软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(2):485−507 [doi: 10.13328/j.cnki.jos.007570] [CSTR: 32375.14.jos.007570]

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

LA-tree: 查询感知的自适应学习型多维索引*

刘佳伟[1,2], 范 举[2,1], 张 超[1], 杜小勇[1,2]

[1](中国人民大学 信息学院, 北京 100872)

[2](数据工程与知识工程教育部重点实验室 (中国人民大学), 北京 100872)

通信作者: 范举, E-mail: fanj@ruc.edu.cn

摘 要: 结构化数据分析通常需要在表格数据的多维属性上执行联合范围查询, 高效的多维索引因此成为数据库系统的关键支撑. 然而, 现有多维索引方法在高维场景下存在局限: 传统多维索引仅按数据分布进行均匀划分, 缺乏对查询特征的感知, 导致筛选效果有限; 而现有学习型多维索引虽引入查询感知, 但划分往往极不均匀, 使部分单元过大, 扫描成本显著增加. 为了解决上述问题, 提出一种新型的 LA-tree 学习型树形多维索引, 同时兼顾数据分布与查询负载感知. 在离线构建阶段, LA-tree 将节点维度选择建模为最小化查询扫描比的问题, 并提出分层贪心搜索算法, 实现了均匀划分与查询感知的统一. 在在线查询阶段, 引入轻量线性模型与分段线性模型, 将传统的数值比较转化为快速映射计算, 在保证结果完整性的同时显著降低筛选延迟. 在动态场景中, 提出基于扫描量监控的自适应增量更新机制, 通过局部子树重构高效适配数据与查询负载的变化, 避免了整体索引重建的高昂代价. 实验结果表明, LA-tree 在多个真实和基准数据集上均显著优于现有方法: 在静态场景中查询用时较最佳基准方法平均降低 52%, 在动态场景中更新开销较重构方法减少 97%, 同时保持低查询延迟与轻量级索引规模.

关键词: 学习型多维索引; 查询感知; 索引更新

中图法分类号: TP311

中文引用格式: 刘佳伟, 范举, 张超, 杜小勇. LA-tree: 查询感知的自适应学习型多维索引. 软件学报, 2026, 37(2): 485–507. http://www.jos.org.cn/1000-9825/7570.htm

英文引用格式: Liu JW, Fan J, Zhang C, Du XY. LA-tree: Query-aware Adaptive Learned Multi-dimensional Index. Ruan Jian Xue Bao/Journal of Software, 2026, 37(2): 485–507 (in Chinese). http://www.jos.org.cn/1000-9825/7570.htm

LA-tree: Query-aware Adaptive Learned Multi-dimensional Index

LIU Jia-Wei[1,2], FAN Ju[2,1], ZHANG Chao[1], DU Xiao-Yong[1,2]

[1](School of Information, Renmin University of China, Beijing 100872, China)

[2](Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University), Ministry of Education, Beijing 100872, China)

Abstract: Structured data analysis typically requires performing multi-attribute queries over tabular data, making efficient multi-dimensional indexes key support for database systems. However, existing multi-dimensional indexing methods face limitations in high-dimensional scenarios. Traditional multi-dimensional indexing methods partition data uniformly based on data distribution but lack the awareness of query features, resulting in limited filtering effectiveness. In contrast, although existing learned multi-dimensional indexes introduce query-awareness, they often produce highly unbalanced partitions, thereby resulting in some oversized partitions and substantially increased scanning costs. To this end, this study proposes LA-tree, a novel learned tree-based multi-dimensional index that balances both data distribution and query workload awareness. In the offline construction phase, LA-tree formulates the selection of partitioning dimensions at each node as an optimization problem of minimizing the overall scan ratio of the query workload, and puts forward a hierarchical greedy search algorithm to achieve the unity of uniform partitioning and query-awareness. In the online query phase, the

* 基金项目: 国家自然科学基金 (62436010, 62502520); 北京市自然科学基金-海淀原始创新联合基金 (L222006)
收稿时间: 2025-09-01; 修改时间: 2025-10-07; 采用时间: 2025-10-30; jos 在线出版时间: 2025-12-10
CNKI 网络首发时间: 2025-12-11

lightweight linear model and piecewise linear model are introduced to transform traditional numerical comparisons to fast mapping computations, thereby reducing filtering latency while ensuring the completeness of query results. In dynamic settings, an adaptive incremental update mechanism based on scan volume monitoring is proposed to efficiently adapt to changes in data and query workloads via local subtree reconstruction, thereby avoiding the high cost of rebuilding the entire index. Experimental results demonstrate that LA-tree outperforms existing methods on multiple real-world and benchmark datasets. In static settings, the query time is reduced by an average of 52% compared with the optimal benchmark method, while in dynamic settings, the update costs are reduced by 97% compared with the reconstruction methods. Additionally, low query latency and lightweight index scale are maintained.

Key words: learned multi-dimensional index; query-aware; index update

索引结构对数据库查询性能至关重要. 随着数据维度的增加, 数据库查询通常在多个属性上包含等值或范围谓词, 单维索引难以同时利用多维条件, 导致大量无效扫描. 相比之下, 多维索引能够在多个维度上联合组织和过滤数据, 从而显著减少冗余访问并提升查询效率. 因此, 高效的多维索引已成为支持现代数据管理与分析的关键技术.

现有多维数据索引方法, 如 KD-tree[1]、Octree[2]、Qd-tree[3]、Flood[4]和 Tsunami[5], 普遍采用空间划分的方式来提升多维查询性能. 在离线阶段, 这类方法将数据映射到多维空间并划分为若干单元 (cell); 在在线查询阶段, 采用一种“先筛选、后扫描”的框架, 即首先通过多维索引筛选出与查询范围有交集的单元, 再扫描单元内的数据以生成查询结果. 通过在筛选阶段排除大量位于查询范围之外的数据, 这一框架可以有效减少数据扫描量, 从而提升查询效率. 具体而言, 根据拟合数据分布的方法, 现有方法可以分为两类.

- 学习型多维索引 Flood 和 Tsunami 通过网格 (grid)[6]结构拟合多维数据分布. Flood 直接通过学习型网格沿各维度独立地对数据进行划分, 从而建立多维索引. 为了解决多维数据普遍存在的维度间相关性, 需要引入条件分布增强网格以拟合数据的多维联合分布, 然而时间与空间开销关于相关维度数量呈指数级增长. Tsunami 通过 Grid-Tree 结构根据查询负载将数据划分为多个子集, 分别用网格拟合数据分布, 同样无法克服网格拟合多维相关性的弱点. 因此该类方法仅适用于低维数据场景, 在维度较高 (如 10 维以上) 的场景下效率不高.
- 树形多维索引结构 (如 KD-tree、Octree、Qd-tree 等) 通过递归划分数据空间拟合多维数据分布, 将数据组织为层次化的树结构, 不仅能够自适应数据的多维联合分布, 还支持高效的层次化剪枝策略, 因此可以快速排除与查询条件无关的单元. 与基于网格的索引相比, 树形索引在维度的可扩展性和空间利用率上更具潜在优势.

基于此, 本文重点研究基于树形结构的学习型多维索引. 现有研究[3−5]表明, 当前多维索引在查询过程中仍存在大量查询范围之外的数据未能在筛选阶段被有效排除, 从而导致扫描量大、扫描时间长, 成为影响查询效率的核心瓶颈. 因此, 本文聚焦优化多维索引的筛选问题, 具体而言: 一方面在离线索引构建阶段优化数据划分, 另一方面在在线查询中提升筛选效率, 最终提升多维索引的整体性能.

然而, 现有的树形多维索引在筛选效果上仍存在局限性. 传统多维索引结构 KD-tree[1]采用递归的空间均匀划分, 能够在叶子节点上保持数据量基本平衡. 但其划分策略完全独立于查询负载. 然而, 真实情况下, 查询负载并不随机, 而是具有一定的查询模式 (query pattern), 即在多维数据的不同属性维度和范围上冷热不均, KD-tree[1]难以针对查询负载对数据划分进行优化, 导致查询需访问过多单元 (即索引的叶子节点). 与此相对, Qd-tree[3]通过学习查询负载, 在高频查询边界处划分数据, 从而可以避免查询访问过多的单元. 然而, 由于划分仅依赖于查询模式, 而不考虑数据分布, 单元间数据量往往极不均衡, 部分单元可能包含大量数据, 依然增加了数据扫描量. 由此可见, 现有树形多维索引方法难以兼顾“均匀划分”与“查询感知”.

针对上述问题, 本文提出了一种新型的学习型多维索引 LA-tree, 在均匀划分的基础上引入查询感知优化, 从而兼顾了数据分布平衡与查询模式适应性. 图 1 给出了 LA-tree 与现有树形多维索引方法的对比示意: 下层 3 幅图展示了在给定多维数据 (灰色散点) 与查询负载训练集 (绿色方框) 上的离线数据划分结果; 上层 3 幅图则对比了在线查询阶段的筛选效果. 结果显示, KD-tree (图 1(a)) 因缺乏查询感知, 筛选不充分, 需扫描的大量灰色区域导致扫描量偏高; Qd-tree (图 1(b)) 虽能针对查询模式优化, 但划分极不均匀. 尽管查询需要访问的单元个数减少, 但需在包含大量蓝色散点的单元中扫描, 依然导致较大的扫描量; 而本文提出的 LA-tree (图 1(c)) 在保持均匀划分的同时融入查询感知优化, 有效减少了需扫描的单元与数据量, 显著提升了查询效率.

为实现高效的多维数据查询, 本文围绕 LA-tree 索引面临的 3 个关键技术挑战进行研究.

挑战 1: 离线索引构建. 在树形索引递归划分多维数据时, 需要同时保证划分的均匀性与查询感知, 这一过程因数据分布与查询负载的复杂性而极具挑战. 为此, 本文将问题建模为节点划分维度选择优化, 提出多层次查询感知的数据划分方法: 通过高效估计不同维度对查询扫描比的影响, 并结合多级搜索贪心算法, 自顶向下地生成树形划分结构, 从而在保证均匀性的同时提升查询性能.

挑战 2: 在线查询处理. 在线查询阶段的核心在于快速且准确地定位与查询范围相交的单元, 并高效完成筛选. 然而, 树结构往往包含大量节点, 叶子节点对应的单元也存储多条数据, 传统依赖数值比较的方法会导致计算开销过大. 为此, 本文提出基于学习模型的高效在线筛选方法: 在中间节点与叶子节点分别引入轻量化模型, 实现查询边界到数据位置的快速映射, 从而有效减少扫描量并避免频繁的数值比较. 同时, 通过误差界限约束保证模型筛选的正确性, 确保筛选与扫描结果的完整性.

挑战 3: 动态更新. 在动态场景下, 数据更新会破坏索引划分的均匀性, 而查询负载的变化也可能使原有的查询感知划分失效. 直接重建索引虽能恢复性能, 但开销过高, 不适合频繁更新. 为此, 本文提出自适应增量更新方法: 通过轻量化模型快速定位更新数据的位置, 实现低开销的增量更新. 同时, 实时监控数据分布和查询负载变化对扫描量的影响, 自动触发局部子树重构, 从而在无需整体重建的情况下持续保持低查询延迟.

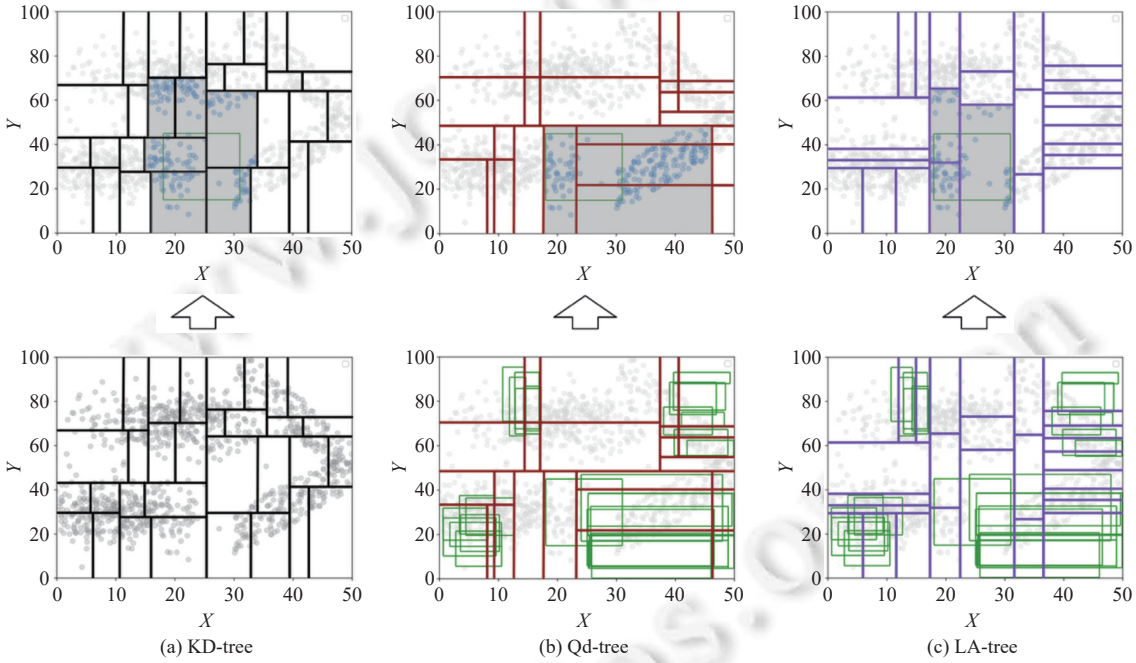


图 1　多维索引不同的数据划分方式对查询性能的影响

总结起来, 本文的主要贡献如下.

