教程：无处不在的跳表、其变体以及现代大数据系统中的应用

Venkata Sai Pavan Kumar Vadrevu、Lu Xing、Walid G. Aref 普渡大学 美国印第安纳州西拉斐特
{vvadrevu,xingl,aref}@purdue.edu

摘要

跳表（Skiplist 或 skip list）最初被设计为一种内存中的数据结构，近年来作为许多 NoSQL、基于云和大数据系统中的主内存组件，受到了广泛关注。与 B 树不同，跳表不需要复杂的再平衡机制，但仍能实现预期的对数级性能。它支持多种操作，包括插入、点读和范围查询。为了使跳表具有更强的通用性，研究人员针对其节点结构、构建算法、链表结构和并发访问等方面进行了多种优化。针对许多大数据系统场景，人们提出并试验了跳表的多种变体。

除了作为内存组件外，跳表还在系统中充当核心索引，用于解决写放大、写停顿、排序、范围查询处理等问题。本教程全面介绍跳表及其变体、优化方法，以及大数据和 NoSQL 系统中跳表的各种应用场景。在本教程中，我们将展示在现代数据系统中使用跳表或类似跳表的结构所具有的优势。

1 引言

Pugh [52], 首次提出的跳表正日益流行，由此还创造了一个新词“skiplist”。它是平衡树 [52], 的一种概率性替代方案，而且平衡起来所需的工作更少。许多系统都需要一种快速、并发的内存组件，用于维护输入数据的有序状态，而跳表正是这类系统的理想选择。跳表已应用于许多数据系统，例如 [1–4, 10, 28, 35, 70]，作为唯一或部分内存组件，实现快速并发插入，之后再与其他有序数据部分合并。跳表或类似跳表的结构也被用作系统中的独立索引，例如 [34, 38, 40, 55]。

教程概述。 我们计划安排一场时长 1.5 小时的教程，分为 5 个部分。

（1）跳表基础（10 分钟）。 我们首先介绍 Pugh [52, 54] 提出的跳表原始设计，详细剖析其结构，并简要介绍其潜在用途和优化方法。

（2）跳表与树的等价性（10 分钟）。 我们将重点介绍跳表与树结构之间的对偶关系。有趣的是，这种对偶关系使得并发控制算法可以在这两种结构之间相互映射。

（3）跳表的变体（20 分钟）。我们将概述适用于跳表的多种优化，包括自适应性、确定性、并发支持、节点结构；以及众多跳表变体，例如基于 DRAM、SSD 和 Intel Optane 持久内存的跳表、NUMA 感知跳表，以及针对特定数据类型和特定操作进行优化的跳表。

（4）现代数据系统中的跳表（40 分钟）。 我们将概述各种数据系统如何在现代数据系统中应用跳表，尤其是在主存组件以及写优化数据系统的缓冲区中。我们将概述以跳表为核心的数据系统，并重点介绍使这些系统在特定工作负载下胜过其他系统的跳表特性。

（5）跳表的其他应用场景（10 分钟）。 跳表广泛应用于其他系统领域，例如网络协议、分布式系统 [16, 43, 49, 58]、视频流 [62], 和区块链 [57]。我们将重点介绍其他基于跳表构建的系统结构，例如跳跃图 [7]。最后，我们将讨论未来研究中的开放问题和挑战。

目标读者。 我们的目标读者范围广泛，包括希望了解跳表的基本结构与演进、其潜在优化方式、跳表如何助益现代数据系统，以及基于跳表的数据存储系统的关键特性的学生、数据库研究人员和从业者。

相关教程。 这是首个关于跳表及其在现代数据系统中应用的教程。近期，其他教程介绍了趋势索引，例如针对 LSM 树的读优化 [56]以及持久内存上的索引 [36]。

2 教程大纲

2.1. 基本跳表。 跳表是由 Pugh [52]提出的一种概率数据结构。插入和点读的预期开销均为 $O(\log n)$，其中 n 是跳表中的条目数。基本上，跳表是由多层链表构成的层级结构，其中上层（$i+1$）链表中的数据是第 i层链表中数据的子集。搜索从最顶层的粗粒度层开始，跳过不必要的比较，再逐层向下，在较低层进行更精细的搜索。节点的高度由预先设定的概率 p 随机决定，该概率决定了提升到上一层的节点所占比例。人们已开发出构建最优或近乎最优跳表的算法 [50]。

