软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006168]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

学习索引: 现状与研究展望*

张 洲[1,2], 金培权[1,2], 谢希科[1]

[1](中国科学技术大学 计算机科学与技术学院,安徽 合肥 230026)

[2](中国科学院电磁空间信息重点实验室,安徽 合肥 230026)

通讯作者: 金培权, E-mail: jpq@ustc.edu.cn

摘 要: 索引是数据库系统中用于提升数据存取性能的主要技术之一.在大数据时代,随着数据量的不断增长,传统索引(如 B+树)的问题日益突出:(1)空间代价过高.例如,B+树索引需要借助 O(*n*)规模的额外空间来索引原始的数据,这对于大数据环境而言是难以容忍的.(2)每次查询需要多次的间接搜索.例如,B+树中的每次查询都需要访问从树根到叶节点路径上的所有节点,这使得 B+树的查找性能受限于数据规模.自 2018 年来,人工智能与数据库领域的结合催生了"学习索引"这一新的研究方向.学习索引利用机器学习技术学习数据分布和查询负载特征,并用基于数据分布拟合函数的直接式查找代替传统的间接式索引查找,从而降低索引的空间代价并提升查询性能.首先对学习索引技术的现有工作进行了系统梳理和分类;然后,介绍了各种学习索引技术的研究动机与关键技术,对比分析了各种索引结构的优劣;最后,对学习索引的未来研究方向进行了展望.

关键词: 数据库系统;索引;机器学习;数据驱动

中图法分类号: TP311

中文引用格式: 张洲,金培权,谢希科.学习索引:现状与研究展望.软件学报. http://www.jos.org.cn/1000-9825/6168.htm

英文引用格式: Zhang Z, Jin PQ, Xie XK. Learned indexes: current situations and research prospects. Ruan Jian Xue Bao/ Journal of Software, (in Chinese). http://www.jos.org.cn/1000-9825/6168.htm

Learned Indexes: Current Situations and Research Prospects

ZHANG Zhou[1,2], JIN Pei-Quan[1,2], XIE Xi-Ke[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

[2](Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences, Hefei 230026, China)

Abstract: Index is one of the key technolgies to improve the performance of database systems. In the era of big data, the traditional indexes, such as B+-Tree, have exposed some limitations. Firstly, they cost too much space. For example, B+-Tree requires an extra O(*n*) space, which is intolerable for big data environment. Secondly, they require multiple indirect searches per query. For example, each query in a B+-Tree requires access to all nodes from the root to the leaf, which limits the search performance of the B+-Tree to the data size. Since 2018, the combination of artificial intelligence and database has given birth to a new research direction called "learned index". Learned indexes use machine learning to learn data distribution and query load characteristics, and replace the traditional indirect index search with a direct search based on fitting functions, so as to reduce the space cost and improve the query performance. This survey firstly systematically sorts out and classifies the existing works of learned indexes. Then, we introduce the motivation and key techniques of each learned index, compare and analyze the advantages and disadvantages of various index structures. Finally, the future research directions of learned indexes are prospected.

Key words: database systems; indexes; machine learning; data-driven

* 基金项目: 国家自然科学基金(62072419, 61672479)
Foundation item: National Natural Science Foundation of China (62072419, 61672479)
收稿时间: 2020-07-27; 修改时间: 2020-10-09; 采用时间: 2020-11-02; jos 在线出版时间: 2020-12-02

索引是数据库系统中用于提升数据存取性能的主要技术之一.传统数据库系统一般使用磁盘作为持久性存储设备.但由于磁盘的访问延迟高,磁盘 I/O 次数成为影响数据库系统性能的关键指标.如何快速地在磁盘中找到数据,成为数据库系统中的重要问题.自上世纪 70 年代以来,数据库领域的研究人员提出了各种各样的索引结构,用于满足不同数据类型的存取性能需求.例如,著名的 B+树索引结构[1]提供了 O($log_m n$)的查询时间复杂度(m 为一个树节点的容量)[2].通常情况下,B+树索引的 I/O 次数为 3 至 4 次,远远优于无索引的搜索算法(例如二分搜索).而且,由于数据在 B+树的叶节点中有序存储,它也可以支持高效的范围查询.

传统的 B+树索引是基于磁盘而设计的,即假设整个索引完全存储在磁盘中.为了提升数据库系统的读写性能,现代数据库系统倾向于将整个索引缓存在内存中.当数据库规模较小时,缓存索引是可行的.然而,在大数据时代,数据量可能达到 PB 级甚至百 PB 级,例如,物联网中的高频传感器采集数据、大型互联网交易平台的日志数据等每天都会产生大量的新数据,使得数据库规模极速增长.这种情况下,传统的 B+树索引因其具有 O(n)的空间复杂度将无法全部缓存在内存,进而影响查询性能.虽然近年来研究者提出了内存数据库(main memory database)[3]来支持大数据的实时存取,但内存数据库的索引结构[4,5]也同样具有 O(n)的空间复杂度.研究表明,在商用内存数据库中,索引占用了全部内存空间的 55%,严重侵占了数据库系统的内存资源[6].总体而言,传统索引在大数据环境下呈现出两个主要问题:(1)空间代价过高.例如,B+树索引需要借助 O(n)规模的额外空间来索引原始的数据,这对于大数据环境而言是难以容忍的.(2)每次查询需要多次的间接搜索.例如,B+树中的每次查询都需要访问从树根到叶节点路径上的所有节点.这使得 B+树的查找性能在大规模数据集上将变得较差.

近年来,人工智能、机器学习技术的快速发展引起了数据库领域的重视.如何借助人工智能、机器学习技术来优化数据库索引等传统数据库技术,成为当前数据库领域关注的热点问题.MIT 的研究人员 Kraska 等人在 2018 年 SIGMOD 会议上首次提出了学习索引(learned index)[7]的概念.他们将机器学习方法应用于索引结构中,目的是降低索引的空间代价,同时提升索引的查询性能.传统的 B+树索引使用简单的 if 语句递归地划分空间,这一方法没有利用数据的分布规律,具有次优的空间代价和查询性能;而学习索引则使用机器学习模型替代传统的索引结构,可以大幅降低传统索引的空间代价并提高查询性能.

从目前的发展趋势看,学习索引因其理论上的优势,有望成为索引技术的下一代研究方向.近年来国际数据库界在这一方向上开展了大量研究,并取得了一系列成果,例如 SIGMOD 2020 就收录了 7 篇学习索引相关论文.但是,目前还没有专门的综述文章对学习索引这一研究方向进行系统总结.本文对学习索引的现有研究工作进行了系统梳理和分类,介绍并对比了各种类型的学习索引技术,并展望了未来研究方向,以期为相关的研究人员提供一定的参考.

与相关综述性文章的区别:

(1) 学习索引是当前国内外较新的一个研究方向,目前还没有专门针对学习索引的综述文章.本文是第一篇系统总结学习索引研究进展的中文综述.自 2019 年以来的一些综述文章关注于机器学习化数据库技术[8-11],但它们只简要介绍了学习索引的概念,没有专门总结该方向的最新进展和发展趋势.需要指出的是,学习索引的研究进展主要集中在近两年,本文涵盖了包含 SIGMOD 2020 会议在内的最新相关工作,提供了该方向较全面的研究工作总结和展望.

(2) 提出了一个新的学习索引研究分类框架.该框架从范围查询、点查询、存在查询、测试基准等方面对当前研究进行了分类和总结,这一分类框架可以为后续研究提供参考.

(3) 讨论了学习索引的未来发展趋势,给出了 7 个值得研究的未来方向,并进行了讨论.这些方向基于我们对现有工作的深入思考,对相关研究人员有一定的参考价值.

本文的组织结构: 前四节分别介绍学习索引在面向一维范围查询的索引(第 1 节)、面向多维范围查询的索引(第 2 节)、面向点查询的索引(第 3 节)和面向存在查询的索引(第 4 节)中的相关工作,第 5 节讨论了学习索引的测试基准,第 6 节对学习索引的未来研究方向进行了展望,最后,第 7 节总结了全文.

1 面向一维范围查询的学习索引

面向一维范围查询的索引结构,需要高效地支持插入、删除、更新、点查询和范围查询等操作.这一类型的索引结构应用最为普遍,被大多数通用型数据库采纳为默认索引.常见的面向一维范围查询的索引结构有 B+树和日志结构合并(log-structured merge,简称 LSM)树[12].目前,使用机器学习模型替代这一类型的索引结构吸引了较多的关注,因此本节首先讨论面向一维范围查询的学习索引方面的研究进展.

在 1.1 节中,我们首先介绍学习索引的基本思想和面向主索引的学习索引及相关的优化技术.之后,1.2 节介绍了面向二级索引的学习索引.1.3 节对比分析了已有的学习索引技术差异.最后,在 1.4 节中我们结合实验结果对比了传统 B+树和学习索引,分析了学习索引的优势和存在的不足.

1.1 面向主索引的学习索引

为了能够高效地支持范围查询操作,面向一维范围查询的索引通常要求底层数据按照查找键有序存放.这类索引在传统关系数据库中称为主索引(primary index).传统 DBMS 默认会在主码上创建主索引.由于大多数数据库应用都会频繁执行基于主码的查询,因此主索引的优化对于提升数据库系统的查询性能有着重要的价值.因此,早期的学习索引研究工作也大都集中在主索引上.

(1) 初次尝试

B+树是传统数据库系统中常见的一维索引结构.B+树可以看作是一个模型,它的输入是一个键,输出是该键对应的记录在排序数组中的位置.B+树不会索引排序记录的每个键,而只索引一个内存块或磁盘页中的第一个键.这有助于显著减少索引存储的键的数量,而不会对索引性能造成很大的影响.这一特性让 B+树和大多数机器学习模型看起来很相似.可以将 B+树看作一个回归树(regression tree),它将一个键映射到一个具有最大误差界限(error bound)(即内存块或磁盘页大小)的数组位置,并保证如果该键对应的记录存在,就可以在误差范围内找到它[7].因此,用其他机器学习模型替代 B+树是可行的,只要这些模型也能够提供类似的保证.显然,所有的回归模型都能够提供类似的保证,包括线性回归模型和神经网络模型.

首先考虑一个理想情况下的示例.假设数据库中共有 1000 个记录,它们使用整数型的键,分别对应数字 1 至 1000.在这种情况下,我们使用线性回归模型替代 B+树索引,只需要保存模型的参数(即斜率和截距),具有 O(1)的空间复杂度.由于模型能够准确地预测出记录位置,该模型的查询的时间复杂度也是 O(1).此外,如果后续插入的数据仍然满足这一规律,模型不需要更新就可以继续为新插入的数据提供索引支持.综上所述,对于理想的情况,机器学习模型在空间代价、查询性能和索引更新代价上全方面优于传统的 B+树.然而,在大多数应用中,数据的分布不是严格遵循某种规律的,数据分布的规律也不会如此简单.总的来说,面向一维范围查询的学习索引的设计目标如下:在一个排序数组上,使用一个或一组模型有效地近似累积分布函数(cumulative distribution function,简称 CDF),使用该模型可以高效地预测出给定键在数组中的位置.

Kraska 等人基于上述思想进行了学习索引的初次尝试.他们使用 Tensorflow[13]在一个 web 服务器日志记录数据集上构建了一个主索引[7],然后使用 ReLU(rectified linear unit)激活函数训练了一个每层 32 个神经元的双层全连接神经网络,并以时间戳作为输入预测记录在排序数组中的位置.然而,实验结果表明,该模型的平均执行时间高达 80 微秒,远远差于 B+树的 300 纳秒和二分搜索的 900 纳秒的搜索时间.总的来说,神经网络模型的低效率是由三方面原因造成的:

1) Tensorflow 的设计目标是高效地运行大规模的模型,而非小模型.并且,使用 Tensorflow 需要承担一个较大的调用开销.
2) 由于 B+树使用简单的 if 语句递归地划分空间,所以它可以覆盖所有的数据,能够为面向全体数据的查找提供固定的误差范围,而不必关系数据的具体分布特点.相比之下,机器学习模型只能保证当拟合 CDF 曲线较好时取得较准确的查询结果,但机器学习的预测难以做到百分之百的精确,这是因为现实世界的数据遵循统计学中著名的固定效应(fixed effect)和随机效应(random effect)[14,15].因此,像神经网络、多项式回归等模型能够高效地将位置确定到数千的范围,却难以精确到单个记录.

