RocksDB 首先写入内存表(memtable)，以便以合理的低延迟快速响应。直到 memtable 被填满，RocksDB 然后将表持久化到云存储卷(例如，SSTable)，从而将随机写转换为顺序写。这种写方案减少了写请求的 数量，实现了较高的写吞吐量。但是 LSM-tree 是多层的结构，一次读操作需要遍历多个层，导致读放大，再由于云存储卷上的 IOPS 有限，因此 RocksDB 的读性能会受到很大的限制。在 LSM 存储中避免读延迟爆炸（latency spike）有几个挑战。 1. 读请求性能波动很大，因为云存储容量扩展能力不够灵活，无法及时跟上工作负载的变化。由于工作负载的波动，导致 LSM 存储访问云存储的读 I/O 压力变化较大。当 IOPS 超过云存储卷的付费 IOPS 时，请求延迟会增加。2. LSM 存储中的读放大问题进一步加剧了工作负载的波动。因为 LSM-tree 的多层结构导致读取单个键值对可能需要遍历每一层，生成多个 I/O 请求。 3. 云存储卷的速度限制机制与 LSM 存储内部的多线程并发机制相冲突，云存储卷上多线程之间的请求拥塞导致延迟数倍增加。 4. LSM 存储的内部操作

加大了对云存储卷读延迟的损害。不规律的 flush 操作或不确定大小的 comaction 操作会导致访问云存储卷的 IOPS 突然增加，导致尾延迟高。 5. 为了降低的尾延迟，成本和性能之间的权衡成倍地增加了成本，导致了严重的资源浪费和有限的吞吐量提高。 对于上述挑战，一个自然的解决方案是与云存储容量购买更高的 IOPS 预算，确保 LSM 存储的 I/O 数量不超过 IOPS 上限，以保持最佳延迟。然而，这增加了成本。此外，实际生产环境中的峰值 IOPS 需求很难预测。相反，作者的目标是在特定 IOPS 预算下探索 LSM 存储的最佳性能。

基于以上的问题，作者提出了了 Calcspar，一个基于 Amazon EBS 的云存储容量合约感知 LSM 存储，它减少了延迟峰值，并且可以容忍外部工作负载波动和内部操作争用。Calcspar 首先采用波动感知缓存，它结合了热点感知的主动预取和位移感知的被动缓存，以适应不断变化的工作负载。预取策略在高负载期间识别热点并在低负载期间主动预取热点，从而平滑外部负载变化。然后，Calcspar 利用一个感知拥塞的 IOPS 分配器为不同的请求分配优先级，并避免由于有限的 IOPS 预算而增加的延迟。Calcspar 的结构（如图 3 所示）主要分为以下几个方面：

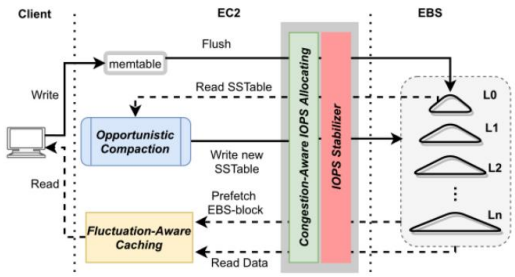

图 3 Calcspar 的整体结构

3.1.1 EBS IOPS 稳定器（IOPS Stabilizer for EBS）

为了防止 EBS 进入透支状态从而导致应用程序无法撤回挂起的请求，所以 Calcspar 不是被动地检测意外的延迟峰值，而是通过仅提交具有最高优先级的请求来主动控制高负载期间的 I/O 数量，如图 4 所示。

它调节请求速率以匹配 EBS I/O 预算，从而消除透支延迟峰值。其本质是向上层应用程序模仿 EBS 内部的令牌限速机制，该机制被广泛忽视。我们要求每个请求在访问 EBS 之前必须获得一个令牌。令牌的数量由支付的 IOPS 决定，每秒刷新一次。通过控制令牌数量，Calcspar 保证发送到 EBS 的请求不会超过支付的 IOPS，因此 EBS 处理延迟可以在几十微秒内得到保证。因此，一旦请求成功获得令牌，应用程序就可以期待更稳定的延迟。