2.2. 跳表与平衡树的对偶性。
在第一篇关于跳表的论文中，Pugh 提到，平衡树和跳表的性能界限处于同一数量级，但跳表比平衡树更容易实现 [52]。 [47] 展示了 2-3（2-3-4）树与 1-2（1-2-3）确定性跳表之间的对应关系。 [41] 通过添加两种 B 树结构，将确定性跳表映射为 B 树。跳树 [45] 是跳表与 B 树之间的一种中间结构，而 [20] 则描述了如何将跳表解释为随机平衡的二叉搜索树。

2.3 跳表并发控制

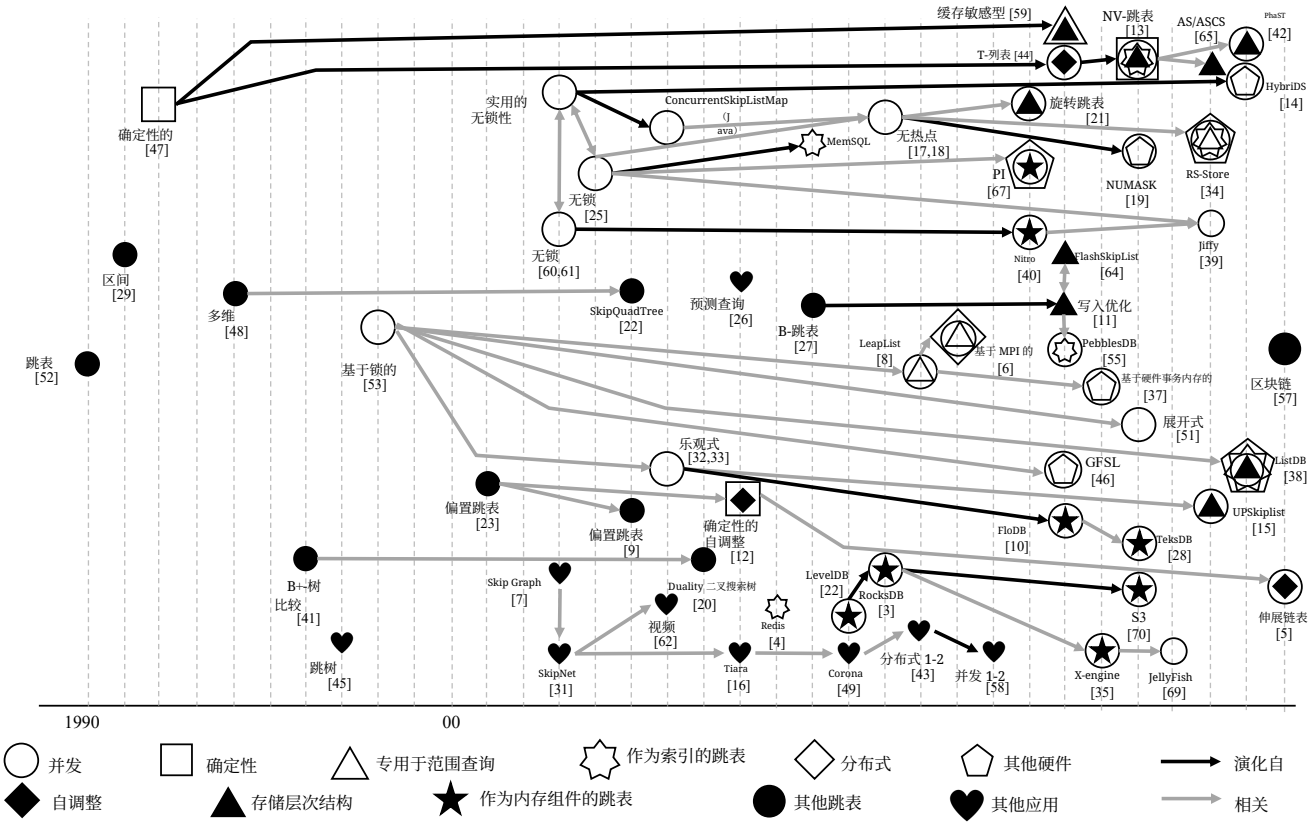

图 1：跳表的变体及其应用场景。请注意，由于所有跳表变体都演化自 1990 年的原始跳表，为了更清晰地展示，图中省略了从原始跳表指向其他所有变体的连线。

基于锁的。 Pugh 讨论了在插入或删除元素时使用写锁的并发跳表 [53]。Herlihy 等人 [32, 33] 提出了一种乐观式基于锁的跳表，该跳表要求在插入和删除操作之前进行简短的基于锁的验证。展开式跳表 [51] 在每个节点中组织多个键，并使用基于组互斥的高级锁定原语来实现并发。