(1) 提出 LA-tree 索引结构. 设计了一种基于学习型空间划分多叉树结构的多维索引 LA-tree, 在同一框架下实现了“均匀划分”与“查询感知”, 解决了传统 KD-tree 与 Qd-tree 在数据划分上的局限性.

(2) 针对 LA-tree 设计了高效的算法. 分别设计了多层次查询感知的数据划分方法、基于学习模型的高效在线筛选方法, 以及自适应增量更新方法, 有效地解决了离线索引构建、在线查询处理和动态更新这 3 大挑战.

(3) 在多种数据集和查询负载下进行了充分的实验. 在权威的基准数据集上, LA-tree 在静态场景下平均查询用时相比已有最佳方法减少约 52%, 且在高维查询下优势更加显著. 在动态场景中, 自适应增量更新方法使索引更新用时相比定期重构减少 97%, 同时保持低查询延迟和较小的索引规模, 验证了 LA-tree 在性能上的优势.

本文第 1 节给出问题定义, 介绍多维查询与学习型索引的基本概念. 第 2 节描述 LA-tree 的索引结构与整体算法框架. 第 3 节提出多层次查询感知的数据划分方法. 第 4 节提出基于学习模型的高效在线筛选方法. 第 5 节进一步探讨自适应增量更新机制, 以应对动态场景下的数据和查询负载变化. 在第 6 节中, 我们通过多种数据集和查询负载的实验, 评估了 LA-tree 在静态与动态环境下的性能表现. 第 7 节回顾相关研究工作. 最后, 第 8 节对全文进行总结.

1 问题定义

本文针对多维结构化数据上的范围/等值查询, 研究高效多维索引的构建与查询处理问题. 本节将介绍所需的基本概念, 并给出问题的形式化定义.

● 多维数据: 根据多维数据模型[7,8]定义多维数据, 数据表包含多行多列 (属性), 每行代表一条数据记录, 对应一个多维空间内的点; 每列代表一个属性, 对应多维空间的一个维度. 例如, TPC-H 交易订单数据表 Lineitem 以价格、货量、折扣等多个维度描述交易数据. 形式化地, 考虑一个 n 行 d 列的数据表 T, 包含 n 条数据记录 $T=\{t_1,t_2,\ldots,t_n\}$, 以及 d 个维度 $A=\{a_1,a_2,\ldots,a_d\}$, 每条记录可以表示为一个 d 维向量 $t_l=\langle v_{t_l,a_1},v_{t_l,a_2},\ldots,v_{t_l,a_d}\rangle$, 而对数据表 T 取单独一维度 a_j 投影, 即为一大小为 n 的数组 $T^{a_j}=\left[v_{t_1,a_j},v_{t_2,a_j},\ldots,v_{t_n,a_j}\right]$.

● 多维查询: 在多维数据的基础上定义多维查询. 用户经常从多个角度同时查询多维数据, 例如, 对于 TPC-H 交易订单事实数据表 Lineitem, 筛选出同时满足价格小于等于 2 000 000、货量大于等于 10、折扣小于等于 5 的订单, 即是一条典型的多维查询, “SELECT * FROM LINEITEM *T* WHERE L_EXTENDEDPRICE<=2000000 AND L_QUANTITY>=10 AND L_DISCOUNT<=5;”. 本文聚焦多维数据上的范围查询, 也可很自然地推广至等值查询. 形式化地, 定义多维查询, 一条同时包含多个范围约束谓词 (涵盖点查询) 的 SQL 查询形如:

SELECT * FROM T WHERE P_1 AND P_2 AND … AND P_p;

其中, 每个范围约束谓词 P_x 作用在某个维度 (如 a_j) 上: $lb_x\leqslant a_j\leqslant ub_x$. SQL 查询的谓词也可以拓展到含 OR 逻辑连词的情形, 可以被转化为析取范式加以处理. 本文遵循多维索引研究的惯例, 为更好突出多维属性范围查询, 针对谓词仅包含 AND 逻辑连词的查询展开讨论. 注意到多条范围约束可能作用在同一维度上, 显然可以合并这些谓词, 最终将多维属性范围查询描述为多维数据 T 对应的有限多维空间中的一个超立方体范围: $q:[lb_{q,a_1},ub_{q,a_1}]\wedge[lb_{q,a_2},ub_{q,a_2}]\wedge\ldots\wedge[lb_{q,a_d},ub_{q,a_d}]$, 其中每个范围 $[lb_{q,a_j},ub_{q,a_j}]$ 的两端也可以闭区间或开区间. 特别地, 若查询 q 在某维度 a_j 上没有任何约束条件, 则相应范围约束等价于该维度值域范围 $[\min\{T^{a_j}\},\max\{T^{a_j}\}]$.

● 查询负载: 在查询定义的基础上, 定义查询负载 $Q=\{q_1,q_2,\ldots,q_{|Q|}\}$ 为查询的集合. 通常, 一个查询负载 Q 中的查询并非完全随机的, 而是具有一定的分布特征, 比如每条查询谓词的维度组合服从一定的分布, 以及每条查询对应超立方体的位置和大小服从一定的分布.

● 问题定义: 最后给出多维索引功能, 即检索查询结果的形式化定义. 在数据 T 上在线查询 q, 索引 $I_T(q)$ 返回查询范围内的所有数据条目行号集合, 即:

$$I_T(q)=\left\{i\mid lb_{q,a_j}\leqslant v_{t_i,a_j}\leqslant ub_{q,a_j},t_i\in T,\forall a_j\in A\right\}\tag{1}$$

利用多维索引进行查询处理时, 一般包括两个基本的步骤.

● 步骤 1: 筛选. 根据查询谓词约束, 过滤掉不相关的数据, 留下可能满足查询条件的候选扫描集.

● 步骤 2: 扫描. 将候选扫描集中的每一条数据与查询谓词进行对比, 得出最终的查询结果.

这里, 不妨将索引的筛选和扫描步骤输出的结果分别表示为两个单独的集合 $I'_T(q)$ 和 $I''_T(q)$, 得到:

$$I'_T(q)=\left\{i\mid v_{t_i,a_j}\in R_q,t_i\in T,\forall a_j\in A\right\}\tag{2}$$

其中, R_q 是索引根据查询 q 的谓词范围筛选出的一个更粗略的空间范围, 保证 $q\subseteq R_q$. 结合公式 (1) 和公式 (2) 得到 $I_T(q)=I''_T(q)=\left\{i\mid lb_{q,a_j}\leqslant v_{t_i,a_j}\leqslant ub_{q,a_j},i\in I'_T(q),\forall a_j\in A\right\}$, 其中 R_q 包含 q 越紧, 候选扫描集大小 $\left|I'_T(q)\right|$ 越小, 索引扫描耗时越少, 也即索引筛选效果越好. 这里我们引出度量索引筛选效果的一个重要指标: 扫描比 $ScanRatio(I_T(q))$, 即候选扫描集大小相对最终查询结果集大小的比值. 具体计算如下:

$$ScanRatio(I_T(q)) = \frac{|I'_T(q)|}{|I_T(q)|} \tag{3}$$

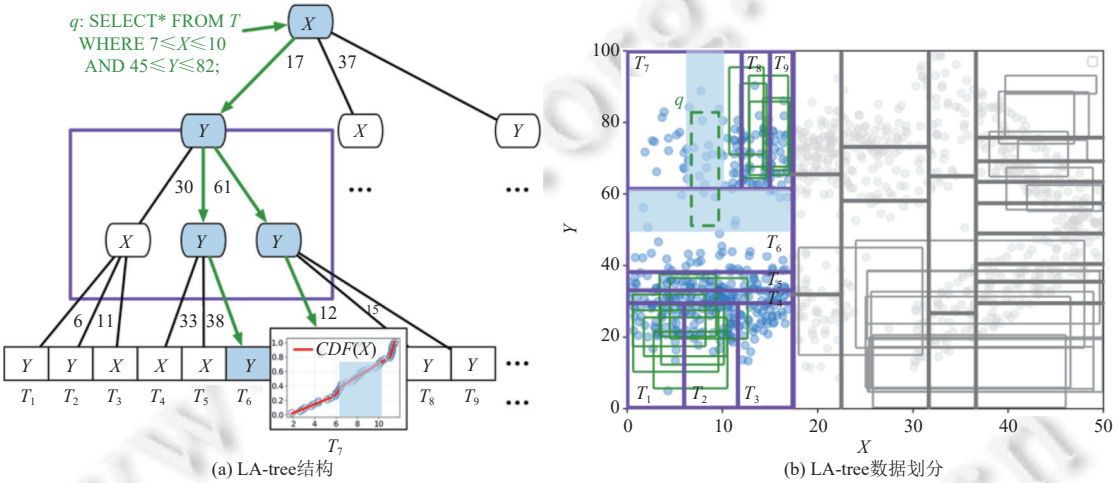
显然, 扫描比大于等于 1, 其数值越小, 说明索引筛选效果越好, 即在相同查询条件下需要扫描的数据量越少.

2 自适应学习型多维索引 LA-tree

针对结构化数据多维查询, 本文提出一种查询感知的学习型多维索引 LA-tree. 本节首先介绍 LA-tree 的基本结构, 随后分别介绍其离线索引构建、在线查询处理和自适应更新问题.

2.1 LA-tree 的基本结构

图 2 为 LA-tree 的基本结构, (a) 为 LA-tree 的基本结构, (b) 为所对应的空间划分情况. 如图 2(a) 所示, LA-tree 采用了与 KD-tree[1]、Qd-tree[3]类似的树形多维索引结构, 其基本结构是一棵空间划分多叉树. 下面从索引功能, 即检索查询结果的角度, 对 LA-tree 的基本结构进行形式化描述, 递归定义各节点的索引功能.



(a) LA-tree结构 (b) LA-tree数据划分

图 2 LA-tree 总览

形式化地, 在公式 (1) 的基础上, 记 LA-tree 上的任意一节点为 N_k, 其对应数据子集为 T_{N_k}. 由于每个节点都可视为一棵 LA-tree 子树的根, 因此其本身可以作为对应数据子集 T_{N_k} 上的索引, 查询结果表示为:

$$I_{T_{N_k}}(q) = \left\{ i \mid lb_{q,a_j} \leqslant v_{t_i,a_j} \leqslant ub_{q,a_j}, t_i \in T_{N_k}, \forall a_j \in A \right\} \tag{4}$$

下面根据节点 N_k 类型的不同, 即中间节点或叶子节点, 给出更为具体的定义.

● 若 N_k 是中间节点, 它将数据按划分维度 a_{N_k} 将数据子集 T_{N_k} 划分为 b 个等量子集, 并将其对应到 N_k 的子节点 $C_{N_k} = \{C_{N_k,1}, C_{N_k,2}, \ldots, C_{N_k,b}\}$ 中, 并产生 $b-1$ 个分位点 $U_{N_k} = \{u_{N_k,1}, \ldots, u_{N_k,b-1}\}$. 显然, 任意查询 q 在 N_k 上的查询结果等于其在所有子节点查询结果的并集, 即有:

$$I_{T_{N_k}}(q) = \bigcup_{lb_{q,a_{N_k}} \leqslant u_{N_k,c} \wedge ub_{q,a_{N_k}} > u_{N_k,c-1}} I_{T_{N_{C_{N_k,c}}}}(q) \tag{5}$$

● 若节点 N_k 为叶子节点, 则对应一个单元 (cell), 其中存储若干数据记录. 通过设置叶子节点来终止递归划分, 可以有效控制树的深度, 避免过多的空间开销与性能消耗.