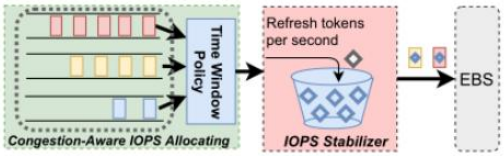

图 4 EBS IOPS 稳定器

3.1.2 感知拥塞的 IOPS 分配（Congestion-Aware IOPS Allocating）

Calcspar 的第二个目标是消除由线程之间的请求拥塞引起的延迟峰值。为了最大限度地降低云存储成本，我们假设付费 IOPS 只能保证满足平均使用量的 I/O。因此，由于 IOPS 稳定器提供的令牌有限，现代 LSM 存储所采用的多线程设计将不可避免地拥塞。许多工作通过调整 I/O 堆栈或利用多设备并行性来优先执行对延迟敏感的 I/O。然而，它们不能防止不太关键的 I/O 请求在每个时间段（例如一秒钟）占用宝贵的可用令牌。为了解决这个问题，Calcspar 使用不同优先级的多队列以及时间窗口策略，以确保关键请求不会偶尔被阻止，并以尽最大努力的方式得到服务。

多优先级的多队列：根据动态时间窗口策略对不同的 I/O 请求进行分类并分配 I/O 令牌。Calcspar 将直接影响 LSM 存储的 I/O 视为用户感知请求，并将不会立即阻碍用户请求的 I/O 分类为非用户感知请求。用户感知请求主要用于响应前台用户请求，因此 Calcspar 将它们放入优先级最高的队列中。非用户感知请求主要来自 LSM 后台任务（例如压缩和预取）。由于不同的后台任务对读/写放大的影响不同，Calcspar 将它们分配给中等优先级或低优先级队列。例如，预取请求进入优先级最低的队列。请注意，稍后会进一步区分压实请求。

动态时间窗口策略：具体来说，Calcspar 总是为高优先级队列分配一秒钟的时间窗口。对于其他队列，Calcspar 根据前一秒分配的令牌，使用公式 allocated_IOPS/Paid_IOPS 动态调整其时间窗口大小。例如，在一秒钟的时间段内，Calcspar 将时间窗口[0,1）、[0.7,1）和[0.9,1）分别分配给高优先级、中优先级和低优先级队列。在这种情况下，时间窗口[0.9，1）表示低优先级队列中的请求在第 0.9 秒

之前无法获取令牌。相反，时间窗口为[0，1）的高优先级队列中请求在到达时有资格竞争令牌。

3.1.3 波动感知缓存（Fluctuation-Aware Caching）

热点感知主动预取：当工作负载较轻时，Calcspar会消耗空闲的付费IOPS，通过预取SSTable来换取更好的缓存命中率。Calcspar以EBS块为单位管理数据（例如256KB，这是EBS允许的一个I/O请求的最大大小），这确保每个预取I/O读取尽可能多的数据。Calcspar随后维护一个全局表，以使用基于EBS块访问历史的指数平滑算法来跟踪EBS块的热度。此外，Calcspar周期性地并且主动地重新武装频繁访问的LSM顶层（例如，L0和L1）数据，因为LSM存储器以从上到下的方式检索键值对。

被动转移感知缓存：当工作负载繁重时，EBS延迟感知缓存应最大限度地减少其到EBS的I/O，同时提高空间效率。在这种情况下，Calcspar被动地管理缓存空间。Calcspar改进了以4KB为单位的缓存空间管理，并使用LRU策略来提高空间效率，因为驱逐任何数据都可能受到惩罚，因为用户请求访问EBS时会与一个I/O竞争。