非阻塞。无锁跳表使用比较并交换原语 [25, 60, 61]。由于高层被访问得更频繁，也更容易发生竞争，[17, 18] 将每次更新解耦为快速提交所需的急切抽象修改，以及在后台应用更改的选择性延迟适配。旋转跳表将树的无锁旋转与跳表相结合 [21]。

事务内存。事务内存是实现同步的另一种方式。[37]介绍了一种基于硬件事务内存（HTM）的跳表。

多版本并发控制（MVCC）。MVCC 应用于跳表。X-engine [35] 和 JellyFish [69] 的跳表均通过组织垂直列表来存储同一个键的多个版本。Jiffy [39] 维护一个版本垂直列表，用于存储底层链表中两个相邻节点之间范围内的所有数据。

2.4. 跳表变体

确定性跳表、自适应跳表和偏置跳表。 在最坏情况下，概率型跳表的性能可能无法得到上界保证。在相比之下，确定性跳表的最坏情况开销有界。1-2/1-2-3 跳表 [47] 是确定性的，在任意两个高度为 $h > 1$ 的节点之间，高度为 $h - 1$ 的节点最多有 2-3 个。[12] 的确定性跳表具有自调整特性。T-list [44] 在搜索过程中自顶向下构建跳表，作为搜索的副产品。当指针追踪次数达到预设值时，会提升一个节点，以便于后续搜索。相比之下，伸展链表 [5] 会记录每个节点的访问次数，并据此提升或降低节点。

与假设访问跳表元素时均匀分布不同，偏置跳表 [9, 23] 能够处理非均匀访问模式。

面向新型硬件的跳表。跳表针对各种存储介质而设计，例如 DRAM、SSD 和英特尔傲腾持久内存。跳表也适用于非一致性内存访问架构（NUMA）、图形处理器和近内存处理。

多种存储介质。 跳表经过改造后可适用于块设备、SSD 和持久内存（PM）。写优化跳表 [11] 在每个节点内分配缓冲区。一个节点包含多个枢轴，这些枢轴指向子层节点，而子层节点包含相同的枢轴副本。FlashSkipList [24, 64] 将缓冲区与某一层关联起来，并在满足特定条件时将其向下推送至较低层。NV-skiplist [13] 专为持久内存设计，并采用多种设计来减少持久内存写入次数。UPSkiplist [15] 使用纪元机制解决持久内存中的崩溃一致性问题。原子跳表（AS）和原子选择性一致性跳表（ASCS） [65] 是

针对持久内存进行了写入优化。PhaST [42] 是另一种用于持久内存的跳表，支持并发访问且无需日志。

NUMA感知。非一致性内存访问（NUMA）感知跳表，例如 PI/PSL [66–68] 和编织跳表 [38]，将跳表划分为多个范围，并分配给不同的 NUMA节点。NUMASK [19] 则将上层放置在不同的 NUMA节点中。

图形处理器、RDMA 和近内存处理。GFSL [46] 是一种基于图形处理器的跳表。R-skiplist [34] 是一种基于 RDMA 的跳表。混合跳表 [14] 则用于近内存处理（NMP）。

其他跳表变体。针对跳表的范围查询，还有更多优化方法，[6, 8, 59] 包括利用缓存敏感性、消息传递接口（MPI）或软件事务内存。跳表还可以为其他数据类型建立索引，例如多维数据 [22, 48] 和区间 [29, 30]。跳表也用于处理预测查询 [26]。