上述从索引查询结果的角度定义了树形索引各节点的功能, 对于树形索引这一对象本身, 我们定义其为包含所有节点的集合 $I = \{N_1, N_2, \ldots, N_{|I|}\}$. 图 2 展示了一个二维数据空间下的 LA-tree 示例 (为简便起见, 本文以 X 和 Y 表示数据的两个属性维度), 其中, 图 2(a) 是 LA-tree 的结构示意图, 其中浅蓝色节点表示图中在线查询 q 筛选数据时访问的节点; 图 2(b) 则是相应的空间划分结果, 其中蓝色圆形散点表示数据记录, 绿色实线框表示查询负载, 紫色线段表示空间划分, 绿色的虚线框表示在线查询 q, 浅蓝色阴影覆盖扫描的数据. 其中, 浅蓝色阴影范围小于单

元范围, 这是由于 LA-tree 最后一层叶子节点将单元内数据子集根据累积分布函数 (cumulative distribution function, CDF) 进行排序, 以进一步提高筛选效果.

以图 2(a) 中紫色框内的 4 个节点构成的子树为例展开分析. 首先, 根节点选择了维度 X, 并据此将数据均匀划分为 3 份, 得到分位点 17 和 37. 随后, 根节点的第 1 个子节点在其对应 $X \leqslant 17$ 的数据子集上选择 Y 维度将其对应的数据集均匀划分为 3 份, 得到分位点 30 和 61. 需要指出的是, 根节点的其他子节点可以选择不同的维度进行数据划分. 按照上述方式, 数据划分过程自顶向下递归进行, 直至达到叶子节点. 每个叶子节点对应一个单元 (cell), 存储一个数据记录的集合, 如图 2(b) 中数据子集 T_1和T_2 所示. 基于上述 LA-tree 结构, 在给定在线查询 q 时, 索引先自顶向下逐层筛选与查询范围 $X \in [7,10] \wedge Y \in [45,82]$ 有交集的节点 (图 2(a) 中标为浅蓝色); 随后扫描所有标蓝叶子节点对应的单元得到局部查询结果; 最后将这些局部结果合并, 返回最终的查询答案.

LA-tree 各节点维度是查询感知选择的, 以降低查询负载扫描的数据量. 以图 2(a) 紫框内的划分为例, 考虑图 2(b) 紫框内的查询, 可见查询基本聚成维度 Y 查询范围在 60 以上和 40 以下两簇, 因此首先按维度 Y 均匀划分数据. 接下来, 上方的查询维度 X 选择度普遍很低, 因此这部分数据按维度 X 均匀划分. 而下方的查询中, 与数据范围 $Y \in [30,40]$ 部分有交集的查询维度 X 选择度较高, 因此这部分数据按维度 Y 均匀划分, 而 $Y \in [0,30]$ 部分数据则按维度 X 均匀划分.

尽管都采取了自顶向下的空间划分思路, LA-tree 与现有树形多维索引 (如 KD-tree[1]和 Qd-tree[3]) 仍存在显著区别. 与 KD-tree 相比, LA-tree 在保证数据均匀划分的同时, 通过结合查询负载选择划分维度, 引入了查询感知能力, 从而提高了筛选效果. 如图 2(b) 所示, 查询负载中大部分查询都能被单个或少数个单元覆盖, 且每个单元里在查询负载范围以外的数据点很少, 令在线查询扫描比较低, 查询用时短. 而按照图 1(a) 中 KD-tree 的划分方式, 每个查询将扫描较多的单元以及数据. 与 Qd-tree 在高频查询边界处直接切分数据不同, 在每个中间节点, LA-tree 在结合查询负载选择合适的划分维度后, 按照该维度的数据分布将数据均匀地划分为多个子集, 从而保持全局上的数据平衡.

2.2 LA-tree 的离线索引构建

离线索引构建阶段, LA-tree 采取自顶向下的框架递归划分数据. 具体而言, 在每个中间节点, LA-tree 首先依据查询负载选择合适的划分维度, 以保证“查询感知”; 随后在该维度上依据数据分布将数据均匀地划分为多个子集, 以保证“均匀划分”.

形式化地, 本文将该过程建模为一个优化问题: 在给定查询负载 Q 的情况下, LA-tree 需要在每个中间节点选择合适的划分维度, 从而最小化负载中所有查询在索引上的扫描比之和. 例如, 图 2 中紫色框内的节点划分及对应的数据子集划分显示: 当 4 个中间节点选择了合适的划分维度时, 查询负载 (绿色矩形框) 中的大多数查询仅与少量单元相交, 因此查询范围外的数据几乎无需扫描, 查询负载整体的扫描比较低.

解决上述优化问题颇具挑战: 一方面, 即便仅考虑单个节点, 在查询尚未执行之前, 难以直接获知该节点在不同划分方案下对应的扫描比; 另一方面, 每个节点都可从多个候选维度中自由选择, 使得潜在的数据划分方式呈指数级增长, 从而显著加剧了问题的复杂性.

针对上述挑战, 本文提出多层次查询感知的数据划分方法. 首先, 针对单节点维度选择中扫描比难以直接获知的挑战, 构造一个基于扫描比上界的评分函数. 其基本想法是, 在不实际执行查询的前提下, 依据查询负载在候选维度划分后需扫描的子节点所覆盖的数据量, 对各维度进行估计, 从而高效确定单节点的划分维度. 其次, 面向多节点联合划分维度选择, 将问题归约为集合覆盖问题. 由于该问题属于 NP 难问题, 因此提出分段式多级搜索贪心算法: 按深度将树划分为若干段 (每段含多层节点), 段内对多节点维度组合进行有限枚举面向评分函数优化, 段间则采取贪心策略, 最终生成整棵树的数据划分. 该方法兼顾了筛选效果优化的同时, 实现高效的离线索引构建. 有关多层次查询感知的数据划分的详细介绍请参见第 3 节.

2.3 LA-tree 的在线查询处理

在线查询阶段, LA-tree 利用学习模型加速自顶向下的筛选数据. 具体而言, 在中间节点, 预先基于该数据子集在划分维度上的累积分布函数 (CDF) 训练情况模型, 并利用该模型根据查询范围的边界快速确定需访问的子节点. 在叶子节点上, 同样对数据排序并训练模型学习其分布, 使索引能够在筛选的最后一步精确缩小候选扫描集的

范围. 通过这种方式, LA-tree 在保持筛选精度的同时, 避免了传统方法中低效的频繁数值比较, 将数据筛选转化为学习模型“输入查询边界、输出数据位置”的快速计算问题.

例如, 图 2(a) 中根节点的模型能够根据查询 q 的范围 $X \in [7,10]$ 定位到第 1 个子节点 $X \leqslant 17$, 其他中间节点以此类推. 而左起第 7 个叶子节点的模型则可根据查询范围边界缩小候选扫描集, 对应图 2(b) 的 T_7 区域里浅蓝色阴影覆盖的区域, 从而避免扫描大量查询范围之外的数据.

然而, 上述方法面临两个挑战: 一方面, 学习模型本身具有预测误差, 而索引必须保证结果的完整性, 即模型筛选不能遗漏任何满足条件的数据. 另一方面, 常见模型推理耗时较长, 若直接使用反而降低索引性能.

针对上述挑战, 本文提出基于学习模型的高效在线筛选方法. 首先, 为了确保索引筛选结果的完整性, 提出了误差界限约束, 利用保序模型的性质在学习数据分布时计算边界, 保证查询都不遗漏正确结果. 进一步地, 本文设计了中间节点的线性回归模型 (linear model, LM) 与叶子节点分段线性回归模型 (piece-wise linear model, PLM) 相结合的学习型空间划分多叉树结构, 使每个节点都能够在保证正确性的同时快速完成筛选. 有关该方法的技术细节, 详见第 4 节.

2.4 LA-tree 的索引更新

在动态场景下, 面对数据与查询负载的变化, LA-tree 能够相应地更新索引结构, 从而保持索引的准确性和高效性, 而无需整体重建. 具体包括两个方面: (1) 增量更新, 将新数据实时插入到叶子单元中排序后的正确位置, 实现高效维护; (2) 自适应更新, 当检测到局部的数据划分出现不均匀或不再适应查询负载的情况时, 执行局部数据重划分. 例如, 图 2(a) 中的 LA-tree, 利用增量进行更新, 一条更新数据 $X = 10, Y = 70$ 会被准确地插入数据子集 T_7. 利用自适应更新, 假设有大量 $X \leqslant 17, Y > 61$ 范围内的数据更新使局部数据维度 X 和维度 Y 的相关性明显提高, 或涉及该数据范围的大量新查询的谓词仅包含维度 X, 这棵子树都将以新数据子集与查询负载子集重构.

针对这两点, 本文提出自适应增量式的索引更新方法. 将更新的数据视作点查询, 自顶向下利用节点模型准确定位数据位置. 同时, 将数据分布变化和查询负载变化对子树数据划分的影响统一到扫描量即扫描比, 实时比较新查询在更新后数据上的扫描比和访问节点的评分函数值, 找到扫描比大幅增加的节点, 重构相应子树. 有关该方法的技术细节请参见第 5 节.

3 离线索引构建: 多层次查询感知的数据划分方法

本文将查询负载感知的数据划分过程建模为一个优化问题: 在每个中间节点选择合适的划分维度, 以最小化负载中所有查询的扫描比之和. 为解决这一优化问题, 第 3.1、3.2 节将分别探讨两个关键挑战, 即如何在不实际执行查询的情况下估计单节点的扫描比, 以及如何在多层节点上高效选择划分维度.

3.1 单节点划分维度选择

单节点划分维度选择的目标是最小化 Q 中所有查询在数据表 T 上扫描比之和. 根据公式 (3) 定义, 扫描比的分母总是确定的, 最小化扫描比之和等价于最小化扫描量, 即根据公式 (4), 优化目标为:

$$\min_{a_{N_k} \in A} \sum\nolimits_{q \in Q_{N_k}} \left| I'_{T_{N_k}}(q) \right| \tag{6}$$

由于索引离线构建阶段不执行查询, 节点不同划分维度带来的扫描比变化无法直接衡量. 考虑中间节点结果公式 (5), 我们提出优化比 $OPR_{T,Q}(a)$, 通过数据总量与按某维度划分查询负载所有查询需访问的子节点覆盖数据量之和的比, 估计扫描比之和的上界, 如下:

$$\min_{a \in A} OPR_{T,Q}(a) = \min_{a \in A} \frac{1}{|T||Q|} \sum_{1 \leqslant c \leqslant b} |T_c| \sum_{q \in Q} S(q, T_c^a) \tag{7}$$

其中, $S(q, T_c^a)$ 是一个 0-1 取值的函数, 表示经过节点维度 a 的划分, 查询 q 在数据子集 T_c 关于维度 a 的值域上有无交集. 进一步地, 在 LA-tree 自顶向下构造树的过程中定义评分函数 $OPRS_{T,Q,N_k}(a)$: 以 1:1 的权重混合节点 N_k 的所有祖先节点中维度为 a 的数量占比以及公式 (7) 中的 $OPR_{T,Q}(a)$. 我们的目标是选取使 $OPRS_{T,Q,N_k}(a)$ 最小化的维

度 a. 因此, 评分函数兼顾节点维度多样性, 有利于在线查询时的泛化性能.

● 证明优化公式 (7) 是优化公式 (6) 的上界. 先提出 Q-T 分布这一中间概念, 描述一个查询集合 Q 对一个数据集合 T 不同区域数据的选择度, 也称 Q 在 T 不同区域的热度. 通过比较单个节点维度 a_{N_k} 的不同选择对其子节点 Q-T 分布的影响, 定量推导出节点维度 a_{N_k} 选择对扫描量的影响.

