LSM-tree 研究的演进历程与关键技术综述

Ethan DENG Dongsheng DENG
Fudan University PA Technology

日期：2025 年 5 月 17 日

1 介绍

LSM-tree 是一种面向写密集型负载优化的存储结构，已广泛应用于 LevelDB、RocksDB、HBase、Cassandra 等主流键值存储和 NoSQL 系统中。LSM-tree 通过顺序写入和延迟合并，有效提升了写性能，但同时也引入了显著的读开销和写放大等问题。围绕这些挑战，过去十余年中学术界和工业界展开了大量研究，从改进基本合并策略到利用新型硬件，以持续提升 LSM 引擎的性能和效率。已有研究工作可以大致归纳为多个方向，包括减少写放大、提升读性能、键值分离、优化压缩（合并）策略、异构硬件加速、自动参数调优以及云环境适配等。本文将按时间发展脉络梳理 LSM-tree 研究的主要演进，并结合技术维度对关键系统和方法进行分类综述。c

2 发展阶段划分与分析

LSM-tree 相关研究的发展可大致分为三个阶段：**2011–2015 年**：初期探索阶段，**2016–2019 年**：优化拓展阶段，以及 **2020–2025 年**：新兴技术阶段。下面将按时间顺序介绍各阶段的代表性系统和技术创新，并概括每一阶段的研究重点。

2.1 2011–2015 年：初期探索阶段

这一阶段以 2011 年 Google 推出的 LevelDB 和 2012 年 Facebook 在其基础上开发的 RocksDB 为标志，开启了学术界和工业界对 LSM-tree 的初步探索和优化尝试。研究重点主要围绕 LSM-tree 存在的写放大问题和读性能瓶颈展开。例如，2012 年提出的 bLSM 引入三层结构（C0-C1-C2）结合 Bloom Filter 降低读放大至接近 B-Tree 水平；设计“spring and gear”调度器，有效避免传统 LSM 合并过程导致的写停顿。同时，LevelDB 和 RocksDB 明确了以 Leveled 分层合并为主流的策略，在降低读放大的同时，也暴露出较高写放大的缺陷。这一阶段的工作为后续更深入的优化与扩展研究奠定了重要基础。

2.2 2016–2019 年：优化拓展阶段

随着 SSD 的普及，2016 年前后 LSM 研究进入了全面优化阶段，研究者开始从多维度提升 LSM-tree 性能。在降低写放大方面，Facebook 于 2016 年提出了 WiscKey，首次引入“键值分离”

思想，将大 value 从 LSM 树中剥离至顺序日志，仅在树中存储键及指针，大幅减少了写入和合并放大。随后，多项研究沿此思路继续优化 LSM 的写性能。

与此同时，读性能优化也成为研究热点。2017 年提出的 Monkey 系统通过数学模型优化 Bloom 过滤器的内存分配，有效减少查询访问的运行数量，提升了查询性能；CMU 开发的 PebblesDB 则引入分段日志结构（FLSM），通过将层级分割成多个子组件减少了单次合并的数据量，降低了写放大并维持较好的读延迟。此外，Dostoevsky 则提出自适应合并策略，通过跳过部分不必要的合并操作，更好地平衡了读写和空间成本。

2.3 2020–2025 年：新兴技术阶段

进入 2020 年代，LSM-tree 系统的研究迎来了与新硬件深度融合的阶段。随着 NVMe SSD、NVM、DPU、FPGA 和 GPU 等设备的普及，研究者开始探索通过计算卸载和近数据处理来缓解传统合并瓶颈。PinK 系统将顶部索引常驻 DRAM、移除 Bloom filter 并将 compaction 操作下沉至 SSD 控制器，有效降低了写延迟。X-Engine 基于 FPGA 构建流水线压实单元，提升了吞吐和能效；GPU 加速方案通过解析、排序、生成等任务的并行执行显著提升了压实吞吐，同时借助 GPUDirect 降低了数据传输开销。D2Comp 则利用 DPU 的硬件引擎执行压实与压缩，缓解了主机计算与网络压力。此外，SmartSSD 被用于 AegonKV 中处理 KV 分离存储中的垃圾回收任务，实现前后台读写与 GC 的并行化。这些工作展示了 LSM-tree 在多种硬件形态下实现合并卸载与性能突破的潜力，也为后续软硬协同优化打下基础。