2.5. 跳表在大数据系统中的应用

跳表作为内存组件。通常，基于磁盘的键值存储会使用内存缓冲区来辅助数据写入。内存缓冲区需要具备很高的并发性，并且易于保持有序，因此跳表是内存缓冲区的理想选择。许多键值存储采用包含内存组件的日志结构合并树（LSM树）。LevelDB [2] 使用跳表来组织内存中的数据。RocksDB 的内存缓冲区采用跳表 [3]，，并提供哈希跳表变体，通过跳表链接哈希桶。该变体在点读时比较次数更少，但为了在范围查询中保持数据有序，需要承担更多开销。X-engine [35] 支持多版本跳表。Redis [4] 在其有序集合中使用跳表来提升搜索效率。HBase [1] 也使用跳表来存储内存中的数据。

跳表 + X。为了让 LSM键值存储能够随主存容量增长而扩展，人们引入了辅助结构，与跳表协同工作 [10, 28, 70]。S3 [70] 采用双层内存结构。底层是半有序跳表：键无序，但所有节点的整体范围有序。为了提升搜索效率，顶层的守卫键由神经网络模型选取，以跳过对跳表节点的遍历。缓存敏感索引用于组织顶层的守卫键。FloDB [10] 采用两级内存结构：顶层是小型并发哈希表，底层是大型并发跳表，适合并行扫描和写入，并支持低延迟更新。TeksDB [28] 的主存中包含两种互补结构：布谷鸟哈希和跳表，分别用于快速点读和范围查询。

基于跳表的系统。除了作为内存组件外，跳表也广泛用作索引。Nitro [40]使用无锁跳表作为其核心索引。多版本并发控制、垃圾回收、快照管理器和恢复管理器等其他组件则构建在其之上。还有其他系统依赖跳表的关键特性，例如保持平衡、支持原地插入、维持数据有序以及支持范围查询。

平衡结构。LSM树的一个缺点是写放大较高。数据在到达底层之前可能会被写入多次，从而浪费写入I/O带宽。为缓解这一问题，LSM树像B/B+-树一样将数据划分为多个范围。然而，维护树结构并非易事。PebblesDB [55]将跳表与LSM树结合，构建了碎片化LSM树（FLSM）。每个FLSM层都使用从插入键中选取的守卫键来划分该层的键空间。每个守卫键都关联着若干SSTable。守卫键的选取方式与跳表类似，均从插入的键中选取。

原地有序插入。LSM树可能会出现写停顿。数据在刷新到磁盘之前会先累积在 DRAM 中。如果排序和压缩速度较慢，且写缓冲区无法按需刷新，传入的写入操作可能会被阻塞，从而导致高延迟。跳表通过在允许快速原地插入的同时维护有序状态来解决这一问题。ListDB [38] 在持久内存上采用了 NUMA感知跳表，以实现基于跳表的键值存储。ListDB 是一个三级 LSM树，其中 DRAM 中包含一个易失层，PM 中包含 L0 和 L1 两层。ListDB 通过消除 MemTable 刷新以及持久数据归并排序过程中的写入操作，降低了写放大。在向 DRAM 的 MemTable 插入数据时，相同的数据及其（元数据）会作为日志写入 PM。随后，这条带有预分配跳表指针的日志条目会作为跳表节点插入 L0。在 L0-L1 归并排序时，数据通过原地合并完成，且仅需修改跳表指针。

范围查询处理。基于哈希的键值存储在 RDMA 上大多只支持点查询。跳表支持范围查询且并发能力很强，因此非常适合用作基于 RDMA 的键值存储。RS-store [34] 是一个使用 RDMA 的基于跳表的键值存储。其 R-skiplist 将多个条目分组到一个块中。R-skiplist 通过分区来提高可扩展性，每个分区由一个专属线程负责。如果由于服务器 CPU 利用率过高而无法处理客户端发出的请求，客户端就会通过 RDMA 访问 R-skiplist，而无需占用服务器 CPU。此类远程访问为只读操作，并且远程范围扫描不会大量占用服务器 CPU：服务器只需返回一次块的起始地址，客户端便可发起远程 RDMA 读取，以获取后续块。