3) B+树的搜索计算代价较低,而神经网络模型的计算需要多次乘法操作,具有较高的计算代价.

为了解决上述问题,Kraska 等人[7]提出了学习索引框架(learning index framework,简称 LIF)和递归模型索引(recursive model index,简称 RMI).LIF 是一个索引合成系统,用于生成索引配置、自动地优化和测试索引.LIF 能够学习简单的模型(例如线性回归模型),并依赖 Tensorflow 训练复杂的模型(例如神经网络模型).但是,LIF 在执行模型时不使用 Tensorflow,而是自动提取模型的权重并生成高效的 C++索引代码.LIF 的代码生成是专为小模型设计的,它消除了 Tensorflow 中管理大模型的所有不必要的开销和工具[16].实验结果表明,LIF 执行简单模型只需要 30 纳秒.

RMI 旨在解决学习索引的精度问题,它受到了混合专家模型[17]的启发.RMI 的结构(如图 1 所示)是由多个模型组成的分层模型结构.其中,第 1 层只有一个模型,其余每一层都包含多个模型.RMI 中的每一个模型都以键作为输入,并返回一个位置.上层模型的输出结果用于选择下一层的模型,最后一层模型的输出作为 RMI 的输出.RMI 的分层模型结构有以下优点:

1) 与使用一个复杂的大模型相比,RMI 的执行成本更低.与 B+树相比,上层模型的输出直接用于选择下层模型,不需要执行额外的搜索.
2) 它利用了之前实验得到的结论,即机器学习模型可以拟合 CDF 的大体形状,这是 RMI 中第 1 层模型的作用.由于建立在整个数据集上的单一模型难以精确定位到单条记录,RMI 将数据集划分为更小的子空间,并在每个子空间上训练模型,从而提高查找精度.
3) RMI 允许构建混合使用多种模型,因此能够充分利用不同模型的优点.对于第 1 层模型,使用神经网络是一种较好的选择,因为神经网络通常能够学习复杂的数据分布.RMI 的底层可以考虑线性回归模型,因为线性回归模型具有更低的空间代价和执行成本.如果数据分布难以拟合为某种模型,叶节点允许退化为 B+树节点,这保证了学习索引的最差查找性能与 B+树相当.

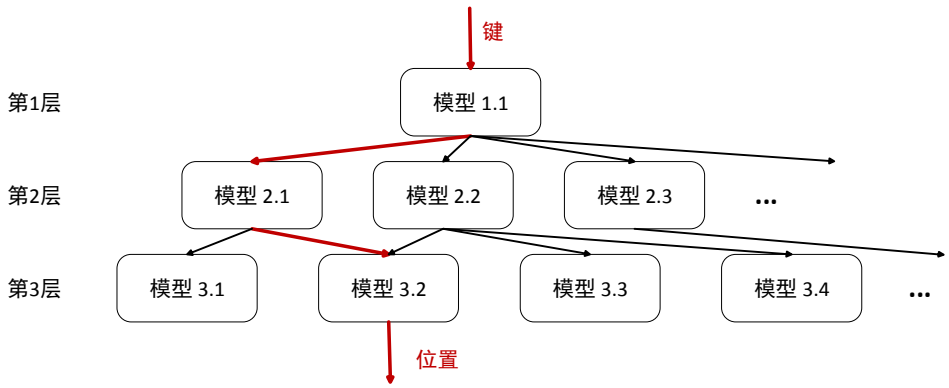

Fig.1 Layered models in RMI

图 1 RMI 中的分层模型

在 RMI 输出预测的位置后,还需要在排序数组中进行最后的搜索.RMI 的最底层的每个模型在训练时会保存一个误差界限,后续的搜索策略使用预测值作为中心点,在误差范围内执行二分搜索. Kraska 等人报告的实验结果[7]令人振奋:在多个数据集上,学习索引的查找性能比 B+树快 1.5 倍至 3 倍,并且仅占用了比 B+树低 2 个数量级的空间.

Kraska 等人提出的学习索引虽然具有较好的查找性能和较低的空间代价,但它也存在一些不足:

缺陷 1: 这类学习索引不支持插入操作,因为插入数据可能会违反模型预先保存的误差阈值;同时,大量的新数据会改变键的分布,并导致模型过时,最终降低模型在查找时的预测精度.

缺陷 2: 在 Kraska 等人提出的学习索引中,一个最底层模型覆盖的数据集空间比 B+树的一个叶节点覆盖的数据集空间更大,因此,如果模型的最大误差较大,将导致在数据集上执行二分搜索的代价变高.

缺陷 3: 这类学习索引要求底层数据按照查找键有序存储,因此只能用作主索引.由于主索引在一个表上只能存在一个,因此如果需要在非码属性上建立多个学习索引,则无法实现.而且,在非码属性上建立学习索引时也需要对数据进行排序,带来额外的排序、指针操作等代价.

(2) 进一步的优化策略

针对上述缺陷,近两年来研究人员提出了一系列的优化方法.这些优化方法大致可分类两类:一是优化面向主索引的学习索引,使其能够克服缺陷 1 和缺陷 2;二是提出面向二级索引的学习索引.下面简要介绍第一类面向主索引的学习索引优化策略,第二类面向二级索引的学习索引将在 1.2 节中详细讨论.

1) 基于缓冲区的插入策略与固定大小的模型误差. FITing-Tree[18]使用分段线性函数(piece-wise linear function)拟合数据分布,其索引结构如图 2 所示.FITing-Tree 将已排序的数据按范围划分成多个段(segment),每个段使用一个线性回归模型进行拟合.FITing-Tree 的结构与传统的 B+树十分相似,不同的是,它的叶节点存储每个段的起始键和斜率.与初版学习索引不同的是,FITing-Tree 设计了分段策略,它使用一个预先设置的误差阈值作为参数,在分段时保证每个段中的所有数据都能落在模型预测值的误差范围内;如果不能满足给定的误差阈值,则启用一个新的段.通过分段策略,FITing-Tree 可以限制最坏情况下的查找性能.在对数据分段时,FITing-Tree 提出了使用类似 FSW(feasible space window)方法[19,20]的贪心算法,具有 O(n)的时间复杂度和 O(1)的空间代价.此外,该索引使用异地插入(out-of-place insertion)策略,在每个段内部预留一个固定大小的缓冲区,在缓冲区满之后将缓冲区数据与段数据合并,重新执行分段算法.实验表明,FITing-Tree 只需要比 B+树低 4 个数量级的空间代价,就能够达到与 B+树相近的查找性能.此外,FITing-Tree 具有优于 B+树的构造速度和略差于 B+树的插入性能(假定 B+树同样使用基于缓冲区的插入策略).

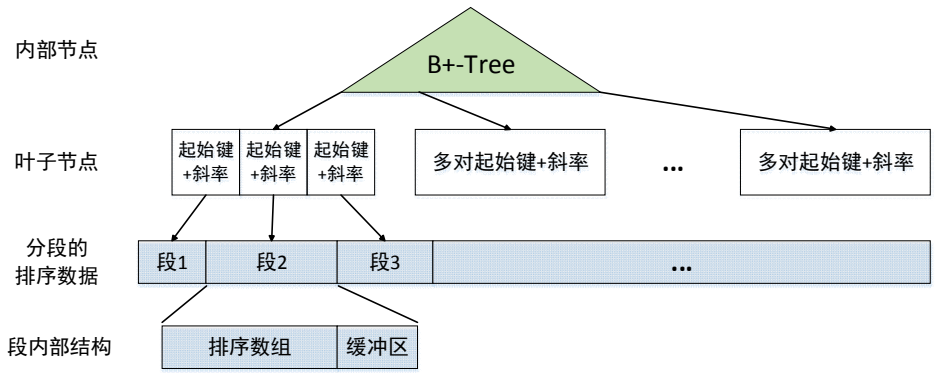

Fig.2 Index structure of FITing-Tree

图 2 FITing-Tree 索引结构

2) 就地插入(in-place insertion)策略. ALEX[21]提出了另一种支持插入的方案,即在叶节点的排序数组中预留一定的空隙.如图 3 所示,当需要插入一个新的键时,首先通过模型来预测插入的位置.如果这是正确的位置,即新插入的键在这个位置保持数组有序,并且这个位置是一个空隙,则直接将键插入到空隙中即可.这是插入的最佳情况,由于键被精确地放置在模型预测的位置,因此之后的基于模型的查找将直接命中.如果模型预测的位置不正确,ALEX 使用指数搜索来找到准确的插入位置.同样,如果插入位置是空隙,那么我们直接在那里插入键;如果插入位置不是空隙,则通过将键沿最接近空隙的方向移动一个位置,在插入位置形成空隙.这种插入策略不仅具有优秀的写入性能,还提升了基于模型的查找性能.与初版学习索引的另一个区别是,ALEX 使用指数搜索(exponential search)替代二分搜索,这是由于在 ALEX 中的基于模型的查找更容易落在正确位置附近,所以指数搜索的性能更加优秀.此外,ALEX 还提出使用 PMA(packed memory array)[22]作为空隙数组的备选方案,PMA 能够提供比空隙数组更优秀的最差插入性能.

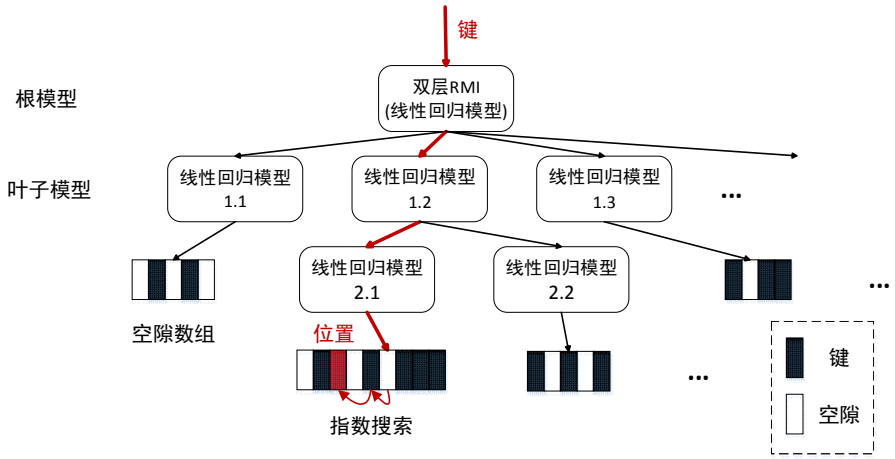

Fig.3 Index structure of ALEX

图 3 ALEX 索引结构

ALEX 提出使用动态 RMI 模型,如图 3 所示.ALEX 被构建之初包含一个双层 RMI 模型作为根模型,以及与叶节点一一对应的叶子模型,所有模型都是线性回归模型.在 ALEX 中,随着空隙数组中被插入更多元素,数组中的空隙数量会减少,插入性能随之下降.当空隙数组的密度达到阈值时,空隙数组会进行分裂;同时,原先对应该叶节点的模型变为内部模型,并在分裂之后的数组上重新训练模型作为新的叶子模型.与初版学习索引相比,ALEX 将空间代价降低了 3000 倍,将查找性能提高了 2.7 倍,同时提供了插入能力.与 B+树相比,ALEX 将空间代价降低了 5 个数量级,将查找性能提高了 3.5 倍,而且插入性能略微优于 B+树.

AIDEL[23]采用与 FITing-Tree[18]类似的索引结构、给定的模型误差和基于模型误差的分段算法,它们唯一的区别是插入策略.AIDEL 通过与记录绑定的排序列表(可视为溢出块)来支持插入.当插入新数据时,AIDEL 首先基于模型找到目标位置,然后将数据插入到目标位置指向的排序列表中.由于排序数组中的数据没有改变,所以插入不会影响模型的误差.采用类似 FITing-Tree 结构的工作还有 ASLM[24]和 PGM-index[25],其中 ASLM 使用了不同的贪心分段算法,并且允许自适应地选择插入策略和更复杂的模型.