如图 3, Q_{N_k} 在 T_{N_k} 上的 Q-T 分布形如一个多维直方图, 每个格子代表 T_{N_k} 值域上的一个小区域, 其中的数值代表 Q_{N_k} 中查询范围与此区域有交集的查询数量占比, 可以理解为热度, 越高的区域越红. 在这个单独的 Q-T 分布和节点维度选择的例子中, a_{N_k} 选择 X 比 Y 好, 结合公式 (6) 理解, 较高的子节点 Q-T 分布热力值总和具有两个有利于全局估计扫描比降低的良好性质. 首先, 任何查询的索引扫描比都不小于 1, 越高的子节点 Q-T 热力和意味着越多不在查询结果中的数据条目已经被筛选到其他子树中, 被排除候选扫描集, 因此查询负载扫描比之和的上限越低. 不失一般性地, 我们记考虑的数据表子集查询负载子集分别为 T 和 Q, 记 Q-T 分布为 $H(Q,T)$ 且都把 T 划分为 m 个区域, 记节点维度选择为 a, 优化目标:

$$\min_{a\in A}\sum_{1\leqslant c\leqslant b}\sum_{1\leqslant p\leqslant m}\left|T_{c,p}\right|(1-H(Q_c,T_c)_p) \tag{8}$$

将公式 (8) 中每条查询从 Q 中展开得到:

$$\min_{a\in A}\sum_{1\leqslant c\leqslant b}\sum_{1\leqslant p\leqslant m}\sum_{q\in Q_c}\left|T_{c,p}\right|(1-H(\{q\},T_c)_p) \tag{9}$$

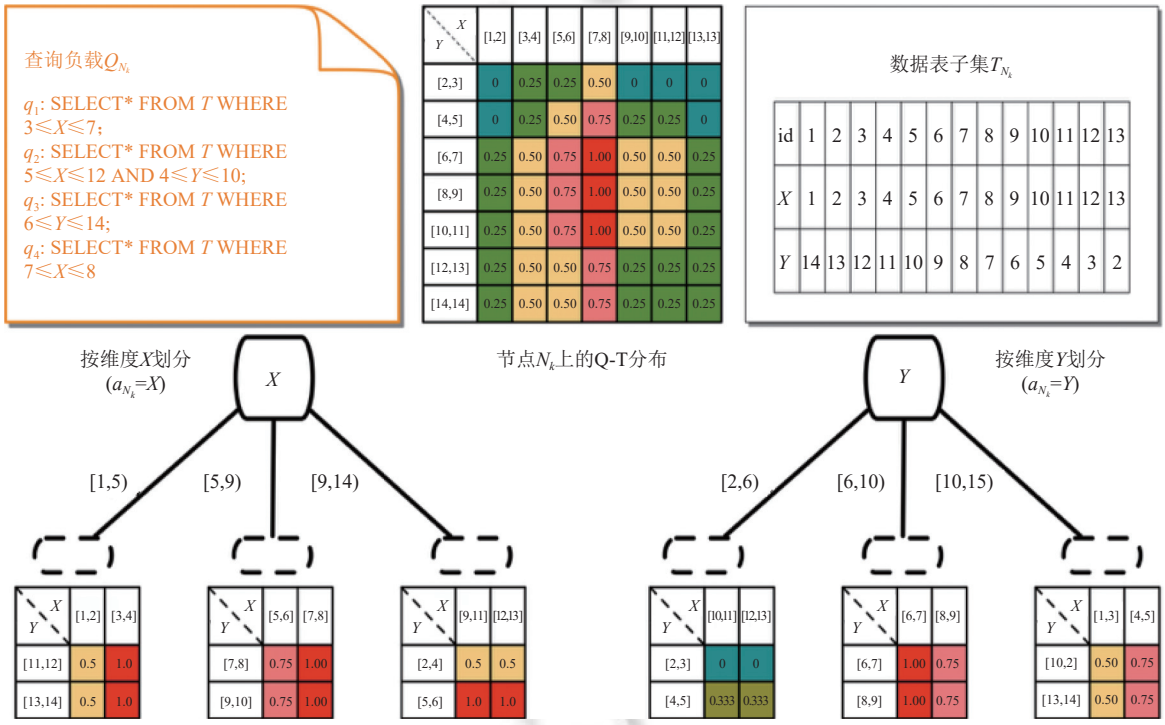


图 3 Q-T 分布及 LA-tree 节点选用不同划分维度对 Q-T 分布的影响

随 m 增大, Q-T 分布精度提升, 直至每个格子精确到仅单个数据条目. 此时我们记 $S(q,T)$ 为一个 0-1 取值的函数, 表示经过节点维度划分, 查询 q 是否需要进一步筛选数据表子集 T (q 是否与 T^a 值域相交), 公式 (9) 化简为 $\min_{a\in A}\sum_{1\leqslant c\leqslant b}|T_c|\sum_{q\in Q}S(q,T_c^a)$, 标准化即推得公式 (7) 的节点维度选择优化比, 进一步可得评分函数 $OPRS_{T,Q,N_k}(a)$, 该指标的计算仅需判断查询负载子集中每个查询 q 的不同单维约束范围 $[lb_{q,a},ub_{q,a}]$ 与 b 个数据分位点之间的大小关系, 无需执行查询, 也无需计算 Q-T 分布.

3.2 多层节点划分维度选择

LA-tree 具有多层的空间划分多叉树结构, 因此单节点划分维度选择仅能解决 LA-tree 查询感知数据划分的一个子问题. 现在分析以评分函数 $OPRS_{T,Q,N_k}(a)$ 为目标的全树节点划分维度选择问题.

● 全树节点划分维度选择是 NP 难问题. 为方便讨论, 不妨将 LA-tree 看作一棵满多叉树, 且树高为 $h+1$ (含叶子节点), 分叉数为 b. 首先, 讨论 N_k 为最底层中间节点的情况, 这是一个典型的单节点扫描比优化问题, 故需要枚举索引可能的维度组合共计 $b^0|A|^1$ 种. 现在, 假设已枚举完成 $h-1$ 层子树, 即枚举了 $b^{h-2}|A|^{h-1}$ 种维度组合, 则最上面一层的根节点 N_1 枚举其划分维度 $a_{N_1}\in A$. 带来整个问题共计枚举 $b^{h-1}|A|^h$ 种维度组合. 至此发现多层次节点划分维度选择是 NP 难问题, 可归约其简化问题 (令 $b=2$, 是否存在一棵 LA-tree, 使 Q 在其上总扫描比为 1) 至精确覆盖问题加以证明, 参考 Hyafil 等人[9]研究最小决策树的思路.

结合公式 (5), 注意到任意节点 N_k 上, 数据子集 T_{N_k} 被不同节点维度 a_{N_k} 等分为 b 份, 产生的数据表子集 $T_{N_k,c}=\left\{t_l \mid \frac{c-1}{b}<CDF(T_{N_k}^{a_{N_k}}=v_{t_l,a_{N_k}})\leqslant\frac{c}{b},t_l\in T_{N_k}\right\},c\in\{1,2,\ldots,b\}$ 有很大差异. 随节点深度增加, 数据子集大小 $|T_{N_k}|$ 关于 b 指数级下降, 因而浅层节点维度的选择对整棵 LA-tree 优化效果的影响比深处节点更大, 这意味着对于每个节点, 其维度选择导致产生不同子问题本身的优化, 比子问题的优化更加重要. 因此, 可将全局优化自顶向下地拆解为局部优化.

● 多级搜索贪心算法. 将优化分数 $OPRS_{T,Q,N_k}(a)$ 的计算推广到多层节点, 并设置一个参数级数 w, 将高度为 h 的树构造问题按每级高度为 w 分段. 段内搜索: 即每高度 w 的子树, 枚举其中每个节点维度, 以从该子树最后一层节点自下而上计算的优化比为目标. 段间贪心: 不同段的子树忽略后效性, 在上层段所有节点确定划分维度后, 独立地搜索下层段各节点划分维度.

分析该算法, 贪心算法本身就可以得到相当优的解, 因为查询负载感知选择维度优化划分数据的过程类似剪枝, 越浅层节点, 对应越大的数据子集, 对其优化越能提高索引筛选效果. 而多级搜索则进一步考虑了后效性的影响, 令全树节点维度选择更接近最优解. LA-tree 离线索引构建见算法 1.

算法 1. LA-tree 离线索引构建: *latree-offline*(T_{N_k}, Q_{N_k} ht_0, ht).

输入: 数据表子集 T_{N_k}, 查询负载训练集子集 Q_{N_k}, 本级枚举起始高度 ht_0, 当前高度 ht;
输出: LA-tree 已构建节点 N_k.

```
1. if ht = h then //叶子节点直接决定节点维度构建
2.    Choose a_{N_k} where Q_{N_k} has smallest sel on T^a_{N_k}
3.    N_k ← Leaf node on T_{N_k}
4. else //中间节点 w 步枚举节点维度, 以最小化 OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k}) 为目标
5.    if ht_0 ⩽ ht then //节点尚未确定, 处于维度选择过程中
6.       if ht = ht_0 + w − 1 then // N_k 是本级 w 层枚举的最后一层
7.          Choose a_{N_k} with smallest OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k})
8.       else //本级 w 层枚举的中间层, 每个 a_{N_k} 都选择最优子问题以最小化 OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k})
9.          for c from 1 to b do
10.            Choose a_{N_k} with smallest OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k}) by latree-offline(T_{N_k,c}, Q_{N_k,c} ht_0, ht + 1)
11.         end for
12.      end if
13.      Try partition T_{N_k} and divide Q_{N_k} by a_{N_k}
14.      Try N_k ← Middle node on T^{a_{N_k}}_{N_k}
15.      if ht = ht_0 then //本级枚举完成, 枚举的根节点确定当前构建的子树
```

16. Determine to build N_k rooted subtree
17. **for** c **from** 1 **to** b **do**
18. $N_{C_{N_k,c}} \leftarrow latree\text{-}offline(T_{N_k,c}, Q_{N_k,c}\ ht_0+w, ht+1)$ //开始下一级 w 层枚举
19. **end for**
20. **end if**
21. **else** //节点已经确定, 只需向下穿过已建节点直到 ht_0+w 层开始下一级枚举
22. **for** c **from** 1 **to** b **do**
23. $N_{C_{N_k,c}} \leftarrow latree\text{-}offline(T_{N_k,c}, Q_{N_k,c}\ ht_0, ht+1)$
24. **end for**
25. **end if**
26. **return** N_k

● 复杂度分析. 记整个查询负载训练集 Q 的平均选择度为 sel, 由于优化分数估计了扫描比的上界, 在每段搜索过程, 本段子树的叶子节点访问量可以近似为 $|T|sel$, 相应地, 本段子树平均每个叶子节点被查询访问次数可以近似为 $|Q|sel$, 同理可近似计算每层每个节点近似访问次数, 在此基础上求得算法时间复杂度为 $\mathrm{O}\left(h^w|T|\lg|T|+\left(\frac{h^wb^w}{w}\lg|T|+h^wb|T|sel\right)|Q|\right)$. 较小和较大的 w 分别接近算法涵盖的两种特殊情形: 完全贪心算法和全树枚举算法, 分别对应复杂度 $\mathrm{O}(h^w|T|\lg|T|+h^wb|Q|sel|T|)$ 和 $\mathrm{O}\left(h^w|T|\lg|T|+\frac{h^wb^w}{w}|Q|\lg|T|\right)$. 由于优化分数 $OPRS_{T,Q,N_k}(a)$ 能很好地表征扫描比变化, 且剪枝作用明显, $w\leqslant 2$ 即可算得较优的划分维度组合.

● LA-tree 超参数 h 和 b 分析. 可近似看作, LA-tree 索引将数据表 T 划分为小于 $\frac{b^h-1}{b-1}$ 个单元, 因而调节 h 和 b 本质是平衡筛选和扫描阶段的用时, 显然 h 比 b 影响更大. 因此调节超参数的算法是, 首先由用户输入索引大小, 进一步计算初始相等的 h 和 b, 再使用邻近搜索方法调节到最佳的 h, 最后在确定的 h 下使用邻近搜索方法调节到最佳的 b. 邻近搜索的每一步, 在 T 和 Q 的随机样本上构建索引并验证查询用时, 由于样本足够小, 超参数可被快速调节.

4 在线查询处理: 基于学习模型的高效在线筛选方法

本文提出的基于学习模型的高效在线筛选方法, 通过在各节点训练线性回归模型拟合数据分布, 使索引能够在每个节点以常数时间复杂度, 根据查询范围边界快速完成数据筛选, 从而同时保证筛选的准确性与高效性.

● 准确性证明. 不失一般性地, 以单维索引问题展开分析. 给定数据表 T 有 n 行且仅单维属性 X, 将其升序排序并记对应顺序条目号为 $\{i_1,i_2,\ldots,i_n\}$, 引入离散数据累积分布函数 (CDF) 刻画 T^X 的分布 $CDF(T^X)$, 表明任意 T^X 取值的相对位置, 即 $CDF(T^X=x_0)=\frac{|\{v\leqslant x_0|v\in T^X\}|}{n}$.