其他应用 跳表广泛应用于点对点视频流网络协议、历史无关文件系统和分布式系统。基于跳表的 Skip Graph 广泛应用于分布式环境中。跳表也应用于区块链。

3 作者简介

VS Pavan Kumar Vadrevu 是普渡大学计算机科学专业的硕士生。他的研究兴趣包括大数据系统、数据库和人工智能。

Lu Xing 是普渡大学计算机科学专业的博士生，导师为 Walid G. Aref。她的研究兴趣包括索引优化和自适应性。

Walid G. Aref 是普渡大学计算机科学教授。他的研究兴趣包括空间、时空、图和传感器数据管理。他是《ACM 空间算法与系统汇刊》（ACM TSAS）的主编。

致谢

Walid G. Aref 感谢美国国家科学基金会通过资助项目（编号：IIS-1910216）提供的支持。

参考文献

[1] 访问日期：2023-04-01。Apache HBase。https://hbase.apache.org/ [2] 访问日期：2023-04-01。Google。https://github.com/google/leveldb [3] 访问日期：2023-04-01。Meta。https://github.com/facebook/rocksdb [4] 访问日期：2023-04-01。Redis。https://redis.io/ [5] V. Aksenov、D. Alistarh、A. Drozdova 和 A. Mohtashami。2023。伸展链表：一种分布自适应的并发跳表。分布式计算 （2023），1–24。