3) 基于 LSM 的插入策略与模型压缩. PGM-index[25]从多个方面优化了 FITing-Tree[18].首先,对于数据分段,它提出使用在时间序列有损压缩和相似度搜索中已被广泛研究的流式算法[26-30]替代 FITing-Tree 中的贪心算法,这种算法已被证明能够获得最优的分段线性模型,同时具有 $O(n)$的最优时间复杂度.其次,对于插入和删除,PGM-index 提出像 LSM 树[12,31]一样维护数据,并在每一层都维护一个学习索引,只在触发合并时更新对应的模型.此外,PGM-index 还提出了压缩的模型,分别为起始键和斜率提供无损压缩.其中,由于初始键是排序数据集的子集,可使用已被广泛研究的压缩方法[32-34];对于斜率,PGM-index 专门设计了压缩算法.PGM-index 还提出分布感知的学习索引,不仅学习数据分布,同时学习查询负载分布[35],获得了更优的平均查询性能.最后,PGM-index 与 FITing-Tree 的一个共同优点是,通过建立代价模型,它们可以帮助用户确定每个分段的最大误差阈值——该阈值满足给定最大空间代价并达到最优查询性能或者满足给定最差查询性能并达到最小空间代价.实验表明,与 FITing-Tree 相比,PGM-index 将空间代价降低了 75%;与 B+树相比,PGM-index 将查询和更新性能提升了 71%.

4) 插值法(interpolation). 与基于 RMI 模型结构的学习索引相比,使用分段线性函数拟合数据只需对数据进行一趟扫描即可完成索引构建,并且允许自下而上构建索引.插值法同样具有这个优点.事实上,样条插值法(spline interpolation)[36]等同于分段线性函数.RadixSpline[37]使用样条插值法拟合数据,并使用基数树(radix tree)来索引样条.Setiawan 等人进一步研究了切比雪夫插值法(Chebyshev interpolation)[38]和伯恩斯坦插值法(Bernstein interpolation)[39]在学习索引中的应用[40].

5) 并发数据结构. XIndex索引[41](如图4所示)在解决前两个缺陷的同时,还考虑了索引结构的并发性.与之前的工作不同的是,它将最底层的模型与对应的数据存储在同一个数据结构中,称为组节点(group node).与FITing-Tree、ALEX 和 PGM-index 相同,XIndex 的所有模型都使用线性回归模型.此外,XIndex 也使用了与FITing-Tree 相同的异地插入策略,并将插入的数据存储在缓冲区中.XIndex 的一个组节点中可以包含多个线性回归模型,并使用一个预先设置的误差阈值.随着缓冲区中的数据合并到排序数据中,若模型的最大误差超过误差阈值,则在组节点中进行模型分裂;若模型数量达到组节点的模型数量上界,则触发组节点分裂.

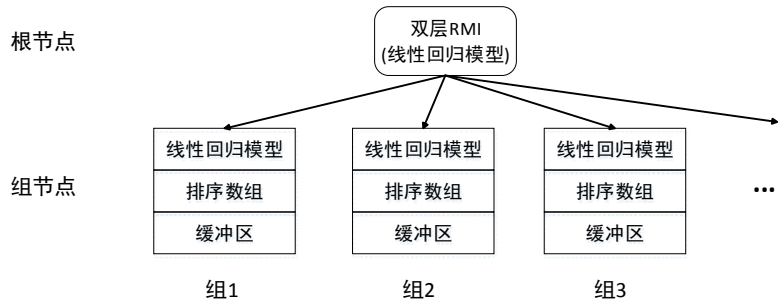

Fig.4 Index structure of XIndex

图 4 XIndex 索引结构

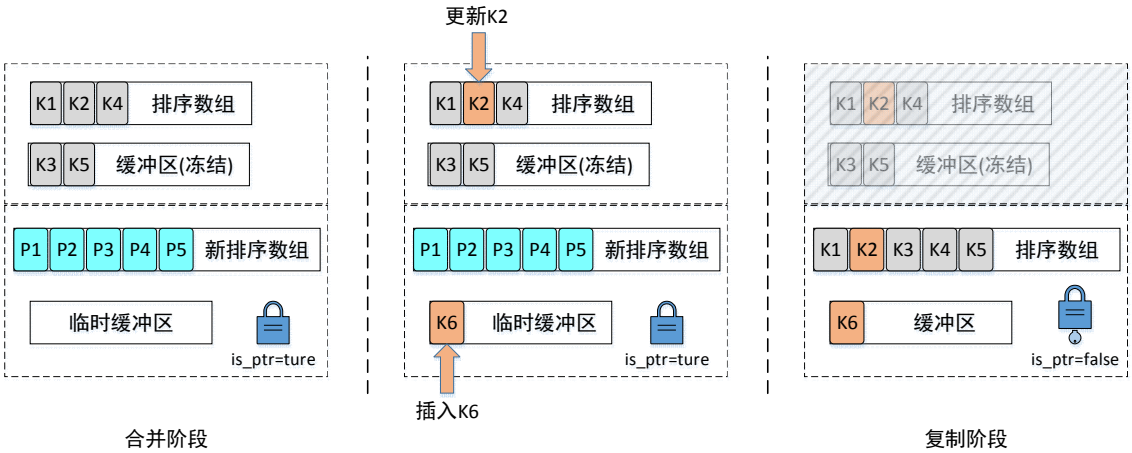

Fig.5 Two phase compaction in XIndex

图 5 XIndex 中的两阶段合并

XIndex 的数据结构支持高并发.它采用已有的乐观并发控制[42-44]、细粒度锁[44-46]和 RCU (read-copy-update)[47]技术,并精心设计了一种两阶段合并(two-phase compaction)算法来支持并发操作.如图5所示,如果缓冲区需要合并到排序数组中,执行过程分为合并阶段和复制阶段.在合并阶段,首先建立一个新组,并将旧缓冲区冻结,此后的数据将被插入到临时缓冲区中,例如图 5 中的 K6.然后,旧排序数组和缓冲区中的数据被合并到新排序数组中,注意,此时新排序数组中存储的是指向旧数组中记录的指针,而非真正的记录.在合并阶段,更新操作将在旧排序数组和缓冲区中执行,例如图 5 中的 K2.然后,XIndex 在新组中训练模型.在完成训练后,XIndex 使用 RCU 屏障(RCU barrier)保证旧排序数组和缓冲区不再被访问,并进入复制阶段.在复制阶段,XIndex 将新排序数组中的每个指针原子替换为最新的记录值.最后,XIndex 回收旧组的内存资源.组分割的过程与之类似,同样分为两个阶段.实验结果表明,与并发索引结构 Masstree[44]和 Wormhole[43]相比,XIndex 的性能分别提升了 3.2 倍和 4.4 倍.

6) 学习式数据划分策略. 上述优化方法倾向于使用更简单的模型,与它们的优化方向相反,Dabble[48]使用多个神经网络模型.在数据空间划分阶段,Dabble 提出使用 K-Means 算法将数据划分为互不相交的 K 个区域.然后,对于每个区域,Dabble 使用一个两层神经网络模型拟合数据.最后,与 PGM-index[25]相似,Dabble 使用 LSM 树的延迟更新策略处理数据插入.

7) 机器学习辅助的传统索引结构. Llaveshi 等人提出保持 B+树索引结构不变,并使用线性回归模型在节点内部加速搜索[49].这种索引结构不是严格的学习索引结构,而是一种机器学习辅助的传统索引结构.该方法提出了一种新的插入策略.由于模型只是节点的辅助组件,所以它延续了传统 B+树节点的就地插入策略.由于插入数据可能会降低模型的精度,所以每个模型都会统计每次预测的误差,当平均误差超过一定阈值时,重新训练模型.这种策略的弊端是不能使用基于误差范围的二分搜索.IFB-Tree[50]同样使用了学习索引的思想提升传统 B+树的性能.它在建立索引时判断每个节点是否是插值友好的,即节点中的所有数据使用插值搜索(interpolation search)的误差都低于给定的阈值.如果节点被标记为插值友好,则在节点中使用插值搜索.与 B+树相比,IFB-Tree 提升了 50%的查找性能.Qu 等人提出了一种 B+树与线性回归模型混合的索引[51],该索引从数据集中剔除离群值(outlier)以提升线性回归模型的预测精度,并使用 B+树索引离群值.

1.2 面向二级索引的学习索引

1.1 节中介绍的学习索引要求要求底层数据按查找键有序存储.如果将学习索引直接用作无序数据上的二级索引(secondary index),则需要将数据按查找键重新排序,并为每条记录附加一个指针.在大规模数据集上,高昂的排序代价以及额外的指针存储空间代价将大大降低学习索引在二级索引上的优势.

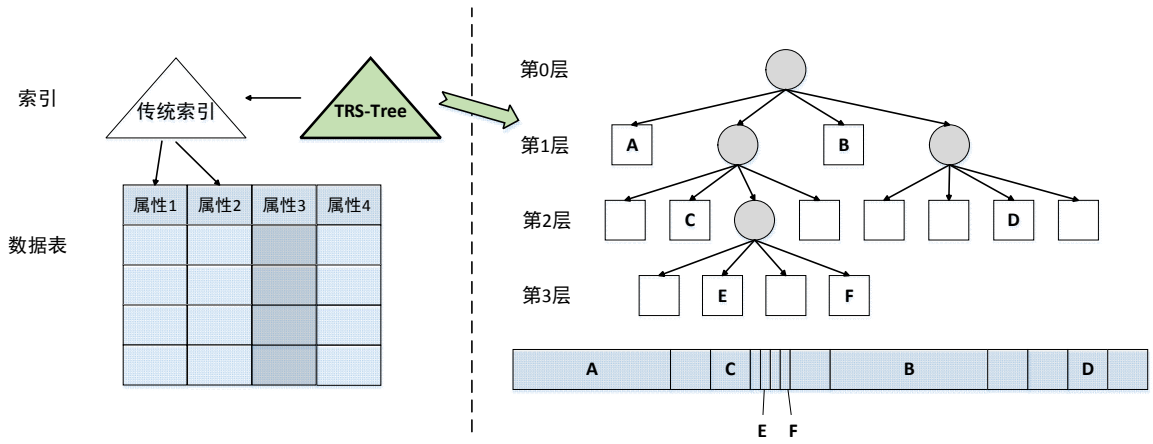

Fig.6 Hermit (left) and TRS-Tree (right, character A-F represents the key range corresponding to the model)

图 6 Hermit(左)与 TRS-Tree(右, 字母 A-F 表示模型对应的属性取值范围)

Wu 等人提出了一种关系型数据库上的简洁二级索引机制 Hermit[52].该索引采用机器学习技术学习列之间隐藏的软函数依赖(soft functional dependency)关系[53,54].如图 6 左所示,属性 1 是数据表的主码,即记录按照属性 1 的顺序存储.若已存在属性 2 的二级索引,且属性 3 与属性 2 存在相关性,那么对于属性 3 的二级索引,可用 TRS-Tree 替代传统的索引结构.TRS-Tree 使用分层回归方法拟合属性 3 与属性 2 的关系,并使用树结构来索引一组回归函数,它的结构如图 6 右所示.TRS-Tree 是一个 k 叉树结构,构造算法均匀地将属性 3 的取值范围划分为 k 个均匀的子范围,直到回归模型能够很好地估计子范围覆盖的条目对.它的叶节点与一个子范围对应,并使用回归模型将属性 3 的子范围映射到属性 2 中的一组记录.图 6 右的上半部分是一棵 TRS-Tree,下半部分表示属性 3 的取值范围,字母 A-F 表示 TRS-Tree 中模型对应的取值范围.TRS-Tree 使用用户设置的误差阈值,但是允许离群值的存在.每个叶节点使用一个离群值缓冲区保存这些离群值.当一个叶节点的离群值与范围内记录

总数的比例达到阈值,则触发节点分裂.

Hermit 支持范围查询,查找过程分为三步.首先,在 TRS-Tree 中查找,返回一组属性 2 上的范围(非离群值)和一组记录标识符(离群值);然后,使用属性 2 的范围作为输入,在传统索引上执行查找,返回一组记录标识符;最后,使用前两步得到的记录标识符获取实际的记录,并验证记录是否满足查询谓词(query predicate),将满足查询谓词的结果返回.实验表明,Hermit 的查找性能略差于传统二级索引,但是空间代价远远低于传统二级索引,并且插入性能是 B+树的 2 倍以上.作者测试了两种拟合函数,其中线性函数能够获得极低的空间代价,而 Sigmoid 函数能够拟合比较复杂的关系.注意,与面向主索引的学习索引有所不同,Hermit 中的模型学习的是两个列之间的关系,而不是键的分布.使用 Hermit 有两个前提条件:两个列之间存在某种联系,并且其中一个列已经建立了索引.