设一线性回归模型 (LM) $M_{T^X}(x)$ 拟合 $CDF(T^X)$, 在离线训练阶段仅根据数据分布求出其误差界限 (error-bound), 记为 $eb_{M_{T^X}}$:

$$eb_{M_{T^X}}=\max\left\{\left|M_{T^X}(v)-CDF(T^X=v)\right|\,|\,v\in T^X\right\} \tag{10}$$

基于公式 (10), 可以证明对于任意在线查询 $[lb,ub]$, 所有查询范围内的数据条目一定在 $M_{T^X}(x)$ 估计的两端位置各向外扩张一个 $eb_{M_{T^X}}$ 的范围内. 即基于 $M_{T^X}(x)$ 的学习型索引可以表示为:

$$I_T([lb,ub])=\left\{i_l,i_{l+1},\ldots,i_r\,|\,M_{T^X}(lb)-eb_{M_{T^X}}\leqslant\frac{l}{n}\leqslant\frac{r}{n}\leqslant M_{T^X}(ub)+eb_{M_{T^X}},v_{i_{l-1},X}<lb\leqslant v_{i_l,X}\leqslant v_{i_r,X}\leqslant ub<v_{i_{r+1},X}\right\} \tag{11}$$

用不等式放缩法可证明公式 (11) 成立, 即该学习型索引 I_T 的正确性. 由于 $M_{T^X}(x)$ 是单调递增函数, 对于 $\forall v\in T^X$, 若在线查询 $[lb,ub]$ 满足 $lb\leqslant v$, 必有:

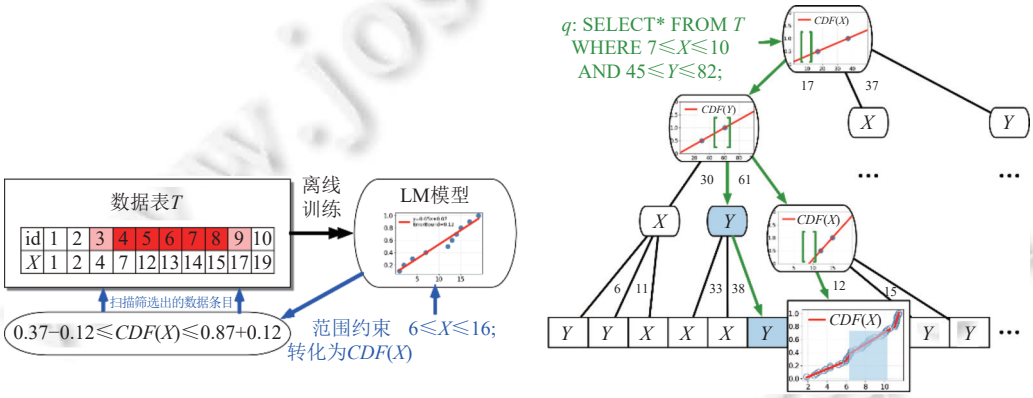
$$M_{T^X}(lb)-eb_{M_{T^X}} \leqslant M_{T^X}(v)-eb_{M_{T^X}} \leqslant CDF(T^X=v) \tag{12}$$

同理, 对于 $\forall v \in T^X$, 若在线查询 $[lb,ub]$ 满足 $ub \geqslant v$, 必有:

$$CDF(T^X=v) \leqslant M_{T^X}(v)+eb_{M_{T^X}} \leqslant M_{T^X}(ub)+eb_{M_{T^X}} \tag{13}$$

证毕. 公式 (12) 和公式 (13) 证明了公式 (11) 将学习模型估计的两端位置按训练阶段确定的误差界限 $eb_{M_{T^X}}$ 向外扩张, 索引不会遗漏任何查询范围内的数据. 即对于任何保序的学习模型, 学习型索引 I_T 都能在误差界限范围内保证准确性.

● 高效性分析. 图 4(a) 以基于 LM 的学习型单维索引为背景, 展示了线性回归模型拟合数据分布并根据约束范围检索数据的基本原理. 离线训练过程中, 仅需 O(n) 的时间快速训练 LM 即完成索引的离线构建. 在线回答过程中, O(1) 时间确定范围比传统方法 O($\log n$) 明显更加高效. 本例中, 传统方法需两次二分或平衡树查找分别确定该查询内数据条目的范围, 共计约 6–8 次数值比较操作. 而此学习型索引训练得到的 LM 满足 error-bound 仅 0.12, 在回答该查询时, 仅需约 2–3 次数值比较操作, 大幅度降低了在线查询用时. 随数据量增大, 二分所需数值比较次数呈对数级增长, 而 LM 的误差通常仍维持在常数范围内, 即数值比较次数几乎不变. 进一步地, 对于数据量较大且分布倾斜的情形, LM 误差将增大. 对此, 可使用分段式线性回归模型 (PLM) 拟合 CDF, 模型输入输出与 LM 一致, 将误差控制在常量范围内. PLM 的正确性支撑类似函数微分原理, 在局部线性化近似函数曲线. 在 PLM 中, 排序后的数据将被分为若干连续的小区间, 由一个总体 LM 定位区间, 而每个小区间内再由一个 LM 拟合局部 CDF, 且保证任意小区间内 LM 的 error-bound 不超过设定的限制, 时间复杂度为 O(1).



(a) 线性回归模型拟合累积分布函数　　(b) LA-tree基于学习模型的高效在线筛选方法

图 4　线性回归模型拟合数据积累分布函数

● 基于学习模型的高效在线筛选方法. 如图 4(b), 总体上, 该方法首先在 LA-tree 树形索引结构中自顶向下地筛选出数据范围与查询范围有交集的所有节点, 直到叶子节点确定需要进一步筛选和扫描的所有单元, 通过扫描得出查询结果子集. 最后, 该方法自底向上地合并各子树上的查询结果子集, 得到完整的查询结果.

对于每个具体的节点 N_k, 若 N_k 是一中间节点, 由于分叉数 b 通常较小, 因而中间节点对 error-bound 不敏感. 自顶向下筛选时, 通过拟合 $CDF(T_{N_k}^{a_{N_k}})$ 的 LM 模型, 对查询 q 的谓词在节点维度 a_{N_k} 上的范围 $[lb_{q,a_{N_k}},ub_{q,a_{N_k}}]$, 用模型估算其两端点在数据子集 T_{N_k} 关于 a_{N_k} 的投影 $T_{N_k}^{a_{N_k}}$ 中的有序位置, 再将估算结果向两端分别扩张 $eb_{M_{T_{N_k}}^{a_{N_k}}}$ 的宽度以保证数据筛选没有遗漏. 这样, 可以得到 $[lb_{q,a_{N_k}},ub_{q,a_{N_k}}]$ 包含的节点内的分位点, 对应了与查询范围相交的、数据子集 T_{N_k} 按维度 a_{N_k} 进一步划分的所有子集. 计算完成后, 即可获得明确的需要访问以进一步细化筛选的所有子节点. 当筛选与扫描结束, 自底向上合并查询结果时, 再在节点 N_k 上求这些被访问的子节点返回的查询结果子集的并集, 作为节点 N_k 返回的查询结果子集. 结合公式 (2) 和公式 (11), 经上述流程, 节点 N_k 的查询结果子集为:

$$I_{T_{N_k}}(q)=\bigcup_{M_{T_{N_k}}^{a_{N_k}}(lb_{q_0,a_{N_k}})-eb_{M_{T_{N_k}}^{a_{N_k}}} \leqslant \frac{c}{b} \leqslant M_{T_{N_k}}^{a_{N_k}}(ub_{q_0,a_{N_k}})+eb_{M_{T_{N_k}}^{a_{N_k}}}} I_{T_{N_{C_{N_k,c}}}}(q) \tag{14}$$

若 N_k 是一个叶子节点, 由于在一定的 h 和 b 设置下, 一个叶子节点可能包含较多数据条目, 使用 LM 模型筛选数据可能带来较高的 error-bound, 因而叶子节点使用 PLM 模型拟合按节点维度 a_{N_k} 排序后的数据子集的分布 $CDF(T_{N_k}^{a_{N_k}})$. 自顶向下筛选时, 叶子节点是最后一层, 通过该 PLM 模型, 对查询 q 的谓词在节点维度 a_{N_k} 上的范围 $[lb_{q,a_{N_k}}, ub_{q,a_{N_k}}]$, 用模型估算其两端点在数据子集 T_{N_k} 关于维度 a_{N_k} 的投影 $T_{N_k}^{a_{N_k}}$ 中的有序位置, 再将估算结果向两端分别扩张 $eb_{M_{T_{N_k}}^{a_{N_k}}}$ 的宽度保证数据筛选没有遗漏. 注意, 叶子节点 PLM 模型估算位置的过程与中间节点 LM 模型稍有不同, 具体来说先定位输入值所在分段 (piece), 再用段上的局部 LM 模型估算位置. 计算完成后, 即可得到 $[lb_{q,a_{N_k}}, ub_{q,a_{N_k}}]$ 包含的所有数据条目, 进一步扫描以确定每条数据是否在查询范围内. 当筛选与扫描完成, 自底向上合并查询结果时, 叶子节点 N_k 直接返回经过“筛选+扫描”确定的查询结果子集. 结合公式 (1) 和公式 (11) 有:

$$I_{T_{N_k}}(q)=\left\{i \mid lb_{q,a_j} \leqslant v_{t'_i,a_j} \leqslant ub_{q,a_j}, \forall a_j \in A, t'_i\left\{M_{T_{N_k}^{a_{N_k}}}(lb_{q,a_{N_k}})-eb_{M_{T_{N_k}^{a_{N_k}}}} \leqslant \frac{l}{n} \leqslant \frac{r}{n} \leqslant M_{T_{N_k}^{a_{N_k}}}(ub_{q,a_{N_k}})+eb_{M_{T_{N_k}^{a_{N_k}}}}\right\}\right\} \tag{15}$$

最终, 查询结果由根节点返回即 $I_T(q)=I_{T_{N_1}}(q)$. 上述高效在线筛选方法, 在 LA-tree 每个节点上, 都能利用机器学习模型以 O(1) 的开销快速准确筛选子节点和数据, 令查询扫描比进一步降低的同时无需与中间节点的 $b-1$ 个分位点以及叶子节点的排序数据做数值比较, 效率很高. 算法 2 描述了上述包含公式 (14) 和公式 (15) 的基于学习模型的高效在线筛选方法.

算法 2. LA-tree 在线查询处理: *latree-online*(N_k, q).

输入: LA-tree 节点 N_k, 在线查询 q;
输出: 节点索引功能返回的数据条目号集合 $I_{T_{N_k}}(q)$.

1. **if** $C_{N_k}=\varnothing$ **then** //叶子节点筛选与扫描
2. $I_{T_{N_k}}(q) \leftarrow$ Scan data tuples T_{N_k} after filtering by q_0 with PLM on dimension a_{N_k}
3. **else** //中间节点筛选
4. $I_{T_{N_k}}(q) \leftarrow \varnothing$
5. $fchild, tchild \leftarrow$ Filter child nodes by $\left[lb_{q,a_{N_k}}, ub_{q,a_{N_k}}\right]$ with LM on dimension a_{N_k}, refine by $\pm eb$
6. **for** c **from** $fchild$ **to** $tchild$ **do**
7. $I_{T_{N_k}}(q) \leftarrow I_{T_{N_k}}(q) \cup latree\text{-}online(N_{C_{N_k,c}}, q)$
8. **end for**
9. **end if**
10. **return** $I_{T_{N_k}}(q)$

上述算法 2 入口为 *latree-online*(N_1, q), 函数执行完毕后将返回查询 q 在数据 T 上的结果.

5 在线索引更新: 自适应增量式的索引更新方法

LA-tree 的在线索引更新由两部分组成: 增量更新与自适应更新. 增量更新保证新数据能够被实时插入并维护局部有序性, 自适应更新则通过动态调整局部数据划分以适应数据分布和查询负载的变化, 两者共同确保索引在动态环境下依然保持“均匀划分”与“查询感知”的特性. 增量更新: 该方法应对动态场景中的数据更新操作, 分为两步骤: 先定位增删改的数据条目在索引中储存的位置, 再对该位置删除或插入新元素. 这里第 2 步较为简单, 使用标记法即可在 O(1) 的时间内解决. 对于第 1 步, 我们将数据条目转化为点查询, 即对新数据条目 $t'=\langle v_{t',a_1}, v_{t',a_2}, \ldots, v_{t',a_d}\rangle$ 转化为对应查询 $q:[v_{t',a_1}, v_{t',a_1}] \wedge [v_{t',a_2}, v_{t',a_2}] \wedge \ldots \wedge [v_{t',a_d}, v_{t',a_d}]$, 该查询基数恒为 1, 显然通过自顶向下的节点模型高效映射, 仅需 O(h) 时间即可完成新数据条目的定位, 该过程如算法 3 所示.

算法 3. LA-tree 增量数据索引更新: *latree-incremental-data-update*(N_k, t).

输入: LA-tree 节点 N_k, 更新数据条目 t;

1. **if** $C_{N_k} = \varnothing$ **then**
2. Insert/delete/update data tuple t after locating t with PLM on dimension a_{N_k}
3. **else**
4. $c \leftarrow$ Choose the child node by $t^{a_{N_k}}$ with LM on dimension a_{N_k}, refine by $\pm eb$
5. *latree-incremental-data-update*($N_{C_{N_k,c}}$, t)
6. **end if**
7. **return**

● 数据更新对索引影响. 在 LA-tree 对数据的划分不发生改变之前, 上述增量更新过程不会影响任何节点上机器学习模型的 error-bound. 由公式 (12) 和公式 (13) 可以保证, 新数据总会被更新到各节点正确的分位点之间, 即最终更新到正确的单元. 这种情况下, 任何范围包含新数据的查询, 不会发生结果遗漏. 然而, 随着数据更新的累积, 数据分布的变化将使得部分区域数据量明显超过平均水平, 即 LA-tree “均匀划分”的特性不再保持. 这会导致范围与该区域有交集的查询扫描量升高, 查询用时变长.