[6] S. Alam、H. Kamal 和 A. Wagner。2014。一种用于范围查询的可扩展分布式跳表。收录于 HPDC。315–318。[7] J. Aspnes 和 G. Shah。2007。跳跃图。《算法学报》（TALG）3，4（2007），37–es。[8] H. Avni、N. Shavit 和 A. Suissa。2013。LeapList：设计支持事务内存的范围查询所获得的经验。收录于 PODC。299–308。[9] A. Bagchi、A. L. Buchsbaum 和 M. T. Goodrich。2005。偏置跳表。《Algorithmica》42（2005），31–48。[10] O. Balmau、R. Guerraoui、V. Trigonakis 和 I. Zablotchi。2017。FloDB：释放持久性键值存储中的内存。收录于 EUROSYS。80–94。[11] M. A. Bender、M. Farach-Colton、R. Johnson、S. Mauras、T. Mayer、C. A. Phillips 和 H. Xu。2017。写优化跳表。收录于 PODS。69–78。[12] P. Bose、K. Douïeb 和 S. Langerman。2008。跳表和 B 树的动态最优性。收录于 SODA。Citeseer，1106–1114。[13] Q. Chen、H. Lee、Y. Kim、H. Y. Yeom 和 Y. Son。2019。非易失性内存中基于跳表的键值存储的设计与实现。《Cluster Computing》22（2019），361–371。[14] J. Choe、A. Crotty、T. Moreshet、M. Herlihy 和 R. I. Bahar。2022。HybriDS：面向近内存处理架构、缓存感知的并发数据结构。收录于 SPAA。321–332。[15] S. Chowdhury 和 W. Golab。2021。一种适用于持久内存的可扩展可恢复跳表。收录于 SPAA。426–428。[16] T. Clouser、M. Nesterenko 和 C. Scheideler。2008。Tiara：一种自稳定确定性跳表。收录于 SSS 2008（10）。Springer，124–140。[17] T. Crain、V. Gramoli 和 M. Raynal。2012。一种便于处理竞争的非阻塞跳表。博士论文。INRIA。[18] T. Crain、V. Gramoli 和 M. Raynal。2013。一种无热点的非阻塞跳表。收录于 ICDCS。IEEE，196–205。[19] H. Daly、A. Hassan、M. F. Spear 和 R. Palmieri。2018。NUMASK：适用于 NUMA 的高性能可扩展跳表。收录于 DISC 2018。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik。[20] B. C. Dean 和 Z. H. Jones。2007。探索跳表与二叉搜索树之间的对偶性。收录于 ACM-SE。395–399。[21] I. Dick、A. Fekete 和 V. Gramoli。2017。一种适用于多核的跳表。《Concurrency and Computation: Practice and Experience》29，4（2017），e3876。[22] D. Eppstein、M. T. Goodrich 和 J. Z. Sun。2005。跳跃四叉树：一种适用于多维数据的简单动态数据结构。收录于 SOCG。296–305。[23] F. Ergun、S. Cenk Şahinalp、J. Sharp 和 R. K. Sinha。2001。适用于高度倾斜访问模式的偏置跳表。收录于 ALENEX 2001 修订论文集 3。216–229。[24] A. Fevgas、L. Akritidis、P. Bozanis 和 Y. Manolopoulos。2020。闪存设备中的索引：挑战、当前方法与未来趋势综述。《The VLDB Journal》29（2020），273–311。[25] M. Fomitchev 和 E. Ruppert。2004。无锁链表和跳表。收录于 PODC。50–59。[26] T. Ge 和 S. Zdonik。2008。一种高效处理预测查询的跳表方法。《VLDB》1，1（2008），984–995。[27] D. Golovin。2010。B 跳表：一种更简单、具有唯一表示形式的 B 树替代方案。arXiv 预印本 arXiv:1005.0662（2010）。[28] Y. Han、B. S. Kim、J. Yeon、S. Lee 和 E. Lee。2019。TeksDB：为高性能键值存储编织数据结构。《ACM 计算系统测量与分析学报》3，1（2019），1–23。[29] E. N. Hanson。1991。区间跳表：一种查找与某点重叠的所有区间的数据结构。收录于 WADS。Springer，153–164。[30] E. N. Hanson 和 T. Johnson。1996。使用区间跳表为活动数据库中的选择谓词建立索引。《Information Systems》21，3（1996），269–298。[31] N. J. Harvey、J. Dunagan、M. Jones、S. Saroiu、M. Theimer 和 A. Wolman。2002。SkipNet：一种具有实用局部性特征的可扩展覆盖网络。（2002）。[32] M. Herlihy、Y. Lev、V. Luchangco 和 N. Shavit。2006。一种可证明正确的可扩展并发跳表。收录于 OPODIS。Citeseer。103。[33] M. Herlihy、Y. Lev、V. Luchangco 和 N. Shavit。2007。一种简单的乐观式跳表算法。收录于 SIROCCO。Proceedings 14。Springer，124–138。[34] C. Huang、H. Hu、X. Qi、X. Zhou 和 A. Zhou。2021。RS-Store：基于 RDMA 的跳表键值存储，可高效执行范围查询。《Frontiers of Computer Science》15，6（2021），156617。[35] G. Huang、X. Cheng、J. Wang、Y. Wang、D. He、T. Zhang、F. Li、S. Wang、W. Cao 和 Q. Li。2019。X-engine：用于大规模电子商务事务处理的优化存储引擎。收录于 SIGMOD。651–665。[36] K. Huang、Y. He 和 T. Wang。2022。持久内存索引的过去、现在与未来。《VLDB》15，12（2022），3774–3777。[37] M. Kardaras、D. Siakavaras、K. Nikas、G. Goumas 和 N. Koziris。 [n.d.]。快速并发跳表与 HTM。