1.3 现有学习索引技术的对比

Table 1 Technology comparison of learned indexes

表 1 学习索引技术对比

名称	内部节点	叶节点	数据划分	模型误差	插入策略	节点分裂	其他优化
初始版本[7](2018)	神经网络	线性回归/B 树节点	不划分	不固定	不支持	不支持	
FITing-Tree[18](2019)	B 树节点	线性回归	贪心算法	用户配置	异地插入(缓冲区)	贪心算法	
AIDEL[23](2019)	枚举	线性回归	贪心算法	用户配置	就地插入(溢出块)	贪心算法	
ALEX[21](2020)	双层 RMI(线性回归)	线性回归	等分	不固定	就地插入(插入空隙)	等分	空隙数组&指数搜索
PGM-index[25](2020)	多层 RMI(线性回归)	线性回归	贪心算法	用户配置	异地插入(LSM)	贪心算法	模型压缩
XIndex[41](2020)	双层 RMI(线性回归)	线性回归(分组)	分组(等分)	不固定(给定上界)	异地插入(缓冲区)	等分	两阶段合并
Dabble[48](2020)	枚举	神经网络	K-Means	不固定	异地插入(LSM)	K-Means	
Hermit[52](2019)(二级索引)	k 叉树	线性回归/逻辑回归	递归 k 等分	用户配置	就地插入	k 等分	离群值缓冲区

表 1 给出了目前代表性的学习索引,并总结了不同索引之间的技术差异.

内部节点的功能是索引叶节点,它需要能够拟合 CDF 的大体形状.初始版本使用神经网络模型,能够较好地拟合数据分布.但是,神经网络模型的执行具有较高的乘法代价,而且训练速度较慢,所以初始版本的学习索引不支持插入操作.在后续的研究中,由线性回归模型构成的双层 RMI 模型被认为是较好的选择.

叶节点的功能是拟合某一小段数据,出于执行代价、易于更新和空间代价的综合考虑,线性回归模型在当前是最优选择.

数据划分指的是底层数据的划分策略.为支持插入操作,数据被分段存储,每一段数据与一个叶节点的模型对应.为了使每一段数据都满足用户给定的误差阈值,FITing-Tree[18]和 PGM-index[25]使用串行的贪心算法执行分段并训练模型,具有 $O(n)$的时间复杂度.其他版本的学习索引使用等分策略,可以并行的在每个段中训练模型.

模型误差指的是叶节点的模型误差,误差的大小会影响后续的二分搜索性能.初始版本的学习索引没有预先给定模型误差,所以它的最差查找性能是不确定的.为了解决这一问题,FITing-Tree[18]和 PGM-index[25]使用预先设置的误差阈值串行执行数据划分.而 XIndex[41]和 Hermit[52]采用递归等分数据的策略,如果数据段中的模型误差超过给定阈值,等分数据并重新训练模型.这种方式虽然可以并行执行,但是可能需要多次训练模型,性能优劣取决于初次数据划分.

在**插入策略**上,异地插入策略被广泛采纳,因为异地插入不仅能够分摊数据移位代价,还能够分摊模型重训练代价,因为数据插入会改变模型的误差.然而,异地插入不可避免地降低了查询性能.AIDEL[23]使用与记录绑定的溢出块存储新插入的数据.理论上,与异地插入策略相比,溢出块策略的插入性能略差,查找性能更好,但是溢出块的管理比缓冲区更复杂.ALEX[21]使用空隙数组结构,配合使用就地插入策略和指数搜索策略,获得了较优的查找和插入性能.与表 1 中涉及的方法不同,Hadian 等人提出使用就地插入策略,但是每次插入后不立即重新训练模型,而是在每个段中统计插入数量,并在查找时使用模型误差加上插入数量作为二分搜索的界限[55].当某个段中的插入数量足够多时,重新训练模型.Bilgram 等人为学习索引建立了代价模型,用于判断何时进行重训练[56].

在**节点分裂**上,与传统的 B+树不同,所有学习索引的分裂都是由模型触发的.当一段数据对应的模型不能满足给定的误差阈值时,触发模型分裂.其中,XIndex[41]的一个组节点中可以包含多个模型,而在其他学习索引中一个数据段只对应一个模型,在模型分裂的同时数据段也需要分裂.

1.4 与传统B+树索引的对比

与 B+树相比,学习索引额外引入了机器学习模型以提升索引性能.学习索引之所以能够提升性能,其关键之处在于:

(1) 一个机器学习模型能够覆盖几万个键,这使得学习索引的叶节点数量远远小于 B+树.这有助于降低索引的空间代价.

(2) 机器学习模型能够在几万个键中快速地将搜索范围缩小到几十个键.这是通过模型计算实现的,而不是通过传统 B+树上的逐层键值比较和指针访问操作,因此可以降低索引查找代价.

Table 2 The fitting power of the linear models

表 2 线性模型拟合能力

误差边界	B+树叶节点数量	B+树叶节点大小	线性模型数量	学习索引叶节点平均大小	放大倍数
8	6250000	16	375938	266	16.63
16	3125000	32	101791	982	30.69
32	1562500	64	26537	3768	58.88
64	781250	128	6793	14721	115.01
128	390625	256	1798	55617	217.25

我们测试了在学习索引中最常用的模型——线性模型的拟合能力,测试使用 OptimalPLR 算法[25,30],这是一种 O(n)时间复杂度的分段线性拟合算法,能够获得给定误差边界的最小分段数量,测试数据集为 1 亿个正态分布的浮点型随机数.测试结果如表 2 所示,在给定的误差边界下,一个线性模型所覆盖的键数量远高于 B+树节点,这意味着学习索引的叶节点数量远少于 B+树索引,从而降低空间代价.同时,更少的叶节点意味着学习索引的树高要低于 B+树索引,降低了学习索引的查找时间.

同时,与 B+树相比,学习索引也存在一些额外代价,总结如下:

(1) 模型计算代价.学习索引在查找过程中使用模型计算代替了传统 B+树的键值比较和指针操作.当使用比较简单的线性模型时,模型计算代价较小.但如果数据分布难以用线性模型进行拟合,必须使用复杂模型时,则会引入较大的计算代价.

(2) 模型拟合代价.学习索引借鉴机器学习的训练过程,通过学习数据的分布来构建计算模型.为了保证模型的准确性,一般要求训练的数据要足够大.这将导致较高的模型拟合代价,使得学习索引的构建速度要远远慢于 B+树索引.实验表明,使用 OptimalPLR 算法构建学习索引的速度只相当于 B+树构建速度的十分之一.而且,当索引的数据发生较大变化时,例如插入或删除了大量键值,需要对模型进行更新或者重建,这同样会影响学习索引的性能.

(3) 数据移位代价.由于学习索引的叶节点远远大于 B+树节点,所以它在执行插入时存在较大的数据移

位代价.尤其当模型预测的准确率较低时,数据移位的代价将更高.但如果重新构建计算模型又会带来较大的延迟.

2 面向多维范围查询的学习索引

多维查询是指一次查询涉及多个属性维度.最经典的多维查询应用是空间查询.这类查询返回二维平面空间中的一个矩形范围.传统的一维索引对于多维查询不再有效,需要设计专门的索引结构,即多维索引或空间索引.目前关于空间索引已有大量研究成果,主要可分为以下三类(详细的多维索引研究报告可参考综述文献[57,58]).

第一类索引将空间划分为区域,并对这些区域建索引,典型的索引结构有 KD 树[59](面向 k 维空间)、四叉树(quadtree)[60](面向二维空间)和八叉树(octree)[61](面向三维空间).第二类索引将数据集划分为不同的子集,然后对这些子集进行索引,经典的索引结构有 R 树[62]和它的变体[63-65].第三类索引将多维空间转换为一维空间,然后在一维空间上建索引,典型的转换方法有希尔伯特空间填充曲线(Hilbert space filling curve)[66]和 Z 顺序曲线(Z-order curve)[66,67].这些结构有的是专为二维或三维空间设计的,有的结构可以适用于更高维空间,或者易于扩展到高维场景.

将学习索引的思想应用到多维索引中并不容易,因为学习索引要求数据有序,而多维数据并不是天然有序的.一种直观的解决方式是使用第三类索引,即将多维空间上的数据投影到一维空间上,并在一维空间上建立学习索引.这种方法的难点在于投影策略的设计.如果使用简单的多维 CDF 作为映射函数,那么具有相同映射值的两个点可能在多维空间中相距很远.所以,设计优秀的映射函数,或者说设计优秀的数据布局(data layout)策略,是设计学习多维索引的关键.SageDB[68]提出将空间划分为多维网格,数据按照网格进行排序,但是文中并没有给出实现细节.Wang 等人使用 Z 顺序曲线进行投影,提出了学习 ZM 索引[69],并在一维空间上建立 RMI 模型.学习 ZM 索引的查询性能优于 R 树,并且空间代价远远小于 R 树.

在空间查询应用中,除了查询确定的空间范围外,另一种常见的查询方式是 k-最近邻(k-nearest neighbors,简称 KNN)查询.KNN 查询返回距离某一点最近的 k 个目标.目前已有大量基于 R 树的 KNN 查询研究.然而,SageDB[68]和学习 ZM 索引[69]都没有关注 KNN 查询.ML-Index[70]是一种支持 KNN 查询的多维学习索引,它使用的投影策略是一种基于 iDistance 方法[71]的改进策略.ML-Index 首先在多维空间中选择一定数量的参考点进行分区,每个参考点代表分区的中心.在 iDistance 方法中,多维空间中的点被映射为该点到分区中心的距离加上分区编号与一个常数的乘积.然而,由于每个分区的大小不同,很难为这个常数找到合适的值.ML-Index 使用偏移量替代这个乘积,解决了分区重叠问题.在得到的一维投影上,ML-Index 构建了一个由简单回归模型组成的 RMI. ML-Index 使用基于 iDistance 的方法执行 KNN 查询,并使用基于数据的范围近似方法[72]执行范围查询.ML-Index 的空间代价远小于传统的多维索引,KNN 查询速度优于 iDistance 方法;与学习 ZM 索引[69]相比,ML-Index 在二维空间的性能略差,但是在更高维的空间中性能更优.

LISA[73]结合了 SageDB 和 ML-Index 的思想.它首先使用网格划分数据,然后借助一个映射函数将数据映射到一维空间.映射函数需要保证处于小编号网格单元中的点的映射值一定比处于大编号网格单元中的点的映射值小,LISA 采取了与 ML-Index 类似的基于勒贝格测度(Lebesgue measure)[74]的方法.在一维空间上,LISA 将数据等分到一定数量的分片中,并训练一个分段线性回归模型来预测分片.对于每个分片,LISA 还建立一个局部模型来预测分页.对于范围查询,LISA 首先得到与查询矩形相交的单元格,并依照单元格将查询矩形分成多个小矩形.对于每个小矩形,LISA 使用映射函数、分段线性回归模型和局部模型获取相关的页面.对于 KNN 查询,LISA 使用格子回归(lattice regression)模型[75]结合范围查询进行搜索.此外,LISA 还支持插入操作,它采取异地插入的方式将数据插入到模型预测的页面中.实验表明,与 KD 树和学习 ZM 索引相比,LISA 具有略差的空间代价,但是大幅降低了查询的 I/O 代价;与 R 树相比,LISA 具有略优的查询 I/O 代价和空间代价.

Flood[76]扩展了 SageDB 中关于多维索引的优化思想,它是网格索引(grid index)[76]的变体.与前面介绍的工作不同,它使用机器学习方法来调优数据布局.对于 d 维空间中的数据,Flood 首先对 d 个维度进行排序,并使用

排序中的前 d-1 维在数据上覆盖一个 d-1 维网格,使用第 d 维作为排序维给每个网格单元中的数据排序.给定一个查询范围,Flood 首先找到与查询范围的超矩形相交的网格单元,然后在每个网格单元中使用排序维确定需要扫描的范围,最后扫描数据并过滤出与查询匹配的记录.Flood 的网格布局有几个需要调优的参数,分别是每个维度被划分的列数,以及选择哪个维度作为排序维.这些参数很难调优,因为他们取决于许多互相影响因素,包裹查询过滤器在维度上的频率、每个维度上选择系数的平均值和方差、维度之间的相关性、维度在查询负载中的相关性等.Flood 首先随机生成一些数据布局,并使用合成的数据集和查询负载生成许多训练实例,使用得到的特征训练一个随机森林回归模型[77],最终生成一个成本模型.然后,Flood 迭代选择排序维,并使用梯度下降搜索寻找每个维度的最优列数,得到 d 个布局,并使用成本模型在 d 个布局中选择目标函数成本最低的布局.在每个网格单元内部,Flood 使用分段线性回归模型拟合依照排序维进行排序的数据.实验表明,Flood 比多种著名的空间索引[59,63,67,78,79]快 40 至 250 倍,并节省了 50 倍以上的空间.目前还有一些基于机器学习的数据布局调优工作[80,81],但它们不属于学习索引的范畴,因此本文不展开讨论.