● 查询负载变化对索引影响. 随查询负载的变化, 查询范围分布将发生变化, 部分节点的维度和分位点可能被查询范围完全包含, 从而失去筛选数据的效果. 这同样会导致查询扫描量升高, 查询用时变长.

综上, 更新的关键影响在于检查查询扫描量的变化. 随着数据更新与查询负载变化的累积, 原有 LA-tree 索引的划分模式可能不再适应于当前的数据分布和查询负载, 导致在线查询扫描比上升, 查询耗时增加, 因而需要对 LA-tree 的划分模式重新调整.

● 自适应更新. 利用动态扫描量自适应判断 LA-tree 索引的哪些节点需要重构以调整数据划分, 该方法既不按照数据分布的变化判断, 也不按照查询负载的变化判断, 而是当局部索引的性能真正受到两者共同变化的影响, 体现在节点的评分函数受新查询较高的扫描比影响, 增大幅度超过阈值, 即性能明显劣化, 判断以该节点为根的子树需要重构以重新拟合新的数据分布与查询负载. 如图 5, 若干新查询执行过程中, 参与筛选的节点在自底向上结果合并时, 以子节点返回的查询扫描比快速更新节点评分函数. 在这一过程中, 如检测到评分函数大幅增大, 如图 5 中红框内的节点, 则标记该相应子树需重构, 并在本次查询结束后执行.

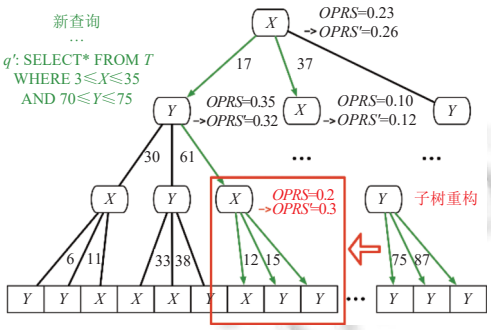


图 5 LA-tree 自适应更新子树重构

算法 4 展示了自适应重构的框架, 检查出满足上述条件的节点, 并对相应子树调用离线构建算法 2 重构子树. 如此, 所有需要重构的子树全部重新优化了性能, 此外没有任何多余的调整操作.

算法 4. LA-tree 自适应索引子树重构: *latree-adaptive-reconstruction*(N_1, q).

输入: LA-tree 根节点 N_1, 在线查询 q;
输出: 更新后的 LA-tree 索引根节点 N'_1.

1. $N' \leftarrow$ *latree-adaptive-check*(N_1, q) //需更新节点集合

2. $Q \leftarrow Q \cup \{q\}$ //更新查询负载训练集
3. **for** N'_c **in** N' **do**
4. $ht'_0 \leftarrow$ Height of N'_c
5. $N'_c \leftarrow latree\text{-}offline(T_{N'_c}, Q_{N'_c}\ ht'_0, ht'_0)$
6. **end for**
7. **return** N'_1

算法 5 是自适应子树重构触发. 这是算法 4 高效的关键, 触发方法随在线查询算法 2 同时执行, 实时检查索引是否需要更新, 并确定需要更新的局部, 在当前查询完毕后执行索引更新, 即自适应索引子树重构算法 4. 我们通过在节点增加一个数值变量维护实时评分函数变化, 在线查询过程中以 O(1) 的额外时间开销不断更新, 高效且准确. 在自底向上合并查询结果子集的过程中, 不断地将查询扫描比与所访问节点评分函数做加权平均, 更新节点的评分函数. 当一个节点当前评分函数增高超过阈值 τ, 说明该节点为根的子树不再适应当前的新数据和查询, 将节点加入重构节点集合. 同时, 考虑到不同层次数据子集之间的包含性, 当上层节点加入重构节点集合后, 将自动从集合剔除其所有后代节点, 避免重复构造. 由于越上层的节点覆盖的数据范围越大, 查询数量越多, 从而受新数据和新查询影响越小. 因此, 算法实际通常只触发少量下层节点重构, 更新耗时少.

算法 5. LA-tree 自适应子树重构触发: $latree\text{-}adaptive\text{-}check(N_k, q, \tau)$.

输入: LA-tree 节点 N_k, 在线查询 q_0, 阈值 τ;
输出: 需重构节点集合 N', q_0 的真实局部扫描比 sr_{q_0}.

1. **if** $C_{N_k} = \varnothing$ **then** //叶子节点直接计算扫描比
2. $\varnothing, sr_q \leftarrow$ Leaf node not to reconstruct. Scan ratio of q on T_{N_k} by PLM on dimension a_{N_k}
3. **else** //中间节点合并子树扫描比, 更新当前 $OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k})$
4. $fchild, tchild \leftarrow$ Filter child nodes by $\left[lb_{q,a_{N_k}}, ub_{q,a_{N_k}}\right]$ with LM on dimension a_{N_k}, refine by $\pm eb$
5. **for** c **from** $fchild$ **to** $tchild$
6. $N', sr_q \leftarrow$ Union and average of $latree\text{-}adaptive\text{-}check(N_{C_{N_k,c}}, q_0)$
7. **end for**
8. Update $OPRS'_{T_{N_k},Q_{N_k},N_k}(a_{N_k})$ by merging sr_q //中间节点用局部扫描比加权平均更新评分函数值
9. **if** $OPRS'_{T_{N_k},Q_{N_k},N_k}(a_{N_k}) > (1+\tau)OPRS_{T_{N_k},Q_{N_k},N_k}(a_{N_k})$ rises sharply **then** //本节点及整棵子树需要重构
10. $N' \leftarrow \{N_k\}$
11. **end if**
12. **end if**
13. **return** N', sr_q

综上, 算法 3 与算法 4、算法 5 共同组成了高效的 LA-tree 索引自适应增量更新方法.

6 实 验

6.1 实验数据集和查询负载

本文在公开且常用的评测数据集 Stock[10]与 TPC-H Lineitem[11]上进行了实验. 为进一步评测多维索引在高维数据下的性能, 又在较新的基准数据集 DSB Sales[12]上开展实验. 表 1 总结了各数据集及其对应查询负载的基本情况.

(1) 数据集 Stock 记录了多支股票在多个日期的价格变化. 其特点是数据记录规模大, 部分维度值域较广, 且数据维度间相关性较强, 适于考察多维索引在大规模数据上的性能表现.

(2) 数据集 TPC-H Lineitem 是最常见的 OLAP 基准测试数据, 包含商品交易的事实记录, 其特点是数据记录规模较大、维度值域广, 但维度间相关性较弱.

(3) 数据集 DSB Sales 由 Ding 等人[12]设计, 是较新的 OLAP 基准测试集合, 包含大量交易记录. 其特点是数据记录规模较大、维度数量高, 因此本文选用该数据集用于评估多维索引在高维数据下的性能表现.

表 1 实验所用的数据集与查询负载统计情况

数据集	数据表行数 (M)	数据表维度	查询负载	查询维度	查询谓词范围
Stock	20	7	Uniform	1–4	均匀分布, 谓词宽度正态分布
			Skew	1–4	向极值倾斜, 谓词宽度指数分布
TPC-H Lineitem	3	8	Uniform	2–5	均匀分布, 谓词宽度正态分布
			Skew	2–5	向极值倾斜, 谓词宽度指数分布
DSB Sales	6	34	Uniform	2–5	均匀分布, 谓词宽度正态分布
			Skew	2–5	向极值倾斜, 谓词宽度指数分布
			HD	7–33	同Uniform

由于多维索引主要关注查询谓词对数据表多维属性范围的同时约束, 部分数据表没有附带查询或附带查询不符合实验场景, 我们用随机模板的方式构造了查询负载. 具体来说, 先随机生成多个模板, 每个模板覆盖数据表的部分属性维度. 之后在生成查询的过程中, 第 1 步随机选择一个模板, 第 2 步服从查询负载设置的查询范围分布随机生成模板包含的每个属性维度上的约束范围, 合起来作为查询谓词. 该方法既符合模板法查询生成的惯例, 又保证了查询的随机性和多样性. 在此基础上, 我们构造了均匀 (Uniform) 和倾斜 (Skew) 两种谓词范围分布不同的查询负载, 其中, Uniform 查询负载谓词的中点位置、范围宽度关于数据值域均匀分布; Skew 查询负载谓词的中点位置、范围宽度关于数据值域指数分布. 此外, 针对高维数据集 DSB Sales 构造了高维 (high-dimensional, HD) 查询负载, 使每条查询谓词约束的属性维度提高 3–15 倍, 更加具有挑战性.

6.2 评价指标、对比方法和实验设计

(1) 实验 1: 静态场景. 多维索引主要应用场景侧重于优化在线查询性能, 而数据更新不频繁, 且现有工作对更新功能考虑较少. 因此, 我们按照现有工作惯例, 先评测最重要的静态场景中的索引查询性能, 其评测指标如下.

● 在线查询平均用时. 这是首要指标, 即对于测试集每条查询, 计算从调用索引在线查询入口函数到该函数返回包含所有查询范围内数据的指针的数组的用时, 再求平均用时. 进一步地, 我们分析多维索引查询过程中筛选和扫描两个阶段的用时, 但在索引内部函数中反复计时会显著干扰索引性能使测试结果失真, 因而我们通过两个常用且关键的指标间接对比索引在两个阶段分别的用时.

● 在线查询平均扫描比. 计算索引扫描的数据条目数和查询范围内数据条目数的比值, 由于小基数的查询易使该指标突增 (尽管扫描的数据条目数仍然很少), 故计算执行整个查询负载测试集的扫描比而非每个查询平均扫描比. 一般来说, 若该指标较小, 索引扫描查询范围外的条目数较少自然扫描阶段开销较小, 筛选效果较好.

● 在线查询平均访问单元数量. 计算索引平均一条查询筛选出需访问的单元数量. 一般来说, 若该指标较小, 说明索引在筛选阶段用时较短, 也即筛选效率较高.

● 索引离线构建用时和索引大小. 作为次要指标, 虽然索引性能以在线查询性能为主要指标, 也应该兼顾合理的离线构建用时和索引大小.

● 基准方法. 分为传统和学习型两类. 传统方法包括 Clustered、KD-tree 和 Octree. 其中, Clustered 是单维索引, 仅对查询负载谓词中选择度最低的属性维度建立排序索引, 由一次二分查找直接获得查询结果, 用于模拟数据库系统中最常用的聚簇索引. KD-tree[1]是经典的多维空间划分二叉树, 常用于高效的多维索引使用. Octree[2]是经典的三维空间划分八叉树, 一般用作空间索引, 我们参考现有工作[4,5]对其的拓展, 令每个节点随机任选数据表的 3 个维度做三维空间划分, 使其作为多维索引使用. 学习型方法包括 Qd-tree 和 Tsunami. 其中, Qd-tree[3]总是在查询边缘对数据做划分, 对查询负载训练集强拟合, 考虑到查询较多, 所有查询边缘全部参与划分使得分片过于细碎, 大幅增加索引占用空间大小同时反而拖累在线查询性能, 我们在每个数据集的每个查询负载上都为 Qd-tree 分别调优了划分

总数限制. Tsunami[5]在学习型网格的基础上对查询负载分布和数据中的二维相关情形做了优化, 由于作者未公开其用于调优的模型, 无法使用该方法调优索引, 我们按照其论文中说明的另一种迭代方法调优参数, 考虑论文提及调优模型的训练需在大量不同的数据集反复构建索引, 两种调优方法增加的索引离线构建用时是可比的.

(2) 实验 2: 泛化性能与动态场景. 学习型索引大多使用了查询负载优化索引性能, 因此应评价无查询负载训练集和查询负载偏移场景的泛化能力. 对于查询负载偏移场景, 索引在每个数据表的 Uniform 查询负载上离线构建索引, 在线查询 Skew 负载. 此外, 相比其他多维索引方法, LA-tree 以高效的索引自适应增量更新方法拓展了其更新支持能力, 故我们额外增加 LA-tree 索引更新性能纵向评估, 以朴素的定期重新构建 LA-tree 索引作为基准更新方法与 LA-tree 索引自适应增量更新方法作纵向比较.

● 读写混合负载. 为增强更新对原索引的影响, 提高实验挑战性, 该负载包含普通查询和新数据插入指令. 其中包含的普通查询为 TPC-H Lineitem 数据集的 Skew 查询负载, 而考虑到 TPC-H Lineitem 数据集的相关性弱, 我们新增的 3M 行数据用属性各维度单独排序的方式增强数据相关性. 实验 2 在 TPC-H Lineitem 数据集及对应的 Uniform 查询负载训练集上离线构建 LA-tree 索引, 然后在线运行我们设计的读写混合负载. 基准更新方法将整个负载分为 10 个区间, 在区间间隔处重新构建 LA-tree 索引而在每个区间中将新增数据保存至缓冲区, 与之前构建好的索引一起处理查询结果. LA-tree 索引自适应增量更新则将每条新增数据通过中间节点 LM 和叶子节点 PLM 自顶向下地插入到 LA-tree 的正确位置, 再对 LA-tree 做自适应重构.