缓存集成：Calcspar集成了上面介绍的两种缓存策略，并根据工作负载在它们之间切换。这两个策略管理相同的缓存空间，但在任何时候，只有一个处于活动状态并收回数据。当最高优先级的队列请求消耗95%以上的令牌时，Calcspar会认为工作负载很重，并激活Shift Aware被动缓存策略。否则，Calcspar将使用HotspotAware主动预取策略获取可用的付费IOPS。

3.1.4 机会主义压实（Opportunistic Compaction）

Calcspar的最后一个目标是修复LSM压缩I/O。

在启动压缩后，其对至少两个SSTable的批量读取操作和对至少一个SSTable的写入操作将与用户I/O请求竞争。另一方面，LSM存储逐级检索键值对，并以写时复制的方式合并SSTables，为Calcspar提供了区分不同级别压缩作业I/O的机会，从而减轻了LSM压缩操作在付费IOPS方面的竞争。对于显著影响读取I/O放大的L0表，Calcspar优先对其进行压实。对于L1和L2 SSTables，Calcspar将它们的压缩I/O放入中等优先级队列，在那里它们被机会主义地处理。对于L2以下级别的SSTables，Calcspar将这些压缩I/O分配给优先级最低的队列，因为短期延迟对性能没有明显影响。

3.1.5 实验评估

作者采用部署最广泛的AWS作为我们的测试平台。EC2实例为m5d2xlarge，配置有8个vCPU和32 GB内存。默认情况下，性能评估使用具有100 GB容量和1000 IOPS的代表性io2存储卷。使用两个benchmark，Mixtraph和YCSB来评估性能。YCSB是一个广泛用于评估键值存储系统的基准，提供了表1中列出的六种工作负载配置和键值对访问分布模型。YCSB还可以提供统一分布的工作负载。Mixtraph是Facebook开发的最新基准测试。工作负载在空间上更加本地化，以更好地模拟Facebook生产工作负载，并生成更准确的键值查询。默认情况下，基准测试在所有键值存储中的10个线程中运行，SILK除外，因为不支持多线程。在所有实验中，首先将1亿个键值对插入到键值存储系统中，并且键值存储在其初始状态下具有大约25GB的数据。

表 1 YCSB 工作负载

Workload	Description
A	write-intensive:50% Update, 50% Read, Zipfian
B	read-intensive: 5% Update, 95% Read, Zipfian
C	read-only: 100% Read, Zipfian
D	read-latest: 95% Read, 5% Insert, Latest
E	scan-intensive: 5% Update, 95% Scan, Zipfian
F	write-intensive:50%Read,50%read-modify-write,Zipfian

实验的总体结果分析如下：

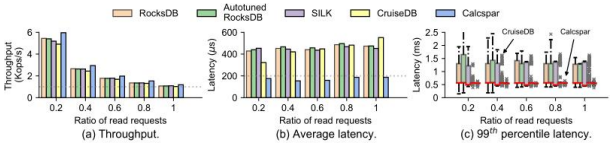

图 5 Mixgraph 工作负载下不同读取请求比率的评估结果

Calcspar 在所有读取比例下的吞吐量都优于其他系统： 如图 5(a)所示，在不同的读取比例下，Calcspar 的吞吐量始终超过其他系统。这是因为 Calcspar 充分利用了 Mixgraph 负载的高空间局部性，从而实现了更高的性能。

Calcspar 显著降低了平均延迟： 如图 5(b)所示，Calcspar 的平均延迟不超过 200μs，比其他键值存储系统低 45~ 66%。这表明 Calcspar 在处理读写请求时具有更高的效率，能够更快地响应客户端。

Calcspar 实现了更低且更稳定的尾延迟：如图5(c)所示，99th 百分位数延迟的统计图表显示，Calcspar 的箱线图体积最小，几乎没有超长延迟的异常值。这意味着 Calcspar 在处理高并发请求时，能够保持更低且更稳定的尾延迟，减少了用户在请求处理过程中的等待时间。