Table 3 Comparison of learned multi-dimensional indexes

表 3 学习多维索引对比

名称	投影策略/数据布局	一维空间学习模型	KNN 查询	插入	其他优化
SageDB[68](2019)	网格划分	RMI	未讨论	不支持	
学习 ZM 索引[69](2019)	Z 顺序曲线	RMI	不支持	不支持	
ML-Index[70](2020)	基于 iDistance 的改进策略	RMI (简单模型)	基于 iDistance 的改进策略	不支持	
LISA[73](2020)	网格划分+基于勒贝格测度的投影	分段线性回归模型	基于格子回归模型的查询算法	异地插入	用于处理分页的局部模型
Flood[76](2020)	基于成本模型的网格划分	分段线性回归模型	支持 (未讨论)	不支持	机器学习训练的成本模型

表 3 给出了面向多维空间的学习索引对比.其中,SageDB 和学习 ZM 索引使用了已有方法,而 ML-Index 和 LISA 使用了改进的映射函数,而 Flood 则使用机器学习方法优化数据布局.在将数据映射到一维空间后,这些索引普遍采用了前面讨论过的 RMI 和分段线性回归模型.更进一步,ML-Index 和 LISA 讨论了如何支持 KNN 查询,LISA 还探索了如何支持数据插入.

3 面向点查询的学习索引

现实世界中存在着某些特定的应用,它们不支持或不需要范围查询.针对这些应用需求,人们设计了面向点查询的索引结构.例如,哈希索引具有 O(1)的查询时间复杂度和空间复杂度,优于 B+树索引结构;Trie 树是一种面向字符串查询的索引结构,被广泛用于文本词频统计;倒排索引被广泛应用于搜索引擎中,使用文本反向检索文档.目前,学习索引在哈希索引和倒排索引中都已有一些研究,本节将介绍这些技术.

3.1 传统哈希索引

由于哈希索引本身就是一组函数模型且具有 O(1)的查找性能,所以不能再使用之前的优化思路,即使用模型代替索引以减少空间代价并提升预测精度.然而,哈希索引存在哈希冲突问题,这会导致索引性能下降.完美哈希(perfect hashing)[82]和布谷鸟哈希(cuckoo hashing)[83]分别致力于在静态数据集和动态数据集上减少哈希冲突.Kraska 等人提出,通过学习键的分布可以得到更好的哈希函数,即能够减少哈希冲突[7].与第 1 节中面向一维范围查询的学习索引不同,学习哈希索引的目标是将学到的数据分布均匀的映射到给定数量的哈希桶中,这只需要将键的 CDF 按照给定的哈希桶数量缩放即可.与面向范围查询的学习索引相同,学习哈希索引继续使用 RMI 模型学习键的分布.与简单的哈希函数相比,学习哈希索引显著降低了哈希冲突的概率.同时,与完美哈希

的大小会随着数据集大小增长不同,学习哈希索引的大小不会受到数据集大小的影响.然而,在较小的数据集上,学习哈希索引显然在空间效率上不具备优势.

3.2 局部敏感哈希

局部敏感哈希(locality sensitive hashing,简称 LSH)[84]与传统的哈希索引不同,它利用哈希冲突加速近似最近邻(approximate nearest neighbor,简称 ANN)搜索.目前,LSH 在图像检索等高维数据检索中得到了广泛应用.LSH 的主要思想是通过设计一个哈希函数使得高维空间中距离相近的两点很大概率具有一样的哈希值.LSH 利用随机投影或排列来生成与数据无关的哈希函数,但这种方法容易遇到性能瓶颈和语义鸿沟(semantic gap)[85]问题.为了解决性能问题,人们提出了使用机器学习方法学习数据依赖,进而得到面向应用的哈希函数,从而生成更有效且紧凑的哈希码[86,87]并保证语义一致性[88-91].为了实现这一目标,人们对机器学习模型进行了大量探索[92],包括使用卷积神经网络(convolutional neural networks,简称 CNN)作为哈希函数[93]等.与 3.1 节中的学习哈希索引不同的是,LSH 相关的工作通过学习数据分布来指导哈希函数的设计,而没有改变基本的数据结构.本文主要关注使用机器学习模型替代索引结构这一创新观点,故不展开讨论这部分内容.而且,利用机器学习技术优化 LSH 已经历长久的研究,详细内容可参看综述文献[92].

3.3 倒排索引

倒排索引是文档检索中最常用的数据结构之一,它存储单词到文档的映射,从而可以快速检索包含某些单词的文档.倒排索引目前已被广泛应用于搜索引擎[94,95].随着互联网中数据量的急速增长,存储完整的倒排表需要巨大的空间代价.受到学习索引的启发,Oosterhuis 等人使用深度神经网络(deep neural networks,简称 DNN)模型学习得到一个布尔函数[96],该函数接受一个单词项和一个文档作为输入,并预测该单词项是否存在于该文档中.然而,他们没有介绍学习布尔模型的具体细节,而只是研究了如何应用该模型.作者结合了已有的倒排索引查询优化方法[97,98],并展示了一个优秀的学习布尔模型能够带来巨大的空间收益.

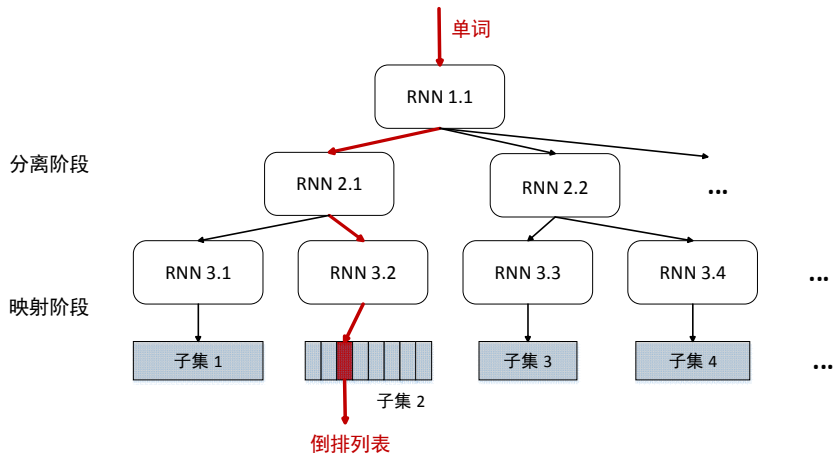

Fig.7 Index structure of Pavo

图 7 Pavo 索引结构

单词词典是倒排索引的重要组成部分,它记录在文档集合中出现的单词,以及该单词对应的倒排列表在倒排文件中的位置信息.对于一个规模很大的文档集合,单词词典包含大量不同单词.因此,需要高效的数据结构来加速单词词典中的查找,常用的结构包括哈希、B+树和 Trie 树等.受到 Kraska 等人提出的学习哈希索引的启发,Pavo[99]使用基于循环神经网络(recurrent neural network,简称 RNN)的分层模型替代倒排索引中的哈希函数.如图 7 所示,Pavo 使用类似 RMI 的层次模型结构,它内部的所有模型都采用 RNN 模型,每个 RNN 模型由一个单层的长短期记忆(long short-term memory,简称 LSTM)层和一到两个全连接层组成.每一层中的不同模型负责

互不相交的数据子集,并在不同层中递归地划分数据集.实验表明,与传统哈希索引相比,Pavo 具有更低的哈希冲突概率和更高的空间利用率,但是查询速度慢 3 至 4 倍;此外,在相同的模型参数量下,无监督学习对数据分布的拟合效果优于有监督学习.

4 面向存在查询的学习索引

存在查询用于判断某一元素是否存在于一个集合中.最经典的支持存在查询的数据结构是布隆过滤器(Bloom filter)[100].在数据库中,布隆过滤器通常作为索引的辅助组件出现.例如,谷歌的 Bigtable[101]使用它判断键是否存在于 SSTable 中,从而避免不必要的磁盘查找;BF-Tree[102]使用它替代 B+树的叶节点,减少索引的空间占用;此外,它还被用于网络安全等领域[103].布隆过滤器由一个二进制位向量和一组哈希函数组成,具有高插入性能、高查找性能、低空间占用等优点.当插入一个新的元素时,布隆过滤器使用哈希函数将键映射得到一组数字,并在二进制位向量中将这组数字对应的位置设为 1.当查找某个元素时,如果元素对应的一组哈希值在二进制向量中都为 1,则布隆过滤器判定该元素存在于集合中,否则判定不存在.注意,由于存在哈希冲突,所以布隆过滤器只能保证假阴性概率(false negative rate,简称 FNR)为 0,但是假阳性概率(false positive rate,简称 FPR)大于 0.布隆过滤器的 FPR 与空间代价呈负相关,在大数据集上,为了获取一个较低的 FPR,需要很高的空间代价,这限制了布隆过滤器的空间效率.

Kraska 等人提出了学习布隆过滤器[7],通过学习数据分布降低哈希冲突的概率.注意,与第 3 节中的学习哈希索引不同,学习哈希索引的目标是降低存在的键之间的冲突概率,而学习布隆过滤器的目标是降低存在的键与不存在的键之间的冲突概率.Kraska 等人提出的学习布隆过滤器将存在索引视为一个二元概率分类任务,并训练一个 RNN 模型预测键存在的概率.学习布隆过滤器使用一个概率阈值,当存在概率大于这个阈值时,则判定该键存在,否则判定不存在.通过调节概率阈值的大小,能够获得需要的 FPR 大小.然而,与传统布隆过滤器不同的是,使用 RNN 模型和概率阈值不能保证 FNR 为 0,而且 FNR 大小与 FPR 大小呈负相关.所以,如图 8 左所示,对于 RNN 模型判定不存在的键,需要再使用一个后置布隆过滤器,从而将 FNR 降为 0.实验表明,与传统布隆过滤器相比,学习布隆过滤的空间效率更高.

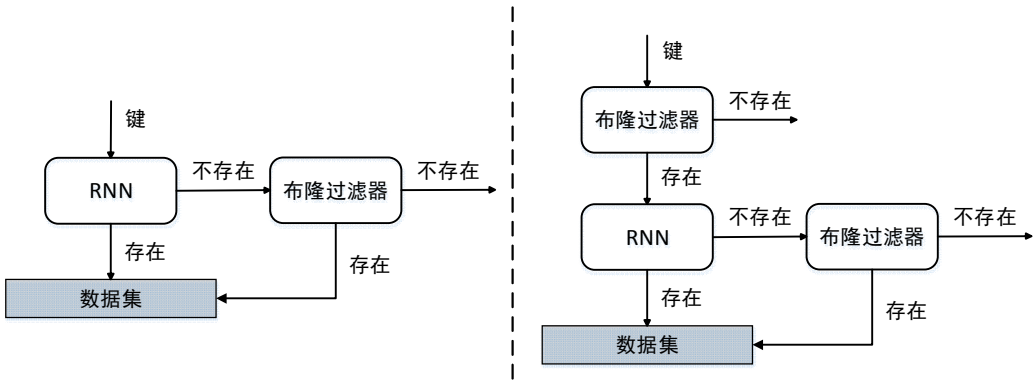

Fig.8 Original learned Bloom filter (left) and sandwiched learned Bloom filter (right)

图 8 原始学习布隆过滤器(左)和三明治结构学习布隆过滤器(右)

Mitzenmacher 改进了学习布隆过滤器,提出了三明治结构的学习布隆过滤器[104].除了使用一个后置布隆过滤器外,三明治结构还使用了一个前置布隆过滤器,如图 8 右所示.由于后置布隆过滤器的大小与通过 RNN 模型的假阴性元素数量呈正相关,所以通过使用前置布隆过滤器消除更多的假阴性,能够降低后置布隆过滤器的空间代价.三明治结构的另一个优点是它与 Kraska 等人提出的学习布隆过滤器结构相比具有更强的鲁棒性.如果用于学习布隆过滤器的训练集和测试集具有不同的数据分布,那么 RNN 模型的 FNR 可能远远大于预期.增加一个前置布隆过滤器能够缓解这个问题,因为它预先过滤了一部分假阴性元素.Ada-BF[105]提出利用 RNN 模

型输出的概率评分的分布,自适应调整不同区域的哈希函数数量,从而调节不同区域的 FPR,以获得更好的整体 FPR.