在系统设计与应用拓展方面，LSM-tree 结构逐步适应分布式部署与云原生环境的需求。Nova-LSM 将 LSM 存储解耦为处理、日志与存储三大组件，配合动态范围划分与并行压实机制，有效提升了系统可扩展性与负载均衡能力。Calcspar 针对云存储延迟波动问题，引入 IOPS 稳压与机会性压实策略控制尾延迟；CaaS-LSM 通过将 compaction 作为服务独立部署，降低了计算节点负担并提升跨区域资源利用率。RocksMash 则采用冷热分层存储，将高频访问层驻留本地 SSD，其余数据下沉至云端，兼顾性能与成本。与此同时，LSM-tree 也向时序数据、图数据等特定负载场景演化。Lethe 系统基于 TTL 和双索引机制支持范围删除，避免空间膨胀；Poly-LSM 在图数据中融合顶点中心与边中心布局，通过查询-更新比自适应选择策略，并结合压缩编码提升查询性能与存储效率。上述研究表明，LSM 已不仅是“写优化存储结构”，而正在演变为可嵌入多场景、可调度、可弹性的核心数据引擎。

读写优化仍是该阶段 LSM-tree 演进的技术核心。为控制写放大，B LSM 构建层级化树状结构并引入“捆绑式合并”，在 Leveling 与 Tiering 之间灵活切换，提升整体效率；DiffKV 针对不同大小的 value 采用差异化存储策略，并以 vTree 管理延迟排序，从而在多类负载下同时兼顾写入、读取与扫描性能。在读路径方面，TridentKV 引入轻量级学习索引结构并设计可独立删除子树，缓解墓碑影响并提升查询效率；GRF 全局范围过滤器利用形状编码构建键位置映射，显著降低范围查询的系统开销；Disco 跨层构建紧凑索引结构，使查询在最差情况下仅需一次磁盘访问即可定位目标键或确认缺失，实测读取性能比 RocksDB 提升逾 2 倍。此外，自动化优化也成为研究重点。RusKey 借助分层强化学习在线调整合并策略并支持零延迟切换，Camal 则通过成本建模与主动学习在大参数空间中快速定位最优结构，Smoose 框架则支持层级配置独立调整，通过动态规划搜索写查查均衡解。这些成果使得 LSM 不再依赖静态策略配置，而向具备自适应调

参能力的智能结构演进。

3 关键技术与系统综述

从技术维度上进行分类, 主要分为以下几个方向：写放大优化、读性能提升、键值分离、压实策略优化、异构硬件加速、自动调优和云环境适配。每一方向均贯穿多个发展阶段，下面分别加以总结。

3.1 写放大优化

写放大是 LSM-tree 在多层合并机制下固有的问题，大量研究旨在降低合并过程中重复写入数据带来的开销。早期的 bLSM 通过调整合并调度在一定程度上减少了写停顿，但根本上的写放大问题在于重复的排序合并。PebblesDB 以碎片化日志结构减少全量合并频率，Dostoevsky 则尝试跳过部分不必要的合并操作，从宏观上减少写入冗余。更直接的途径是优化数据布局和合并策略，如 LSM-trie 通过 Trie 索引避免重复排序大量相同前缀的键，从而削减了写放大。此外，新近提出的 B^{+}LSM 和其他混合策略引擎通过按需在 leveled 和 tiered 之间切换，有效地控制了每次合并的数据量和频率。总的来说，这些方法着眼于减少无效的数据重写和减少每次合并规模两个方面，从算法设计上缓解了 LSM 的写放大，在保持高写吞吐的同时降低了后台 I/O 耗费。

3.2 读性能提升

LSM 牺牲读换取写的设计使其读取性能相对较弱，具体表现为查询可能需要在多层次、多个文件中查找目标，带来额外 I/O 和 CPU 开销。为此大量优化围绕如何减少查询需访问的 SSTable 数量以及加速键定位展开。经典的 Bloom 过滤器可以跳过不包含目标键的文件，但无法提供精确定位信息且对范围查询无效。Monkey 优化了过滤器内存分配以降低查询需检查的文件数。进一步的，Disco 索引和学习索引代表了跨层级统筹加速查找的新思路：Disco 通过在 LSM 所有已排序运行上构建紧凑 B+ 树索引，让点查找至多访问一次磁盘即可确定存在性；学习索引（如 TridentKV 中的索引块）利用数据分布的可学习模式，将查询键迅速映射到近似位置，再通过有限线性搜索找到精确位置。同时，对于范围查询，全局范围过滤器（GRF）等技术通过跟踪键空间的全局信息，能够快速判断某范围在整棵 LSM 树中是否存在，从而避免逐层扫描。另一类读优化是减少无效数据干扰，例如 Lethe 和 TridentKV 通过及时清理或隔离被删除/过期的数据，避免查询为无用数据付出代价。总体而言，读性能提升方向的研究从跳减无关数据访问和加速目标定位两方面入手，让 LSM 在保持写优势的同时，将查询性能逐步逼近传统 B+ 树甚至专用内存索引的水平。