Calcspar 在吞吐量、平均延迟和尾延迟方面具有优势：综合以上三个结论，可以看出 Calcspar 在性能方面表现优秀，尤其在吞吐量、平均延迟和尾延迟方面具有优势。这使得 Calcspar 在处理大规模数据存储和高并发请求时，能够提供更好的性能和用户体验。

3.2 ALDC

LSM 树（Log-Structured Merge Tree）是一种常见的键值存储结构，它通过将数据分层存储在不同的层级中，以提高写入性能。在 LSM 树中，数据首先写入内存中的 memtable，当 memtable 达到一定大小后，会将其刷写到磁盘上的 SSTable 中。随着时间的推移，会产生多个 SSTable，为了保持 LSM 树的性能，需要定期进行压缩操作，将多个 SSTable 合并成一个新的 SSTable。传统的 LSM 树压缩方法通常是由上层驱动的，即当上层的 SSTable 达到一定大小或数量时，触发压缩操作。这种方法的缺点是可能会导致不必要的 I/O 开销，因为它没有考虑底层 SSTable 的实际情况。

ADLC 方法的核心思想是根据底层 SSTable 的实际情况来触发压缩操作（如图 6 所示），以减少不必要的 I/O 开销。具体来说，ADLC 方法会定期评估底层 SSTable 的大小、数量和访问频率等因素，并根据评估结果来决定是否触发压缩操作。如果底层 SSTable 的大小或数量超过一定阈值，或者某个 SSTable 的访问频率较低，那么就会触发压缩操作。

ADLC 方法还提出了一种称为“切片合并”的技术，来进一步优化压缩操作。切片合并是指将多个相邻的 SSTable 合并成一个新的 SSTable，这样可以减少 I/O 开销和内存占用。

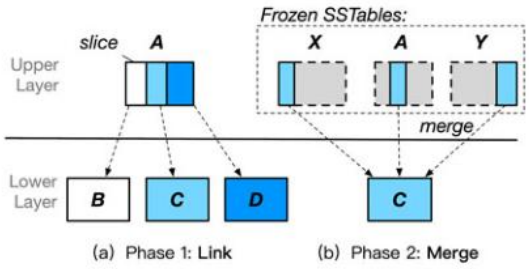

图 6 底层驱动压缩

3.2.1 定期评估底层 SSTable

定期评估底层 SSTable 是 ADLC 方法的核心部分。具体来说，ADLC 方法会定期评估底层 SSTable 的以下几个方面：

大小：评估底层 SSTable 的大小可以帮助判断是否需要进行压缩操作。如果某个 SSTable 的大小超过了一定阈值，那么就可以触发压缩操作。

数量：评估底层 SSTable 的数量可以帮助判断是否需要进行合并操作。如果底层 SSTable 的数量过多，那么就可以触发合并操作。

访问频率：评估底层 SSTable 的访问频率可以帮助判断哪些 SSTable 是最常用的，从而可以优先考虑对这些 SSTable 进行压缩或合并操作。

ADLC 方法通常会使用一些算法来定期评估底层 SSTable，例如定时轮询、基于计数器的算法等。这些算法可以根据具体的应用场景和需求进行选择和调整，以达到最佳的性能和效率。

总之，定期评估底层 SSTable 是 ADLC 方法的一个重要优势，它可以帮助提高 LSM 树的性能和效率，减少不必要的 I/O 开销，从而提高键值存储的性能。

3.2.2 根据评估结果触发压缩操作

根据评估结果触发压缩操作的过程可以分为以下几个步骤：

确定压缩策略：根据评估结果，确定需要进行压缩的 SSTable 和压缩策略。压缩策略可以包括选择合适的压缩算法、设置压缩级别等。

选择压缩时机：根据压缩策略，选择合适的压缩时机。压缩时机可以根据 SSTable 的大小、访问频率、存储空间等因素来确定。例如，可以选择在 SSTable 的大小超过一定阈值时进行压缩，或者在访问频率较低时进行压缩。