([n. d.])。[38] W. Kim、C. Park、D. Kim、H. Park、Y.-r. Choi、A. Sussman 和 B. Nam。2022。ListDB：结合预写日志与持久跳表，实现持久内存上的增量检查点。收录于 OSDI 22。161–177。[39] T. Kobus、M. Kokociński 和 P. T. Wojciechowski。2022。Jiffy：一种支持批量更新和快照的无锁跳表。收录于 PPOPP。400–415。[40] S. Lakshman、S. Melkote、J. Liang 和 R. Mayuram。2016。Nitro：一种快速、可扩展的内存内存储引擎，用于 NoSQL 全局二级索引。《VLDB》9，13（2016），1413–1424。[41] M. Lamoureux, B. Nickerson 等。1996。《B 树与确定性跳表的等价性》。（1996）。[42] Z. Li、B. Jiao、S. He 和 W. Yu。2022。《PhaST：面向持久内存的分层并发无日志跳表》。IEEE TPDS 33, 12（2022），3929–3941。[43] S. Mandal、S. Chakraborty 和 S. Karmakar。2012。《分布式系统中的确定性 1–2 跳表》。载于 IEEE PDGC。IEEE，296–301。[44] F. Mei、Q. Cao、F. Wu 和 H. Li。2017。《一种基于搜索进行平衡的并发跳表》。载于 APPT 2017，Proceedings 12。Springer，117–128。[45] X. Messeguer。1997。《跳树：一种适用于并发方法的跳表替代数据结构》。Info. Théor. et Appl. 31, 3（1997），251–269。[46] N. Moscovici、N. Cohen 和 E. Petrank。2017。《适用于 GPU 的跳表算法》。载于 PACT。IEEE，246–259。[47] J. I. Munro、T. Papadakis 和 R. Sedgewick。1992。《确定性跳表》。载于第 3 届 ACM-SIAM 离散算法年度研讨会论文集。367–375。[48] B. G. Nickerson。1998。《多维数据的跳表数据结构》。技术报告。[49] R. M. Nor、M. Nesterenko 和 C. Scheideler。2013。《Corona：一种稳定的确定性消息传递跳表》。Theoretical Comp. Sc. 512（2013），119–129。[50] C. Parra 和 S. Ferret。1995。《最优和近乎最优的静态加权跳表》。https://books.google.com/books?id=2ztvMwEACAAJ [51] K. Platz、N. Mittal 和 S. Venkatesan。2019。《并发展开跳表》。载于 ICDCS。IEEE，1579–1589。[52] W. Pugh。1990。《跳表：平衡树的一种概率性替代方案》。Commun. ACM 33, 6（1990），668–676。[53] W. Pugh。1998。《跳表的并发维护》。技术报告。[54] W. Pugh。1998。《跳表指南》。技术报告。[55] P. Raju、R. Kadekodi、V. Chidambaram 和 I. Abraham。2017。《PebblesDB：使用分段式日志结构合并树构建键值存储》。载于 SOSP。497–514。[56] S. Sarkar、N. Dayan 和 M. Athanassoulis。2023。《LSM 设计空间及其读取优化》。（2023）。[57] B. C. Singh、Q. Ye、H. Hu 和 B. Xiao。2023。《区块链中多维历史数据的高效轻量级索引方法》。Future Generation Computer Systems 139（2023），210–223。[58] R. Singh、S. Chakraborty 和 S. Karmakar。2015。《分布式消息传递系统中的并发确定性 1–2 跳表》。International Journal of Parallel, Emergent and Distributed Systems 30, 2（2015），135–174。[59] S. Sprenger、S. Zeuch 和 U. Leser。2017。《缓存敏感跳表：现代 CPU 上的高效范围查询》。载于 IMDM。Springer，1–17。[60] H. Sundell 和 P. Tsigas。2004。《可扩展的无锁并发字典》。载于 SAC。1438–1445。[61] H. Sundell 和 P. Tsigas。2005。《适用于多线程系统的快速无锁并发优先队列》。J. Parallel and Distrib. Comput. 65, 5（2005），609–627。[62] D. Wang 和 J. Liu。2006。《使用跳表实现点对点异步视频流》。载于 ICME。IEEE，1397–1400。[63] D. Wang 和 J. Liu。2008。《一种支持录像机交互的按需媒体流动态跳表覆盖网络》。IEEE TPDS 19, 4（2008），503–514。[64] H. Wang 和 J. Feng。2017。《FlashSkipList：闪存设备上的索引》。载于 ACM 图灵奖 50 周年中国纪念大会论文集。1–10。[65] R. Xiao、D. Feng、Y. Hu、F. Wang、X. Wei、X. Zou 和 M. Lei。2021。《面向非易失性内存的写入优化一致性跳表》。IEEE Access 9（2021），69850–69859。[66] Z. Xie、Q. Cai、G. Chen、R. Mao 和 M. Zhang。2018。《现代内存索引的全面性能评估》。载于 ICDE。IEEE，641–652。[67] Z. Xie、Q. Cai、H. Jagadish、B. C. Ooi 和 W.-F. Wong。2016。《PI：并行内存跳表索引》。arXiv 预印本 arXiv:1601.00159（2016）。[68] Z. Xie、Q. Cai、H. Jagadish、B. C. Ooi 和 W.-F. Wong。2017。《面向内存多核数据库系统的跳表并行化》。载于 ICDE。IEEE，119–122。[69] J. Yeon、L. Kim、Y. Han、H. G. Lee、E. Lee 和 B. S. Kim。2020。《JellyFish：一种支持多版本并发控制的快速跳表》。载于 Middleware。134–148。[70] J. Zhang、S. Wu、Z. Tan、G. Chen、Z. Cheng、W. Cao、Y. Gao 和 X. Feng。2019。《S3：一种面向键值存储的可扩展内存跳表索引》。VLDB 12, 12（2019），2183–2194。