3.3 键值分离 c（KV 分离）

键值分离是降低写放大的重要策略，也可以看作独立的研究方向。WiscKey 开创性地将大 value 存储在顺序日志中，只在 LSM 中存储小型的键和指针。这一设计显著减少了 LSM 层级合并的数据量和频率，实验表明对加载和查询性能都有数量级提升。其成功引发了后续大量工作：

DiffKV 在 WiscKey 的基础上更进一步，针对不同大小的值采用差异化的存储介质和方式，小值直接存 LSM、中等值部分排序、大值纯日志并冷热分离，从而解决了传统 KV 分离方案在扫描操作上的效率问题。此外，KV 分离也需要解决日志垃圾回收（GC）的开销问题，AegonKV 通过硬件卸载 GC 将这一瓶颈大幅削弱，实现了高吞吐、低延迟和低空间开销的统一。可以说，KV 分离已成为应对 LSM 写放大的有效途径，但其代价是引入了日志 GC 和查询时的额外一次 I/O（根据实现可能需要通过 key 指针再取 value），这一代价往往也会降低范围查询的效率，这是因为 value 与 key 物理上分离，范围扫描时需对每个 key 单独跳转读取 value，导致大量随机 I/O 访问，严重影响顺序扫描的性能。为此，研究者通过优化缓存、并行 IO 和硬件加速等手段将额外成本降至最低，例如 SpanDB 将 WAL 和顶层数据放在高速 SSD 且对冷热数据动态分层管理，实质上也是一种的 KV 分离思想应用。总体而言，KV 分离方向的研究表明，对于值较大的工作负载，将键和值解耦存储能极大提高 LSM 的效率，是平衡写入和读放大的有效手段。

3.4 压缩策略优化

LSM-tree 的压实 (compaction) 策略直接决定了读写放大的权衡和系统性能。传统 LSM 主要有两种合并策略：Leveling（每层一个排序运行，保证跨层无重叠，读性能好但写放大高）和 Tiering（每层积累多个运行再批量合并，写放小但读需要查多个文件）。如何在这两者之间找到最佳折中，是压实策略优化的核心问题。早期一些系统（如 Cassandra）通过配置混合使用 leveled 或 tiered，但缺乏理论指导。近年来，该方向的研究日趋系统化：Dayan 等人提出 LSM 设计空间探索工具 Compactionary，将压实策略分解为“何时触发合并、数据如何布局、合并粒度、数据移动策略”等原语，让研究者和工程师可以灵活组合并可视化不同策略在各种工作负载下的效果，从而指导压实算法设计。同时，学界也通过建模分析证明单一固定策略难以适应所有负载，因此出现了如前述 B+LSM 这类自适应压实方案，根据节点或层级的压力动态调整策略，在 Leveling 与 Tiering 间切换。还有工作聚焦于压实过程的粒度优化：例如“分区压实”（Partitioned compaction）思想将大的合并任务拆解为多个小范围的数据合并，以减少单次 I/O 冲击并提高并行度，这一思想已部分融入 RocksDB 等工业系统。值得一提的是，Facebook 的 RocksDB 团队也提供了多种压实选项（如基于时间次序的 FIFO 压实等）以适应特殊场景。总的来看，压实策略优化方向经历了从经验调参到自动化探索的过程，目前研究者不仅能够更全面地理解各种策略的利弊，还可以借助工具和算法在庞大的策略空间中搜索近似最优方案。这将有助于未来构建能自适应调整压实策略的 LSM 存储引擎。