上述的学习布隆过滤器只能应用在静态数据集上,因为它们需要预先学习数据集的数据分布.然而,许多应用场景都需要支持数据集的动态更新,这要求布隆过滤器能够支持高效的数据插入.Rae 等人使用元学习(meta-learning)和记忆增强神经网络(memory-augmented neural network)[106,107]解决这一问题,提出神经布隆过滤器[108].神经布隆过滤器不是从零开始学习,而是从公共数据分布的采样中学习,这适用于在公共数据的不同子集上实例化多个布隆过滤器的应用程序,例如,Bigtable 为每个 SSTable 文件应用一个布隆过滤器[101].在数据库应用中,与布隆过滤器和布谷鸟过滤器(cuckoo filter)[109]相比,神经布隆过滤器将空间代价降低了 30 倍.Bhattacharya 等人提出两种方案处理学习布隆器中的数据插入[110],第一种方案通过重训练异地插入的数据,从而更新分类器模型;第二种方案使用动态分区布隆过滤器[111]替代传统布隆过滤器,它的空间代价会随着数据插入而增长.

5 面向学习索引的测试基准

测试基准是进行实验验证的重要组件,包括数据集、负载和对比方案.使用公开的测试基准能够使论文的实验结果更具说服力.由于学习索引是近年来才出现的研究方向,因此目前只有较少的工作研究了面向学习索引的测试基准.SOSD[112]是一个面向一维范围查询的学习索引测试基准,它提供了 RMI 模型的开源实现.此外,它还提供了多种可选方案作为对比,包括直接应用在排序数组上的动态搜索算法以及会占用额外空间的索引方案.其中,动态搜索算法包括二分搜索、基数二分搜索、插值搜索和最近提出的三点差值搜索[113];索引方案包括多种经典的数据库索引:自适应基数树[114]、流行的 B+树实现[115]以及 FAST[116].除了上述对比方案, ALEX[21]、PGM-index[25]和 XIndex[41]也提供了开源实现.SOSD 包含 8 种不同的数据集:图书销售人气数据[117]、Facebook 用户 ID 数据集的上采样版本[118]、OpenStreetMap[119]位置数据的对数正态分布、正态分布和均匀采样版本、密集整数、均匀分布稀疏整数和维基百科文章编辑时间戳[120].此外,使用 web 日志数据集或物联网数据集[121]也是一种可选方案.

SOSD 仅提供了单一的一维范围查询负载,更复杂的工作负载可以考虑数据库领域的流行测试基准.在针对索引插入的研究中,需要使用读写混合负载,可以选择 TPC-C[122]和 YCSB[123]作为测试基准.对于多维查询,可以选用公开的地图数据集[119]或图像数据集[124],以及 TPC-H[125]测试基准.然而,对于倒排索引和布隆过滤器的研究,目前缺乏公开的测试基准.

6 研究展望

虽然目前已有许多工作对学习索引进行了探索,但是与已经经历了数十年发展的传统数据库索引相比,学习索引尚处于萌芽阶段,还需要更多的研究与探索.特别是,学习索引尚处于实验室测试阶段,目前还没有在生产环境中集成学习索引的实例,这需要数据库社区与企业的更多参与和努力.基于我们对目前学习索引研究现状的调研以及数据库、大数据等领域的发展趋势,本文认为下面的一些问题是未来值得探索的研究方向:

适合学习索引的机器学习模型.虽然已有的工作针对学习索引提出了许多模型,但是这些模型的结构比较单一,大多是基于 Kraska 等人提出的模型结构.对于面向一维范围查询的学习索引,已有的工作都直接使用 RMI 模型或设计类似于 RMI 的层次结构,而没有探索其他模型结构的可能;而且,这些工作仅采用简单的神经网络和线性回归模型.虽然线性回归模型具有易于更新、存储代价低、计算量低等优点,但是它并不能模拟所有的数据分布.同样,对于学习布隆过滤器,已有的工作都仅是基于 Kraska 等人提出的学习布隆过滤器结构进行优化,并使用 RNN 模型学习键存在的概率,但还没有人探索将存在索引替换为其它机器学习模型的可能性.此外,学习索引的研究仅限于本文提到的几种索引类型,仍有其它类型的索引值得研究.例如,目前还没有学习式的 Trie 树和基数树.在众多的机器学习模型中,如何选择适合学习索引的机器学习模型是未来值得进一步探索的一个研究方向.

学习索引的模型调优方法.传统的索引仅需要调节少量直观的参数,例如 B+树的节点大小和填充率.使用机器学习模型替代传统索引,使索引调参变得复杂,如何减少学习索引与用户的交互、降低学习索引的调参难度是一个值得研究的方向.这方面的研究成果目前仅有 CDFShop[126],它展示了 RMI 模型的调参过程,并探索了自动化调参的可能性.然而,其它学习索引的自动化调参目前还是空白,有待进一步探索.此外,大多数学习索引仅学习数据分布或仅学习查询负载分布,如何在一个模型中同时学习数据分布和查询负载分布,还需要进一步研究.

可动态更新的学习索引.传统的 OLTP(online transaction processing)应用往往要求索引能够支持动态的数据更新,例如插入和删除.由于机器学习模型大都需要在一个样本数据集上学习得到分布规律,如果数据集动态变化,也就意味着学习得到的分布规律也可能动态变化,由此将导致学习过程的代价剧增.正因为如此,目前大多数的学习索引并不能很好地支持数据集的动态更新.Kraska 等人提出的几种学习索引都不支持数据插入.在面向一维范围查询的学习索引中,已有大量支持插入的学习索引变体.这些研究提出了多种方法来支持插入,包括基本的就地插入方法、基于空隙数组的方法、基于缓冲区的方法和基于 LSM 的方法等.但是,哪种插入方法更好目前尚没有定论,有待进一步的对比研究.此外,为了高效地支持插入,这些学习索引变体都采用简单的分段线性回归模型.然而,对于多维查询、存在查询、文本查询等场景,简单的线性回归模型不再有效,需要采用更复杂的模型,在这些索引类型上支持插入的难度更高,目前的研究进展尚处于起步阶段.

学习索引与其它领域的结合.学习索引的思想不仅仅能够用于数据库索引,而且可以应用在更多的领域中.在计算机视觉领域,IndexNet[127]利用学习索引的观点设计通用的上采样操作符,证明了它在图像匹配方面的有效性.在生物信息学领域,LISA[128]将学习索引的思想应用于 DNA 序列搜索,它将该应用中流行的索引结构 FM-index[129]替换为一个模型;与之类似,Sapling[130]利用学习索引思想加速基因组查找,它将后缀数组的内容建模为一个函数,并使用分段线性模型来有效地近似这个函数.此外,学习索引的思想还能够加速排序算法,文献[68,131]利用机器学习模型学习经验 CDF,快速地将元素映射到排序位置的近似位置,与经典排序算法相比获得了明显的性能提升.加速排序算法能够使更多数据结构和算法受益,例如 database cracking[132].以上研究结果也进一步说明了学习索引思想拥有巨大的潜力,探索机器学习模型在更多数据结构中的应用具有重大的意义.

分布式学习索引.目前,大多数学习索引的研究都是面向单线程负载的.想要高效地支持多线程负载,需要设计并发数据结构.理论上,大多数传统并发数据结构的设计思想仍可用于学习索引,但是需要进行一些调整.目前已有的工作仅有 XIndex,它在学习索引结构中应用了细粒度锁和乐观并发控制技术,这方面仍然具有较大的研究空间.此外,随着远程直接内存访问(remote direct memory access,简称 RDMA)技术[133]的出现,节点间通信延迟能够达到微秒级,这促进了分布式索引结构的发展[134,135].使用分布式数据结构的一个优点是能够将计算负载分摊到每个节点,而使用 RDMA 技术,甚至可以由客户端分摊服务端的计算负载[134,136,137].对于计算量远大于传统索引的学习索引来说,设计一种基于 RDMA 的分布式数据结构是未来值得探索的一个方向.

学习索引与硬件加速的结合.使用机器学习模型替代传统索引结构的一个优点是,它使得索引能够利用计算机中的图形处理器(graphics processing unit,简称 GPU)或张量处理器(tensor processing unit,简称 TPU)[138]的算力.然而,GPU 和 TPU 具有较高的调用代价,考虑设计一种批量处理策略来分摊调用代价是一种可行的方案.目前这方面的研究处于空白状态,研究如何高效地结合 CPU 与 GPU/TPU 是一个有潜力的研究方向.

面向非易失内存的学习索引.非易失内存(non-volatile memory,简称 NVM)兼具内存的低延迟、可字节寻址以及外存的持久存储等优点,有望在未来改变现有的存储架构,使得搭建基于纯内存的内存数据库系统成为可能.近年来,NVM 技术取得了一定的突破,已经诞生了商业产品 Intel Optane DC 持久内存[139].学习索引假设数据存储在内存中,它的预测精度达到了字节粒度,而不是传统索引的磁盘块粒度.NVM 能够为学习索引带来持久性、可字节寻址特性和更大的内存空间,这些都是现有学习索引急需的特性.然而,为 NVM 设计数据结构存在许多额外的挑战.首先,为了提供持久性保证,需要使用更细粒度的持久化指令,例如 clflush 和 mfence 等;其次,NVM 是一种读写不均衡的存储器,写延迟一般高于读延迟,因此需要设计 NVM 写友好的数据结构;最后,由于 CPU 仅支持 8 字节原子写,对于超过 8 字节的更新需要设计额外的方案来保证失败原子性(failure atomicity).

总体而言,虽然目前已有一些面向 NVM 的索引研究,包括 B+树[140-143]、基数树[144]、哈希索引[145-147]和混合结构[148]等,但设计面向 NVM 的学习索引还有许多问题尚未解决,是未来值得深入研究的一个方向.

7 结束语

学习索引尝试使用机器学习模型直接替代传统的数据库索引结构,并提升查找性能和降低索引的空间代价.学习索引是目前机器学习和数据库技术相结合的一个重要突破,因此一经提出即引起了学术界的广泛关注.本文综述了学习索引的研究进展,并提出了学习索引的一个分类框架.基于该分类框架,论文详细讨论了各类学习索引的问题、进展和存在的问题.其中,第一类是面向一维范围查询的学习索引,这一类学习索引得到了最多的关注,在插入策略和模型设计等方面得到了优化.第二类是面向多维范围查询的学习索引,这一类工作主要关注如何将多维空间数据投影到一维空间,并使用面向一维空间的机器学习模型进行学习.第三类是面向点查询的学习索引,这一类工作关注如何使用机器学习方法降低或增加哈希冲突的概率.最后一类是存在索引,这一类工作将存在查询建模为二元概率分类任务.最后,论文还介绍了面向学习索引的测试基准研究进展,并对学习索引的未来研究方向进行了展望.

References:

[1] Bayer R, Mccreight E. Organization and maintenance of large ordered indices. In: Proc. of the Int. Conf. on Management of Data. 1970. 107-141.

[2] Garcia-Molina H, Ullman JD, Widom J. Database System Implementation, Second Edition. Prentice Hall; United States Ed. 1999.

[3] Garciamolina H, Salem K. Main memory database systems: an overview. IEEE Trans. on Knowledge and Data Engineering, 1992,4(6):509-516.

[4] Lehman TJ, Carey MJ. A study of index structures for main memory database management systems. In: Proc. of the Int. Conf. on Very Large Data Bases. 1986. 294-303.

[5] Rao J, Ross KA. Cache conscious indexing for decision-support in main memory. In: Proc. of the Int. Conf. on Very Large Data Bases. 1999. 78-89.

[6] Zhang H, Andersen DG, Pavlo A, Kaminsky M, Ma L, Shen R. Reducing the storage overhead of main-memory OLTP databases with hybrid indexes. In: Proc. of the Int. Conf. on Management of Data. 2016. 1567-1581.

[7] Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N. The case for learned index structures. In: Proc. of the Int. Conf. on Management of Data. 2018. 489-504.

[8] Meng XF, Ma CH, Yang C. Survey on machine learning for database systems. Journal of Computer Research and Development, 2019,56(9):1803-1820 (in Chinese).