触发压缩操作：在确定压缩时机后，触发压缩操作。压缩操作可以包括读取 SSTable 的数据、使用压缩算法进行压缩、写入压缩后的数据等步骤。

在压缩过程中，需要注意避免对正在进行读写操作的 SSTable 进行压缩，以免影响系统性能。

更新索引：在压缩完成后，需要更新索引以反映压缩后的数据布局。更新索引可以包括更新 SSTable 的元数据、更新 B+树索引等步骤。

释放空间：在压缩完成后，释放压缩前占用的空间。释放空间可以包括删除原始 SSTable 的数据、更新存储空间统计信息等步骤。

需要注意的是，触发压缩操作的过程需要考虑系统的性能和资源利用率等因素，以确保压缩操作不会对系统的正常运行造成影响。同时，还需要根据具体的应用场景和需求，选择合适的压缩算法和压缩级别，以达到最佳的压缩效果。

3.2.3 切片合并技术

切片合并技术是 ADLC 方法中的一种优化技术，它可以进一步提高 LSM 树的性能和效率。具体来说，切片合并技术可以将多个相邻的 SSTable 合并成一个新的 SSTable，从而减少 SSTable 的数量和 I/O 开销。

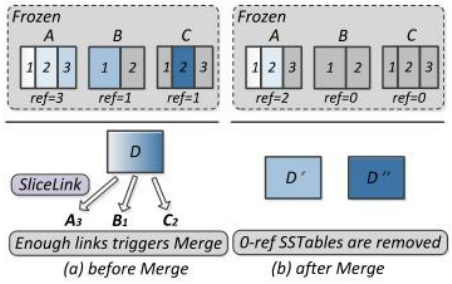

图 7 切片合并技术

切片合并技术的实现过程可以分为以下几个步骤（如图 7 所示）：

选择合并切片：根据一定的策略选择需要进行合并的相邻切片。合并切片的选择可以根据切片的大小、访问频率、存储空间等因素来确定。

读取数据：读取需要合并的相邻切片的数据，并将其存储在内存中。

合并数据：在内存中对相邻切片的数据进行合并。合并过程可以包括排序、合并、去重等操作，以确保合并后的数据具有正确的顺序和完整性。

写入合并后的数据：将合并后的数据写入一个新的 SSTable 中。在写入过程中，需要注意避免对正在进行读写操作的 SSTable 进行合并，以免影响系统性能。

更新索引：在合并完成后，需要更新索引以反映合并后的数据布局。更新索引可以包括更新 SSTable 的元数据、更新 B+树索引等步骤。

切片合并技术可以有效地减少 SSTable 的数量和 I/O 开销，提高 LSM 树的性能和效率。但是，在使用切片合并技术时，需要注意合并时机和合并策略的选择，以避免对系统性能造成影响。同时，还需要根据具体的应用场景和需求，选择合适的合并算法和合并级别，以达到最佳的性能和效率。

3.2.4 实验结果

比较对象：

为了评估我们提出的 LDC 和 ALDC 的效果，它们在 LevelDB v1.19 中实现，我们引入了五种不同类型的代表性比较对象：

UDC：LevelDB v1.19 与默认的分层压缩机制（即上层驱动压缩 UDC）相结合。

RocksDB（分层）：RocksDB 的默认版本也采用 UDC，即分层压缩。在我们的实验中，我们使用 RocksDB v5.10，并使用与 LevelDB 系统相同的配置进行公平比较。

RocksDB（通用）：RocksDB 还支持另一种压缩方式，称为通用压缩，这意味着所有 SSTables 都可以有重叠的范围，并且大量 SSTables 可以参与一轮压缩。与默认的分层压缩相比，它通常具有更好的写入性能和更低的读取性能。