3.5 异构硬件加速

随着硬件技术的发展，LSM 引擎正积极拥抱异构计算和新型介质，以突破单纯靠 CPU 处理所遇到的瓶颈。SSD 内部并行度和带宽的大幅提升，使得 LSM 的瓶颈从 I/O 逐渐转向计算和协调开销，这正是利用异构硬件的切入点。存内计算（In-storage computing）方面，PinK 通过在 SSD 控制器内实现部分 LSM 功能（如索引驻留、硬件合并），将计算推向数据所在处；SmartSSD 则为 AegonKV 的 GC 卸载提供了平台，使后台清理不再与前端竞争 CPU 资源。这些方案利用了存储设备自带的处理能力，在降低延迟的同时释放了主机负载。可编程加速卡方面，FPGA/GPU/DPU 等各展所长：FPGA 方案专注于流水线并行和定制 I/O 接口，实现了高吞吐的排序合并；GPU 方

案发挥其数据并行优势，将排序和压缩等步骤并行化，加速效果显著；DPU 方案则结合其近数据处理和网络卸载能力，在存储分离架构下大幅提升了合并和压缩效率。此外，异构硬件的引入也带来了系统架构上的新考虑，例如如何高效地在 CPU 和加速硬件之间调度任务、传输数据、保持一致性等。总体而言，异构硬件加速正成为 LSM 研究中愈发重要的一环，它充分利用硬件的并行和专用能力，使 LSM-tree 突破了以往纯软件实现的性能上限，为未来超大规模和超高性能的数据管理奠定了基础。

3.6 自动调优

LSM-tree 内部存在众多影响性能的参数（如内存大小、层级容量比、Bloom 过滤器位数、压实触发阈值等等），传统上需要根据经验或离线测试手工配置。但在工作负载动态变化、系统配置复杂化的今天，静态配置往往无法持续保持最优性能。自动调优技术因此开始引入到 LSM 系统中，旨在根据实时情况智能调整参数和策略。基于机器学习的方案是当前的研究热点之一：强化学习应用于 LSM 调优的典型是 RusKey，它将压实策略的选择建模为多层决策问题，通过试探环境反馈来不断改进策略。RusKey 实现了不同层级策略的解耦学习和推断，使 LSM 可以“在线学习”何时该采用哪种合并策略，在负载发生变化时自主切换且几乎无开销。主动学习与模型结合的 Camal 则走另一条路径：它利用少量采样和成本模型快速摸清参数对性能的影响趋势，然后有针对性地追加采样点训练模型，最终得到高精度的性能预测器用于指导参数优化。这种方法大幅减少了探索开销，并能随着系统运行不断修正模型，从而适应数据规模增长和负载变化。除了学习方法，自动调优也包括规则算法优化，如 SILT 等系统通过分析得出最优配置公式，或者基于启发式搜索在有限空间内寻找高性能配置。可以预见，自动调优方向将在未来发挥更大作用——随着硬件和应用的多样化，LSM 引擎将难以靠固定配置适配所有场景，借助机器学习和自适应机制，存储引擎将能够像数据库调优器一样动态优化自身，从而在无人值守下长时间保持高性能运行。

3.7 云环境适配

云计算环境为存储系统带来了新的机遇与挑战。一方面，弹性的计算和存储资源提供了横向扩展的可能；另一方面，网络延迟、带宽限制以及多租户干扰又给性能稳定性提出了要求。为此，近年来 LSM-tree 的研究专门针对云和分布式环境进行了优化。Nova-LSM 通过组件化拆分架构来实现水平扩展：将 LSM 的不同功能模块部署在不同服务器上，借助高速网络协作，使数据和负载可以在集群中分散处理，从而突破单节点性能上限。此设计也方便根据需要独立扩容，例如增加纯存储节点以提高容量，或者增加处理节点提高吞吐。针对云存储服务的特性，Calcspar 等引入了 QoS 和合同感知机制：通过监测 IOPS 使用情况，动态调整 LSM 内部操作的速率和优先级，避免与云存储服务的性能上限冲突。例如当后台合并可能引发突发 I/O 时，系统会主动进行限流或延迟，以平滑写入压力，降低对前台请求的影响。这种“稳压”策略有效减轻了公有云环境中常见的尾延迟飙升问题。另外，云环境下的数据容灾和成本优化也促使 LSM 进行改造，比如将底层存储格式兼容纠删编码以减少冷数据存储费用，或者结合对象存储特性调整 WAL 和检查点策略等等。总的来说，云环境适配方向的研究使 LSM-tree 更加“云原生”，通过对架构和

算法的改进，它能够更好地融入分布式和云端的生态，在保证高吞吐低延迟的同时，提供良好的扩展性、成本效益和服务质量保障。