[9] Li GL, Zhou XH, Sun J, Yuan HT, Liu JB, Han Y. A Survey of machine-learning-based database techniques. Chinese Journal of Computers, Vol.42, 2019, Online No.120:1-33 (in Chinese).

[10] Chai MK, Fan J, Du XY. Learnable database systems: Challenges and opportunities. Ruan Jian Xue Bao/Journal of Software, 2020,31(3):806-830 (in Chinese).

[11] Sun LM, Zhang SM, Ji T, Li CP, Chen H. Survey of data management techniques powered by artificial intelligence. Ruan Jian Xue Bao/Journal of Software, 2020,31(3):600-619 (in Chinese).

[12] Oneil P, Cheng EY, Gawlick D, Oneil E. The log-structured merge-tree (LSM-tree). Acta Informatica, 1996,33(4):351-385.

[13] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. TensorFlow: a system for large-scale machine learning. In: USENIX Symp. on Operating Systems Design and Implementation. 2016. 265-283.

[14] LaMotte LR. Fixed-, random-, and mixed-effects models. Encyclopedia of Statistical Sciences. 2014.

[15] Gardiner JC, Luo Z, Roman LA. Fixed effects, random effects and GEE: What are the differences?. Statistics in Medicine, 2009,28(2):221-239.

[16] Crotty A, Galakatos A, Dursun K, Kraska T, Binnig C, Cetintemel U, Zdonik S. An architecture for compiling UDF-centric workflows. Proc. of the VLDB Endowment, 2015,8(12):1466-1477.

[17] Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le QV, Hinton GE, Dean J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In: Int. Conf. on Learning Representations, 2017.

[18] Galakatos A, Markovitch M, Binnig C, Fonseca R, Kraska T. FITing-Tree: A data-aware index structure. In: Int. Conf. on Management of Data. 2019. 1189-1206.

[19] Liu X, Lin Z, Wang H. Novel online methods for time series segmentation. IEEE Trans. on Knowledge and Data Engineering, 2008,20(12):1616-1626.

[20] Xu Z, Zhang R, Kotagiri R, Parampalli U. An adaptive algorithm for online time series segmentation with error bound guarantee. In: Proc. of the Int. Conf. on Extending Database Technology. 2012. 192-203.

[21] Ding J, Minhas UF, Yu J, Wang C, Do J, Li Y. ALEX: An updatable adaptive learned index. In: Int. Conf. on Management of Data. 2020. 969-984.

[22] Bender MA, Hu H. An adaptive packed-memory array. ACM Trans. on Database Systems, 2007,32(4).

[23] Li P, Hua Y, Zuo P, Jia J. A scalable learned index scheme in storage systems. arXiv: Databases, 2019.

[24] Li X, Li J, Wang X. ASLM: Adaptive single layer model for learned index. In: Int. Conf. on Database Systems for Advanced Applications. 2019. 80-95.

[25] Ferragina P, Vinciguerra G. The PGM-index: A fully-dynamic compressed learned index with provable worst-case bounds. Proc. of the VLDB Endow, 2020,13(8):1162-1175.

[26] Orourke J. An on-line algorithm for fitting straight lines between data ranges. Communications of the ACM, 1981,24(9):574-578.

[27] Buragohain C, Shrivastava N, Suri S. Space efficient streaming algorithms for the maximum error histogram. In: Int. Conf. on Data Engineering. 2007. 1026-1035.

[28] Chen DZ, Wang H. Approximating points by a piecewise linear function. Algorithmica, 2013,66(3):682-713.

[29] Chen Q, Chen L, Lian X, Liu Y, Yu JX. Indexable PLA for efficient similarity search. In: Proc. of the Int. Conf. on Very Large Data Bases. 2007. 435-446.

[30] Xie Q, Pang C, Zhou X, et al. Maximum error-bounded piecewise linear representation for online stream approximation. The VLDB Journal, 2014,23(6):915-937.

[31] Overmars MH. The design of dynamic data structures. Lecture Notes in Computer Science, vol.156. Springer. 1983.

[32] Moffat A and Turpin A. Compression and Coding Algorithms. Springer. 2002.

[33] Navarro G. Compact data structures: A practical approach. Cambridge University Press. 2016.

[34] Okanohara D, Sadakane K. Practical entropy-compressed rank/select dictionary. In: Proc. of the Workshop on Algorithm Engineering and Experimentation. 2007. 60-70.

[35] Orourke J. An on-line algorithm for fitting straight lines between data ranges. Communications of the ACM, 1981,24(9):574-578.

[36] Neumann T, Michel S. Smooth interpolating histograms with error guarantees. In: British National Conf. on Databases. 2008. 126-138.

[37] Kipf A, Marcus R, Renen AV, Stoian M, Kemper A, Kraska T, et al. RadixSpline: a single-pass learned index. In: Proc. of the Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management. 2020.

[38] Brisebarre N, Joldes M. Chebyshev interpolation polynomial-based tools for rigorous computing. In: Proc. of the Int. Symp. on Symbolic and Algebraic Computation. 2010. 147-154.

[39] Alda F, Rubinstein BI. The Bernstein mechanism: Function release under differential privacy. In: Proc. of the AAAI Conf. on Artificial Intelligence. 2017. 1705-1711.

[40] Setiawan NF, Rubinstein BI, Borovicagajic R. Function interpolation for learned index structures. In: Proc. of the Australasian Database Conference. 2020. 68-80.

[41] Tang C, Wang Y, Hu G, Dong Z, Wang Z, Wang M. XIndex: A scalable learned index for multicore data storage. In: ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. 2020. 308-320.

[42] Bronson N, Casper J, Chafi H, Olukotun K. A practical concurrent binary search tree. ACM SIGPLAN Notices, 2010,45(5):257-268..

[43] Cha SK, Hwang S, Kim K, Kwon K. Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems. In: Proc. of the Int. Conf. on Very Large Data Bases. 2001. 181-190.

[44] Mao Y, Kohler E, Morris R. Cache craftiness for fast multicore key-value storage. In: European Conf. on Computer Systems. 2012. 183-196.

[45] Binna R, Zangerle E, Pichl M, Specht G, Leis V. HOT: A height optimized trie index for main-memory database systems. In: Proc. of the Int. Conf. on Management of Data. 2018. 521-534.

[46] Wu X, Ni F, Jiang S. Wormhole: A fast ordered index for in-memory data management. In: European Conf. on Computer Systems. 2019.

[47] McKenney PE, Appavoo J, Kleen A, Krieger O, Russell R, Sarma D, et al. Read-copy update. In: Proc. of the AUUG Conference. 2001.

[48] Gao YN, Ye JB, Yang NZ, Gao XF, Chen GH. A middle layer based scalable learned index scheme. Ruan Jian Xue Bao/Journal of Software, 2020,31(3):620-633 (in Chinese).

[49] Llaveshi A, Sirin U, Ailamaki A, West R. Accelerating B+tree search by using simple machine learning techniques. In: Int. Workshop on Applied AI for Database Systems and Applications. 2019.

[50] Hadian A, Heinis T. Interpolation-friendly B-trees: Bridging the gap between algorithmic and learned indexes. In: Proc. of the Int. Conf. on Extending Database Technology. 2019. 710-713.

[51] Qu W, Wang X, Li J, Li X. Hybrid indexes by exploring traditional B-Tree and linear regression. In: Int. Conf. on Web Information Systems and Applications. 2019. 601-613.

[52] Wu Y, Yu J, Tian Y, Sidle RS, Barber RJ. Designing succinct secondary indexing mechanism by exploiting column correlations. In: Proc. of the Int. Conf. on Management of Data. 2019. 1223-1240.

[53] Ilyas IF, Markl V, Haas PJ, Brown P, Aboulnaga A. CORDS: Automatic discovery of correlations and soft functional dependencies. In: Proc. of the Int. Conf. on Management of Data. 2004. 647-658.

[54] Kimura H, Huo G, Rasin A, Madden S, Zdonik SB. Correlation maps: A compressed access method for exploiting soft functional dependencies. Proc. of the VLDB Endowment, 2009,2(1):1222-1233.

[55] Hadian A, Heinis T. Considerations for handling updates in learned index structures. In: Proc. of the Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2019.

[56] Bilgram R. Cost models for learned index with insertions. AAU CS Master Thesis. 2019.

[57] Ooi BC, Sacks-davis R, Han J. Indexing in Spatial Databases. CiteSeerX. 2007.

[58] Singh HS, Bawa S. A survey of traditional and MapReduceBased spatial query processing approaches. SIGMOD Record, 2017,46(2):18-29.

[59] Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975,18(9):509-517.

[60] Finkel RA, Bentley JL. Quad trees: A data structure for retrieval on composite keys. Acta Informatica, 1974,4(1):1-9.

[61] Meagher D. Geometric modeling using octree encoding. Computer Graphics and Image Processing, 1982,19(2):129-147.

[62] Guttman A. R-trees: A dynamic index structure for spatial searching. In: Proc. of the Int. Conf. on Management of Data. 1984. 47-57.

[63] Beckmann N, Kriegel H, Schneider R, Seeger B. The R*-tree: An efficient and robust access method for points and rectangles. In: Proc. of the Int. Conf. on Management of Data. 1990. 322-331.

[64] Kamel I, Faloutsos C. Hilbert R-tree: An improved R-tree using fractals. In: Proc. of the Int. Conf. on Very Large Data Bases. 1994. 500-509.

[65] Sellis T, Roussopoulos N, Faloutsos C. The R+-Tree: A dynamic index for multi-dimensional objects. In: Proc. of the Int. Conf. on Very Large Data Bases. 1987. 507-518.

[66] Sagan H. Space-filling curves. Springer. 1994.

[67] Ramsak F, Markl V, Fenk R, Zirkel M, Elhardt K, Bayer R. Integrating the UB-Tree into a database system kernel. In: Proc. of the Int. Conf. on Very Large Data Bases. 2000. 263-272.

[68] Kraska T, Alizadeh M, Beutel A, Chi EH, Ding J, Kristo A, et al. SageDB: A learned database system. In: Biennial Conf. on Innovative Data Systems Research. 2019.

[69] Wang H, Fu X, Xu J, Lu H. Learned index for spatial queries. In: Proc. of the IEEE Int. Conf. on Mobile Data Management. 2019. 569-574.

[70] Davitkova A, Milchevski E, Michel S. The ML-Index: A multidimensional, learned index for point, range, and nearest-neighbor queries. In: Proc. of the Int. Conf. on Extending Database Technology. 2020. 407-410.

[71] Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans. on Database Systems, 2005,30(2):364-397.

[72] Schuh MA, Wylie T, Liu C, Angryk RA. Approximating high-dimensional range queries with kNN indexing techniques. In: Proc. of the Int. Conf. on Computing and Combinatorics. 2014. 369-380.

[73] Li P, Lu H, Zheng Q, Yang L, Pan G. LISA: A learned index structure for spatial data. In: Proc. of the Int. Conf. on Management of Data. 2020. 2119-2133.

[74] Royden HL, Fitzpatrick PM. Real Analysis, Fourth Edition. Prentice Hall. 2010.

[75] Garcia EK, Gupta MR. Lattice regression. In: Proc. of the Annual Conf. on Neural Information Processing Systems. 2009. 594-602.

[76] Nathan V, Ding J, Alizadeh M, Kraska T. Learning multi-dimensional indexes. In: Proc. of the Int. Conf. on Management of Data. 2020. 985-1000.

[77] Breiman L. Random forests. Machine Learning, 2001,45(1):5-32.

[78] Nievergelt J, Hinterberger H, Sevcik KC. The grid file: An adaptable, symmetric multikey file structure. ACM Trans. on Database Systems, 1984,9(1):38-71.

[79] Gaede V, Gunther O. Multidimensional access methods. ACM Computing Surveys, 1998,30(2):170-231.

[80] Athanassoulis M, Bogh K S, Idreos S. Optimal column layout for hybrid workloads. Proc. of the VLDB Endowment, 2019,12(13):2393-2407.

[81] Yang Z, Chandramouli B, Wang C, Gehrke J, Li Y, Minhas UF, et al. Qd-tree: Learning data layouts for big data analytics. In: Proc. of the Int. Conf. on Management of Data. 2020. 193-208.