● 评测环境. 所有的基准方法及 LA-tree 都用 C++ 11 实现. 为公平比较, 各方法的入口函数、计时框架、回归模型 (如 LM) 等部分代码都尽量一致. 评测环境: 系统 Ubuntu 20.04.6 LTS; 处理器 Dual CPU System: 2×Intel(R) Xeon(R) Gold 6230 CPU@2.10 GHz (20C40T); 运行内存 1 TB DDR4 ECC; 外存磁盘 4×8 TB HDD (RAID5).

6.3 实验 1: 静态场景索引查询性能评测

我们首先按照在线查询平均用时对索引查询性能做总体比较, 然后通过平均访问单元数量和扫描比两个指标分析造成索引性能瓶颈的问题所在, 最后比较索引的离线构建用时和索引大小. 其中, LA-tree 使用第 3 节提出的临近搜索法, 在数据与查询负载 1% 采样率的小样本上确定了超参数 h 和 b. 具体地, 在 Stock 数据集上 $h = 8$, $b = 3$; 在 TPC-H Lineitem 数据集上 $h = 7$, $b = 10$; 在 DSB Sales 数据集上 $h = 9$, $b = 5$.

如表 2 所示, 最优结果用加粗表示. 传统索引整体性能有限, 其中, Clustered 作为单维索引, 整体性能最差. 但其在 Stock 数据集上表现相对较好, 这是因为 Stock 中“股价”维度的取值重复较少, 使谓词筛选效果较佳, 从而提升了单维索引性能. KD-tree 与 Octree 作为典型的多维索引, 整体明显优于单维索引. 但二者均缺乏查询感知, 并在筛选时依赖大量数值比较, 其查询用时仍比学习型索引多 1–2 倍. 相对而言, 多维划分的 KD-tree 性能优于三维划分的 Octree, 原因在于前者更简洁的节点功能带来的较低算法开销.

表 2 索引在线查询平均每条查询用时比较

数据集	查询负载	Clustered (ms)	KD-tree (ms)	Octree (ms)	Qd-tree (ms)	Tsunami (ms)	LA-tree (ms)	比SOTA提升 (%)	比学习型提升 (%)
Stock	Uniform	41	456	468	43	56	**31**	24	28
	Skew	30	419	338	32	79	**26**	13	19
TPC-H Lineitem	Uniform	178	23	30	19	31	**12**	37	37
	Skew	182	24	29	17	19	**12**	29	29
DSB Sales	Uniform	483	313	442	148	1 109	**71**	52	52
	Skew	429	314	445	134	1 559	**74**	45	45
	HD	901	128	251	256	1 017	**84**	34	67
平均情况		320.6	240.0	286.1	92.7	552.9	**44.3**	52	52

相比之下, 学习型索引在大多数场景下更具优势. Qd-tree 在部分查询负载下平均查询用时较短, 但在 DSB Sales 数据集的高维场景中性能显著下降, 原因在于其非均匀数据划分在高维下劣势凸显. Tsunami 的查询用时整体偏高, 并在 DSB Sales 上性能退化更为严重. 这些结果表明, 基于网格结构的学习型索引在拟合高维数据分布时存在局限.

综合来看, LA-tree 在所有测试中均取得最佳结果, 总体性能相比其余现有最优方法提升约 52%. 其优势来自

优化的树形数据划分与学习模型结合的高效筛选机制, 使其在高维查询场景中表现尤为突出.

实验结论 1. LA-tree 在多维索引方法中表现出最佳的查询性能, 并且在高维场景下优势更加显著. 其原因在于: 一方面, 优化的树形结构实现了更有效的数据划分; 另一方面, 学习模型显著加速了数据筛选.

结合图 6 和图 7 可以看出, LA-tree 在绝大多数测试中的扫描比和平均访问单元数量均为最小, 说明其在线查询在筛选和扫描阶段的耗时均接近最低, 与其在平均查询时间上的实验结果相一致. 值得注意的是, 扫描比与平均访问单元数量存在权衡关系, 而 LA-tree 能在保持平均访问单元数量普遍比其他多维索引低 1–2 个数量级的前提下, 依然实现最佳的扫描比, 尤其在高维数据集 DSB Sales 上优势更加明显. 结合表 3 中展示的各多维索引查询用时中筛选用时的比重, 可以看出 LA-tree 筛选方法效率相比现有方法大幅度提高, 同时结合其极低扫描比, 说明 LA-tree 查询感知的均匀划分同时取得很好的筛选效果.

表 3　多维索引在线查询筛选用时平均占比 (%)

数据集	查询负载	KD-tree	Octree	Qd-tree	Tsunami	LA-tree
Stock	Uniform	44.1	39.7	20.9	39.3	**3.2**
	Skew	47.5	44.1	9.4	44.3	**3.8**
TPC-H Lineitem	Uniform	43.5	40.0	47.4	38.7	**16.7**
	Skew	45.8	41.4	47.1	**5.3**	16.7
DSB Sales	Uniform	48.9	42.5	49.3	55.2	**7.0**
	Skew	48.7	42.7	50.0	57.6	**6.8**
	HD	42.2	38.2	48.0	55.1	**2.4**
平均情况		45.8	41.2	38.9	42.2	**8.1**

在学习型方法中, Qd-tree 表现相对较好, 但在 DSB Sales 数据集的高维查询负载下, 其扫描比较其他查询负载增加近 3.5 倍, 平均访问单元数量也显著上升. 这是因为高维查询导致查询边界数量激增, 非均匀划分放大了扫描比的劣势. Tsunami 整体优于传统方法, 但在 DSB Sales 上平均访问单元数量和扫描比均大幅增加, 与其在该数据集上较高的查询延迟相一致. 这反映出网格结构在高维数据分布下难以兼顾访问单元数量与扫描比调优的瓶颈, 本质原因在于网格难以拟合高维相关性, 尤其在稀疏分布中问题更加突出.

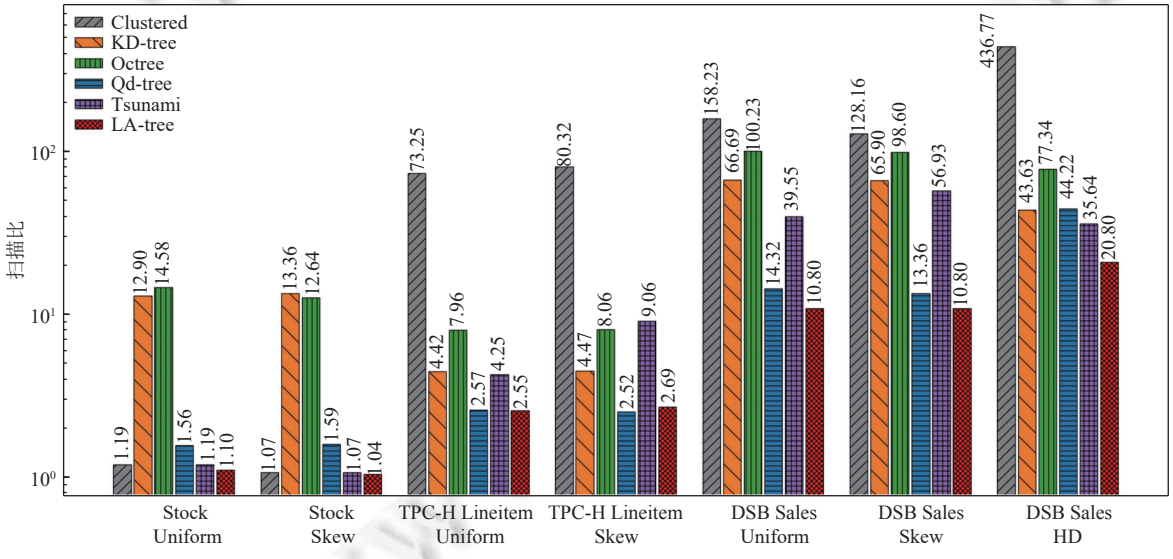


图 6　各索引在线查询平均扫描比对比

在传统方法中, Clustered 作为单维索引不涉及单元划分, 其在 Stock 数据集上的扫描比较低, 与其较短的查询时间相符. 但在其他数据集上, 扫描比显著高于多维索引, 进一步验证了多维索引的有效性. KD-tree 和 Octree 在

扫描比和访问单元数量上均明显优于 Clustered, 但仍不及学习型方法, 说明学习型多维索引在筛选性能上的优势. 二者中 KD-tree 因节点功能更简洁、算法开销更低, 相比 Octree 具有更好的查询性能. 需要注意的是, 快速在线查询依赖于筛选和扫描两个阶段同时具备较低开销, 单一指标的优势不足以显著提升整体性能. 例如, Tsunami 在 Stock 数据集的 Skew 查询负载下实现了与 LA-tree 相当的极低扫描比, 又在 TPC-H Lineitem 数据集的 Skew 查询负载下实现了最小的平均访问单元数量, 但由于另一指标表现明显不足, 其在线查询平均用时依然较长. 实验中 Tsunami 按照其论文提出的代价函数搜索超参数, 该函数目标最小化查询负载在筛选和扫描阶段的总耗时. 在 Skew 负载上, 受查询范围位置分布较为倾斜, 宽度分布方差较大, 可能出现牺牲扫描比但大幅减少划分格子数量提高整体在线查询性能的情况. 这里 Tsunami 在线查询性能不够好的原因主要在于网格难以拟合多维相关性.

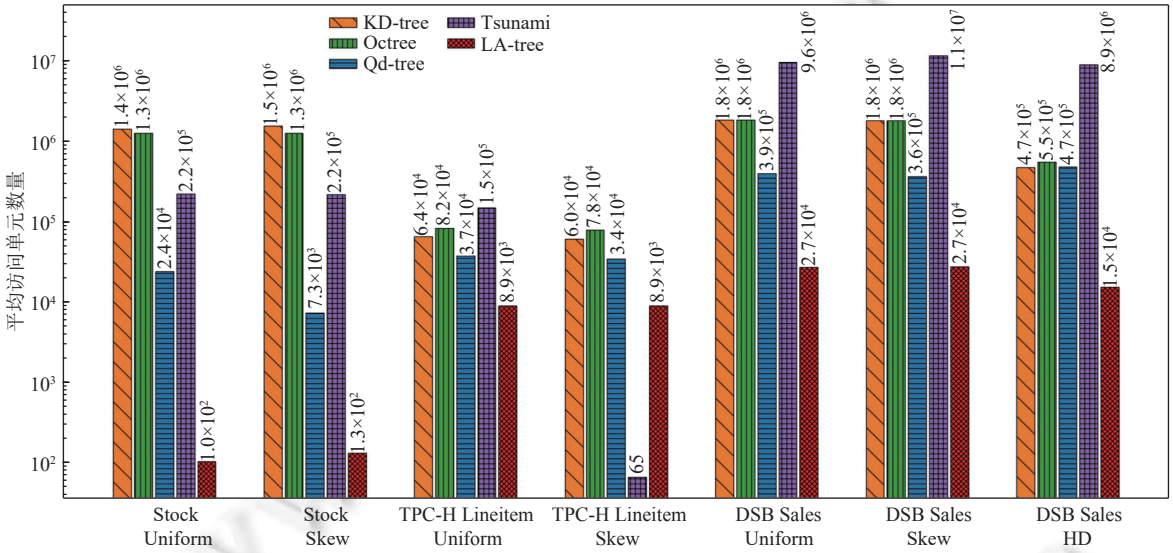


图 7 各多维索引在线查询平均访问单元数量对比

实验结论 2. LA-tree 的多层次查询感知的数据划分方法显著优于现有索引方案. 通过在树形结构中同时兼顾“均匀划分”与“查询感知”, LA-tree 在多维范围查询中展现出更强的筛选能力, 能够有效降低扫描比. 特别是在高维场景下, 其优势更加凸显.

表 4 和表 5 展示了索引在离线构建的用时与空间开销, 最优结果用加粗表示. 学习型索引整体上离线构建耗时较高, 传统多维索引与学习型网格索引 Tsunami 在数据量较大的 Stock 数据集上空间开销急剧膨胀, 显示出较差的可扩展性. 除单维索引 Clustered 外, 其余多维索引均存在较大的空间占用, 但整体仍处于数据库系统可接受范围.