RocksDB（自动调整）：RocksDB 的一项新功能，可以对后台操作（即从内存到磁盘的刷新数据和压缩工作）的总 I/O 带宽进行自我调整。

CuttleTree：哈佛大学针对 LSM 树上层驱动压缩的自适应工作的代表性示例。它根据工作负载的读/写请求比率动态调整第一级大小和 LSM 树的扇出。

LevelDB 和 RocksDB 是 LSM 树键值存储的不同工程实现，具有相同的原理、相似的工作流程和不同的工程细节。在实验中，LevelDB 和 RocksDB 的重要参数设置为相同的值，以进行公平比较。例如，将 memtable 大小设置为 8 MB，SSTables 的大小为 4 MB，bloom filter 大小为每个键 10 位，后台线程数为 1。

实验评估中使用的 YCSB 工作负载如表 2 所示。

表 2 实验中使用的 YCSB 工作负载

Workload	Query Type	Feature
Load	None	Write Only (100% writes)
A	Point Lookups	Update Heavy(50% writes)
B	Point Lookups	Read Mostly (95% reads)
C	Point Lookups	Read Only (100% reads)
D	Point Lookups	Read Latest (95% reads)
E	Range Queries	Short Ranges (95% reads)
F	Point Lookups	Read-modify-write (50% reads)
Splc	Both	Splicing *Load* and *A* ∼ *F*

实验将评估 UDC、LDC 和 ALDC 在 Splc 下的平均延迟和尾部延迟，总请求数为 6000 万。图 8 展示了不同百分位数（即 P90、P95 和 P99）的所有解决方案的平均延迟和尾部延迟。

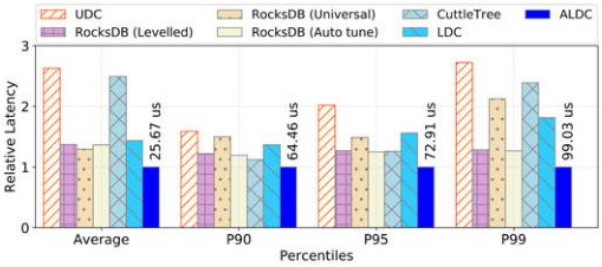

图 8 不同百分位数的所有方案的平均以及尾延迟

首先，基于 RocksDB 的解决方案确实比基于 LevelDB 的解决方案平均延迟和尾部延迟更低，除了 RocksDB（Universal）。RocksDB（Universal）具有更高的压缩效率，因此可以实现非常好的平均延迟（即除了 ALDC 之外的第二好）；但由于其过大的压缩粒度，RocksDB（Universal）的尾部延迟比其他两个基于 RocksDB 的解决方案要高得多。

其次，LDC 和 ALDC 可以有效降低 LevelDB 的平均延迟，因为它们可以减少由于新提出的较低级别的驱动方式而导致的压缩 I/O 放大。并且，ALDC 通过自适应策略进一步减少了压缩 I/O 放大，因此其平均延迟最低，即 25.67 毫秒。

第三，ALDC 在所有情况下始终实现最低的尾部延迟（即 P90、P95 和 P99）。例如，与 UDC 相比，ALDC 将 P95 尾部延迟降低了 50.51%，将 P99 尾部延迟降低了 63.30%。LDC 和 ALDC 可以通过将大的压缩作业拆分为高效的小碎片来有效减少长延迟的数量。因此，用户请求不会长时间被挂起，导致相对较小的尾部延迟。这也在图 10 中的实时吞吐量跟踪中得到了体现。虽然 UDC、LDC 和 ALDC 的实时吞吐量都由于 LSM 树键值存储的批处理压缩 I/O 而波动，但 ALDC 的最低吞吐量高于 LDC，远高于 UDC，结果是等待时间更短，尾部延迟更低。

4 总结和展望

本文主要根据基于 LSM 树键值存储的云存储的发展现状，对 LSM 树、键值存储以及云存储的背景进行了介绍，并且挑选了两个比较有代表性的 LSM 树键值存储的优化方案进行了综述，但是关于基于 LSM 树键值存储的云存储的优化方案远远不止文中提到的这些。后续可以从更多的方案进行优化，比如：优化 LSM 树的压缩时间，优化 LSM 树的压缩算法等。希望后续的研究者们可以进一步进行研究从而进一步提高云存储性能。

参 考 文 献

[1] Yuanhui Zhou , Jian Zhou, WNLO, Shuning Chen, PingCAP. Calcspar:A Contract-Aware LSM Store for Cloud Storage with Low Latency Spikes. In2023 USENIX Annual Technical Conference,(ATC 23), pages 451 – 465, July2023.