[82] Dietzfelbinger M, Karlin AR, Mehlhorn K, Der Heide FM, Rohnert H, Tarjan RE. Dynamic perfect hashing: Upper and lower bounds. SIAM Journal on Computing, 1994,23(4):738-761.

[83] Pagh R, Rodler FF. Cuckoo hashing. Journal of Algorithms, 2004,51(2):122-144.

[84] Indyk P, Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proc. of the Annual ACM Symp. on the Theory of Computing. 1998. 604-613.

[85] Smeulders AW, Worring M, Santini S, Gupta A, Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2000,22(12):1349-1380.

[86] Cayton L, Dasgupta S. A learning framework for nearest neighbor search. In: Proc. of the Annual Conf. on Neural Information Processing Systems. 2007. 233-240.

[87] He J, Chang S, Radhakrishnan R, Bauer C. Compact hashing with joint optimization of search accuracy and time. In: IEEE Conf. on Computer Vision and Pattern Recognition. 2011. 753-760.

[88] Torralba A, Fergus R, Freeman WT. 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2008,30(11):1958-1970.

[89] Wang J, Kumar S, Chang S. Semi-supervised hashing for scalable image retrieval. In: IEEE Conf. on Computer Vision and Pattern Recognition. 2010. 3424-3431.

[90] Norouzi M, Blei DM. Minimal loss hashing for compact binary codes. In: Proc. of the Int. Conf. on Machine Learning. 2011. 353-360.

[91] Xu H, Wang J, Li Z, Zeng G, Li S, Yu N. Complementary hashing for approximate nearest neighbor search. In: IEEE Int. Conf. on Computer Vision. 2011. 1631-1638.

[92] Wang J, Liu W, Kumar S, Chang SF. Learning to hash for indexing big data - A survey. Proc. of the IEEE, 2016,104(1):34-57.

[93] Guo J, Zhang S, Li J. Hash learning with convolutional neural networks for semantic based image retrieval. In: Porc. of the Pacific-Asia Conf. on Knowledge Discovery and Data Mining. 2016. 227-238.

[94] Moffat A, Zobel J. Self-indexing inverted files for fast text retrieval. ACM Trans. on Information Systems, 1996,14(4):349-379.

[95] Zobel J, Moffat A. Inverted files for text search engines. ACM Computing Surveys, 2006,38(2).

[96] Oosterhuis H, Culpepper JS, De Rijke M. The potential of learned index structures for index compression. In: Proc. of the Australasian Document Computing Symposium. 2018. 1-4.

[97] Rossi C, De Moura ES, Carvalho A, Silva AS. Fast document-at-a-time query processing using two-tier indexes. In: Proc. of the Int. ACM SIGIR Conf. on Research and Development in Information Retrieval. 2013. 183-192.

[98] Goodwin B, Hopcroft M, Luu D, Clemmer A, Curmei M, Elnikety S, et al. BitFunnel: Revisiting signatures for search. In: Proc. of the Int. ACM SIGIR Conf. on Research and Development in Information Retrieval. 2017. 605-614.

[99] Xiang W, Zhang H, Cui R, Chu X, Li K, Zhou W. Pavo: A RNN-based learned inverted index, supervised or unsupervised?. IEEE Access, 2019,7:293-303.

[100] Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970,13(7):422-426.

[101] Chang FW, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, et al. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems, 2008,26(2):4-26.

[102] Athanassoulis M, Ailamaki A. BF-tree: Approximate tree indexing. Proc. of the VLDB Endowment, 2014,7(14):1881-1892.

[103] Geravand S, Ahmadi M. Bloom filter applications in network security: A state-of-the-art survey. Computer Networks, 2013,57(18):4047-4064.

[104] Mitzenmacher M. A model for learned bloom filters and optimizing by sandwiching. In: Proc. of the Annual Conf. on Neural Information Processing Systems. 2018. 462-471.

[105] Dai Z, Shrivastava A. Adaptive learned bloom filter (Ada-BF): Efficient utilization of the classifier. arXiv: Data Structures and Algorithms, 2019.

[106] Santoro A, Bartunov S, Botvinick M, Wierstra D, Lillicrap T. Meta-learning with memory-augmented neural networks. In: Proc. of the Int. Conf. on Machine Learning. 2016. 1842-1850.

[107] Vinyals O, Blundell C, Lillicrap T, Kavukcuoglu K, Wierstra D. Matching networks for one shot learning. In: Proc. of the Annual Conf. on Neural Information Processing Systems. 2016. 3637-3645.

[108] Rae JW, Bartunov S, Lillicrap T. Meta-learning neural bloom filters. In: Proc. of the Int. Conf. on Machine Learning. 2019. 5271-5280.

[109] Fan B, Andersen DG, Kaminsky M, Mitzenmacher M. Cuckoo filter: Practically better than bloom. In: Proc. of the ACM Int. Conf. on Emerging Networking Experiments and Technologies. 2014. 75-88.

[110] Bhattacharya A, Bedathur S, Bagchi A. Adaptive learned bloom filters under incremental workloads. In: Proc. of the ACM India Joint Int. Conf. on Data Science and Management of Data. 2020. 107-115.

[111] Negi S, Dubey A, Bagchi A, Yadav M, Yadav N, Raj J. Dynamic partition bloom filters: A bounded false positive solution for dynamic set membership. arXiv: Data Structures and Algorithms, 2019.

[112] Kipf A, Marcus R, Van Renen A, Stoian M, Kemper A, Kraska T, et al. SOSD: A benchmark for learned indexes. arXiv: Databases, 2019.

[113] Van Sandt P, Chronis Y, Patel JM. Efficiently searching in-memory sorted arrays: Revenge of the interpolation search?. In: Proc. of the Int. Conf. on Management of Data. 2019. 36-53.

[114] Leis V, Kemper A, Neumann T. The adaptive radix tree: ARTful indexing for main-memory databases. In: Proc. of the Int. Conf. on Management of Data. 2013. 38-49.

[115] STX B+-Tree. https://panthema.net/2007/stx-btree/

[116] Kim C, Chhugani J, Satish N, Sedlar E, Nguyen AD, Kaldewey T, et al. FAST: Fast architecture sensitive tree search on modern CPUs and GPUs. In: Proc. of the Int. Conf. on Management of Data. 2010. 339-350.

[117] Amazon sales rank data for print and kindle books. https://www.kaggle.com/ucffool/amazon-sales-rank-data-for-print-and-kindle-books

[118] Gjoka M, Kurant M, Butts CT, Markopoulou A. Walking in Facebook: A case study of unbiased sampling of OSNs. In: IEEE Int. Conf. on Computer Communications. 2010. 2498-2506.

[119] OpenStreetMap database. https://aws.amazon.com/public-datasets/osm

[120] Wikimedia downloads. http://dumps.wikimedia.org

[121] Gupta P, Carey MJ, Mehrotra S, Yus R. SmartBench: A benchmark for data management in smart spaces. Proc. of the VLDB Endowment, 2020,13(11):1807-1820.

[122] TPC-C. http://www.tpc.org/tpcc/

[123] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In: Proc. of the ACM Symp. on Cloud Computing. 2010. 143-154.

[124] Deng J, Dong W, Socher R, Li L, Li K, Li F. ImageNet: A large-scale hierarchical image database. In: IEEE Conf. on Computer Vision and Pattern Recognition. 2009. 248-255.

[125] TPC-H. http://www.tpc.org/tpch/

[126] Marcus R, Zhang E, Kraska T. CDFShop: Exploring and optimizing learned index structures. In: Proc. of the Int. Conf. on Management of Data. 2020. 2789-2792.

[127] Lu H, Dai Y, Shen C, Xu S. Indices matter: Learning to index for deep image matting. In: IEEE Int. Conf. on Computer Vision. 2019. 3266-3275.

[128] Ho D, Ding J, Misra S, Tatbul N, Nathan V, Vasimuddin, et al. LISA: Towards learned DNA sequence search. arXiv: Databases, 2019.

[129] Ferragina P, Manzini G. Opportunistic data structures with applications. In: Annual Symp. on Foundations of Computer Science. 2000. 390-398.

[130] Kirsche M, Das A, Schatz MC. Sapling: Accelerating suffix array queries with learned data models. bioRxiv, 2020.

[131] Kristo A, Vaidya K, Çetintemel U, Misra S, Kraska T. The case for a learned sorting algorithm. In: Proc. of the Int. Conf. on Management of Data. 2020. 1001-1016.

[132] Idreos S, Kersten ML, Manegold S. Database cracking. In: Biennial Conf. on Innovative Data Systems Research. 2007. 68-78.

[133] Vienne J, Chen J, Wasiurrahman M, Islam NS, Subramoni H, Panda DK. Performance analysis and evaluation of InfiniBand FDR and 40GigE RoCE on HPC and cloud computing systems. In: IEEE Annual Symp. on High-Performance Interconnects. 2012. 48-55.

[134] Mitchell C, Montgomery K, Nelson L, Sen S, Li J. Balancing CPU and network in the cell distributed B-tree store. In: USENIX Annual Technical Conference. 2016. 451-464.

[135] Ziegler T, Vani ST, Binnig C, Fonseca R, Kraska T. Designing distributed tree-based index structures for fast RDMA-capable networks. In: Proc. of the Int. Conf. on Management of Data. 2019. 741-758.

[136] Mitchell C, Geng Y, Li J. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In: USENIX Annual Technical Conference. 2013. 103-114.

[137] Dragojevic A, Narayanan D, Hodson O, Castro M. FaRM: Fast remote memory. In: Proc. of the USENIX Symp. on Networked Systems Design and Implementation. 2014. 401-414.

[138] Wang YE, Wei G, Brooks D. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv: Learning, 2019.

[139] Yang J, Kim J, Hoseinzadeh M, Izraelevitz J, Swanson S. An empirical guide to the behavior and use of scalable persistent memory. In: USENIX Conf. on File and Storage Technologies. 2020. 169-182.

[140] Oukid I, Lasperas J, Nica A, Willhalm T, Lehner W. FPTree: A hybrid SCM-DRAM persistent and concurrent B-Tree for storage class memory. In: Proc. of the Int. Conf. on Management of Data. 2016. 371-386.

[141] Hwang D, Kim W, Won Y, Nam B. Endurable transient inconsistency in byte-addressable persistent B+-Tree. In: USENIX Conf. on File and Storage Technologies. 2018. 187-200.

[142] Arulraj J, Levandoski JJ, Minhas UF, Larson P. Bztree: A high-performance latch-free range index for non-volatile memory. Proc. of the VLDB Endowment, 2018,11(5):553-565.

[143] Liu J, Chen S, Wang L: LB+-Trees: Optimizing persistent index performance on 3DXPoint memory. Proc. of the VLDB Endowment, 2020,13(7):1078-1090.

[144] Lee S K, Lim KH, Song H, Nam B, Noh SH. WORT: Write optimal radix tree for persistent memory storage systems. In: USENIX Conf. on File and Storage Technologies. 2017. 257-270.

[145] Zuo P, Hua Y, Wu J. Write-optimized and high-performance hashing index scheme for persistent memory. In: USENIX Symp. on Operating Systems Design and Implementation. 2018. 461-476.

[146] Nam M, Cha H, Choi Y, Noh SH, Nam B. Write-optimized dynamic hashing for persistent memory. In: USENIX Conf. on File and Storage Technologies. 2019. 31-44.

[147] Lu B, Hao X, Wang T, Lo E: Dash: Scalable hashing on persistent memory. Proc. of the VLDB Endowment, 2020,13(8):1147-1161.

[148] Zhou X, Shou L, Chen K, Hu W, Chen G. DPTree: Differential indexing for persistent memory. Proc. of the VLDB Endowment, 2019,13(4):421-434.

附中文参考文献:

[8] 孟小峰,马超红,杨晨.机器学习化数据库系统研究综述,计算机研究与发展,2019,56(9):1803−1820.

[9] 李国良,周煊赫,孙佶,余翔,袁海涛,刘佳斌,韩越.基于机器学习的数据库技术综述.计算机学报,2019,(42),论文在线号No.120:1−33.

[10] 柴茗珂,范举,杜小勇.学习式数据库系统:挑战与机遇.软件学报,2020,31(3):806−830.

[11] 孙路明,张少敏,姬涛,李翠平,陈红.人工智能赋能的数据管理新技术研究.软件学报,2020,31(3):600−619.

[48] 高远宁,叶金标,杨念祖,高晓泓,陈贵海.基于中间层的可扩展的学习索引技术.软件学报,2020,31(3):620−633.