[2] Yunpeng Chai, Y anfeng Chai, Xin Wang, Haocheng Wei,and Y angyang Wang. Adaptive lower-level driven compaction to optimize lsm-tree key-value stores. IEEE Transactions on Knowledge and Data Engineering,34(6):2595 – 2609, 2022.

[3] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. Rearchitecting linux storage stack for µs latency and high throughput. In 15th USENIX Sympo- sium on Operating Systems Design and Implementation(OSDI 21), pages 113 – 128. USENIX Association, July2021.

[4] Lucas Lersch, Ivan Schreter, Ismail Oukid, and Wolf-gang Lehner. Enabling low tail latency on multicore key-value stores. Proc. VLDB Endow., 13(7):1091 – 1104, mar 2020.

[5] Junkai Liang and Y unpeng Chai. Cruisedb: An lsm-tree key-value store with both better tail throughput and tail latency. In 2021 IEEE 37th International Conference on Data Engineering (ICDE), pages 1032 – 1043. IEEE, 2021.

[6] Mingzhe Liu, Haikun Liu, Chencheng Y e, Xiaofei Liao, Hai Jin, Y u Zhang, Ran Zheng, and Liting Hu. Towards Low-Latency I/O Services for Mixed Workloads Using Ultra-Low Latency SSDs. In Proceedings of the 36th ACM International Conference on Supercomputing (ICS 22), New Y ork, NY , USA, 2022. Association for Computing Machinery.

附录 1.

1. Calcspar可以从EBS迁移到其余的云存储上吗？

因为Calcspar是在研究EBS内部特征的基础上所设计的，特别是对EBS的延迟模型做了估计，所以Calcspar是对EBS所设计的，不能轻易的进行迁移，但是如果其余的云存储和EBS有相同的延迟模型那么可以对其进行借用，或者对Calcspar的部分特性进行借鉴。如果一个基于LSM树的云存储模型在进行大量数据操作时带宽较低则可以借鉴Calcspar中的机会主义压实操作，选择合适的时间对数据库中的SST进行合并，避免影响前台带宽；如果一个其余的数据库因为线程之间的请求拥塞引起了延迟峰值，那么可以利用Calcspar提出的拥塞感知的IOPS分配进行改进等；类似的如果一个基于LSM树的KV云存储在运行的过程中流量存在很强的波动，那么就可以利用Calcspar提出的波动感知缓存技术，进行热点感知主动预取和被动转移感知缓存等，即在工作负载较轻时，通过预取SST来换取更好的缓存命中率。所以Calcspar虽然不能进行迁移但是可以对其思想进行借鉴，从而改进其余的云存储。

2.Calcspar可以提供什么样的服务保障（如断网等）？

论文中Calcspar只是一个用户与EBS之间的中间件，主要功能是避免数据的波动和减少高峰时期的IOPS从而提高EBS云存储的速度和效率，因为其只是一个中间件所以其不提供如断网之类的服务保障机制。

3.EBS的令牌运行机制与我们熟知生活中部分运行机制是否有相同之处？

与百度云的下载功能类似。当Calcspar中的Submit IOPS不超过付费IOPS时，请求会获得常规令牌并快速返回，线程之间的请求不会阻塞，从而获得最佳延迟，但是当常规令牌获取完之后就会获取透支令牌不保证响应速度，与百度云中的免费下载速度类似，即无论你的网速多快，但你达到了一定的速度之后都无法进一步提高速度。付费用户则相当于常规令牌是无限的，即无论请求的数量多少只要在网速允许的情况下都可以将请求以最快的速度返回，获取最快的下载速度。