表 4 索引离线构建用时比较 (s)

数据集	查询负载	Clustered	KD-tree	Octree	Qd-tree	Tsunami	LA-tree
Stock	Uniform	**2**	33	35	199	236	191
	Skew	**2**	33	35	168	234	184
TPC-H Lineitem	Uniform	**0.4**	4	4	87	206	47
	Skew	**0.4**	4	4	81	205	48
DSB Sales	Uniform	**0.5**	7	9	430	546	406
	Skew	**0.5**	7	9	412	545	410
	HD	**0.5**	7	9	296	546	408

相比之下, LA-tree 在离线构建开销上与其余学习型方法处于同一水平, 且空间占用保持稳定并相对轻量. 这表明 LA-tree 在保证高效查询性能的同时, 能够在离线构建和空间开销两个方面兼顾可扩展性与实用性.

实验结论 3. LA-tree 多层次查询感知的数据划分方法支持高效的离线索引构建. 其在节点层面采用的轻量化 LM 与 PLM 模型, 不仅保证了索引筛选性能, 同时有效降低了离线构建的时间开销与空间占用.

表 5　索引占用空间大小比较 (MB)

数据集	查询负载	Clustered	KD-tree	Octree	Qd-tree	Tsunami	LA-tree
Stock	Uniform	**240**	2 640	3 540	292	938	509
	Skew	240	2 640	3 540	**164**	1 152	524
TPC-H Lineitem	Uniform	**34**	358	463	332	234	207
	Skew	**34**	358	463	332	80	195
DSB Sales	Uniform	**65**	631	803	631	471	322
	Skew	**65**	631	803	631	380	321
	HD	**65**	631	803	631	471	206

6.4　实验 2: 索引泛化性能及动态场景索引更新评测

本节从多个角度评估 LA-tree 在动态场景下的索引性能与泛化能力, 涵盖 3 类典型情形: 无查询负载、查询负载偏移以及索引更新.

如表 6 所示, 最优结果用加粗表示. 首先在不使用查询负载的条件下评估 LA-tree 的性能, 即禁用其多层次查询感知的数据划分方法. 需要注意的是, 由于其他学习型索引无法关闭查询感知特性, 因此此处仅将关闭查询感知优化的 LA-tree 与传统索引方法进行对比. 结果表明, LA-tree 的基于学习模型的筛选方法依然非常高效, 相比最快的传统多维索引平均查询用时降低约 62%, 说明在缺乏查询负载时, LA-tree 仍能作为高性能的学习型多维索引使用. 进一步地, 结合实验 1 中表 2 的结果, 在有查询负载条件下当启用多层次查询感知的数据划分方法后, 完整的 LA-tree 在相同实验条件下还可减少约 52% 的查询用时. 这一结果表明, LA-tree 在索引构建阶段的“查询感知数据划分”与在查询阶段的“学习型筛选优化”两方面相辅相成, 共同作用于减少在线查询时间.

表 6　LA-tree 无查询负载训练集情形在线查询平均每条查询用时

数据集	查询负载	Clustered (ms)	KD-tree (ms)	Octree (ms)	LA-tree (ms)	提升 (%)
Stock	Uniform	41	456	468	**33**	20
	Skew	**30**	419	338	**30**	0
TPC-H Lineitem	Uniform	178	23	30	**18**	22
	Skew	182	24	29	**18**	25
DSB Sales	Uniform	483	313	442	**228**	27
	Skew	429	314	445	**221**	30
	HD	901	128	251	**94**	27
平均情况		320.6	240.0	286.1	**91.7**	62

实验结论 4. LA-tree 的多层次查询感知的数据划分方法与学习型筛选方法均能显著降低在线查询用时. 即使在缺乏查询负载的情况下, LA-tree 依然保持较高的查询效率, 体现出其良好的泛化能力.

查询负载变化令查询范围分布发生变化, 使得利用查询负载训练集优化的学习型索引性能下降. 表 7 比较学习型索引在查询负载变化情形的泛化性能, 最优结果用加粗表示. 不难看出, 对于在 Uniform 查询负载上离线构建索引而在线查询负载为 Skew 的查询负载变化场景, LA-tree 总是表现出最快的在线查询平均用时, 且总体性能比学习型基准方法提升了 54%, 高于之前静态场景提升幅度.

表 7　查询负载变化情形学习型索引泛化性能 (在线查询平均每条用时) 比较

数据集	Qd-tree (ms)	Tsunami (ms)	LA-tree (ms)	提升 (%)
Stock	103	48	**28**	42
TPC-H Lineitem	18	31	**13**	28
DSB Sales	138	1 017	**77**	44
平均情况	86.3	365.3	**39.3**	54

实验结论 5. LA-tree 的树形结构均匀数据划分, 提高索引面对不同查询负载时的鲁棒性和泛化性能. LA-tree 的索引更新主要面临两方面挑战: 一是如何将更新开销控制在足够低的水平, 以避免显著增加系统负担; 二是如何适应不断变化的数据分布与查询负载, 从而持续保持在线查询的高性能.

在表 8 所示的更新实验中, 最优结果用加粗表示. LA-tree 设置各节点自适应更新阈值 τ 为评分函数较索引构建时升高 30%. 如表 8 所示, 与对比方法, 即定期重构方法相比, 自适应增量更新在效果和效率上均具有显著优势, 读写操作吞吐量提高 12 倍. 在索引更新效率方面, 总用时下降了约 97%. 在查询性能和索引规模方面, 自适应增量更新同样优于其对比方法. 定期重构方法在每次更新时需将原始数据、历史查询负载与新增数据、查询合并, 导致大量冗余, 既使索引规模迅速膨胀, 也拖累了查询性能. 而自适应增量更新通过局部重构避免了这种冗余, 从而在平均查询用时和更新后索引大小上均表现更优. 唯一的代价在于, 自适应增量更新需维护额外的分布统计信息, 使初始索引大小增加约 8%. 但这一开销相对较小, 几乎可以忽略不计.

表 8 LA-tree 更新方法综合比较

LA-tree索引更新	读写操作吞吐量 (TPS)	索引更新总用时 (s)	初始索引大小 (MB)	更新后索引大小 (MB)
定期重新构建	11 000	261	**239**	409
自适应增量更新	**143 000**	**7**	259	**305**
提升	12倍	97%	–8%	25%

实验结论 6. LA-tree 的自适应增量更新方法能够同时保持低查询延迟与高更新效率, 相比定期重构方法大幅减少了更新开销, 从而显著提升了动态场景下的索引性能.

7 讨论: 数据库系统集成学习型多维索引

结合已有的学习型多维索引综述和实验类论文[13,14], 学习型多维索引, 如本文提出的 LA-tree, 可以集成进数据库系统. 本节将讨论集成学习型多维索引所需修改的数据库组件, 以及集成后对索引性能对比实验结果的影响.

集成需在数据库系统的索引管理器、存储、查询优化器这 3 个组件上做相应修改. 具体地, 实现索引管理器中离线构建和在线访问索引的接口, 让数据库系统可以调用 LA-tree. 存储的数据分页逻辑与学习型索引的数据划分结合, 避免学习型索引同一单元内的数据跨越多页, 带来不必要的 I/O 开销. 查询优化器部分更换学习型或基于多维统计信息的代价估计方法, 原因是当前数据库系统内置代价估计方法通常假设数据表各维度独立, 估算多维查询的基数很不准确, 需要更准确的代价估计生成合理的查询计划调用索引.

从实验结果影响的角度分析, 由于更多的外存 I/O 开销, 各学习型索引性能相比内存场景会有一定下降, 但相对传统索引的优势, 以及不同学习型索引之间性能差异的趋势基本不变. 首先, 学习型索引数据扫描比显著相比传统索引有数量级的下降, 显然访问的数据分页数也相应下降, 这直接大幅降低了索引的 I/O 开销, 从而减少查询用时, LA-tree 在这方面优势更加明显. 其次, 学习型索引筛选数据的算法复杂度明显低于基于数值比较的传统索引, 且避免了大量比较操作带来的分支预判错误, 大幅降低索引的 CPU 开销. 此外, 随着数据库系统的缓存机制的不断完善, 外存场景相比内存场景对索引性能的影响将有效降低. 本文的实验结果从筛选和扫描两方面综合比较索引性能, LA-tree 相比其他方法的领先优势将直接体现在 I/O 与 CPU 开销的减少上.

8 学习型索引相关工作

● 学习型单维索引. 在数据库系统中, 每个索引通常作用于数据表的单个属性, 即单维索引, 用不同的平衡树数据结构[15,16]或哈希算法[17,18]等把数据映射为多个分区以加速在线查询处理. 这些传统索引方法具有明显缺陷限制其性能进一步提升, 如基于平衡树的索引在查询时需要大量数值比较操作, 使性能较低; 基于哈希算法的索引通常会丢失保序性, 不适用于数值型属性, 而保序哈希算法[18]的函数计算代价高, 额外空间消耗大, 也不适用于数据库系统. 学习型单维索引的相关工作[19−22]主要基于平衡树索引的思想, 利用机器学习模型替代比较操作以进一步减少在线查询用时. Kraska 等人[19]提出学习式索引 RMI, 在 B 树结构上用线性回归模型 (LM) 直接从分位值映射

孩子代替分位值比较, 消除了 B 树中间节点上的数值比较操作, 有效加快 B 树索引的在线查询效率. Galakatos 等人[20]使用分段式线性回归模型 (PLM) 进一步优化了 RMI 的叶子节点, 提高了属性值到数据条目的映射效率. Kipf 等人[21]直接使用回归模型拟合属性值的积累分布, 一次映射筛选与扫描替代了树结构的多层映射筛选, 大幅提升在线查询效率, 但索引更新性能下降. Ding 等人[22]使用神经网络 (neural network) 代替 RMI 节点上的线性回归模型, 降低映射误差提高筛选效果, 并在此重新设计了节点分裂算法提高数据更新操作的效率. 学习型单维索引对多维属性范围/等值查询提升非常有限, 但是其基于学习模型的思想对后续研究有重要启发.

● 传统多维索引. 对于多维查询, 单维索引因为仅能筛选一个属性的值从而在线查询效率低, 学界和业界提出了多种多维索引结构, 同时按多个属性上的谓词约束条件筛选数据. 其中, 二级索引[23]先后筛选两个不同的属性维度. 四叉树 (quadtree)[24]是一棵二维平面划分树, 递归地将平面中的区域按二维坐标轴四象限方向划分为 4 个子区域. 八叉树 (Octree)[2]是一棵三维空间划分树, 递归地将空间中的区域按三维坐标轴八方向划分为 4 个子区域. 这些结构的共同特点是只能对较少数量的属性做索引, 使用场景局限, 且效率较低. Z-value[25]是一种曲线映射方法, 将二维空间的每个位置降维映射到同一根曲线上的点, 也可拓展到更多维, 缺点是计算量过大且对每个属性值本身的有序性造成部分丢失. R 树 (R-tree)[26,27]的核心思想是将空间对象按最小包围矩形 (MBR) 分层组织, 主要用于二维地图的索引, 也可拓展至多维的数据, 但是当维度增加和值域增加时, 树的空间消耗和在线查询用时都大幅增长. KD 树 (KD-tree)[1]是一棵二叉划分树, 按多个维度轮流均匀划分空间, 树节点过多和划分不够精细都会导致在线查询用时长, 而这两点问题难以兼顾优化. 网格 (grid)[6]按每个维度独立划分空间, 把数据映射到超立方体网格单元, 关键问题是忽视了不同属性之间的相关性, 对数据划分不合理导致索引效率低, 尤其不适用具有稀疏性的高维空间. 总体来说, 传统多维索引数据结构, 除了具有单维索引本身的问题, 还有多维空间映射或划分效率低的问题. 这些导致传统多维索引低效的问题, 是学习型多维索引所要解决的主要挑战.

● 学习型多维索引. 第 1 类是地图查询索引. Wang 等人[28]在 Z-value 二维空间降维的基础上, 结合 RMI 的思路, 利用学习模型提升了查询性能. Qi 等人[29]提出 RSMI 模型, 构建了多层网络结构, 每一层都用 Z-value 方法排序数据条目, 进一步提升了查询性能. 王小丽等人[30]提出 ZFT INDEX 模型, 在 Z-value 基础上使用 PLM 模型提升查询性能.

第 2 类是范围/等值查询索引. Yang 等人[3]在数据库分片 (database cracking)[31]的基础上提出 Qd-tree, 不考虑数据分布, 选择高频查询边界划分数据, 减少在线查询时额外扫描的单元, 由于不能均匀划分, 部分仍然需扫描大量数据, 性能较差. Nathan 等人[4]基于网格数据结构提出 Flood, 在每个维度上独立划分数据, 用 LM 模型拟合多维数据分布, 并面向查询负载调整各维度划分密度, 降低多维索引在线查询的扫描量, 但只适用于各维度分布独立的数据和各查询谓词范围比较一致的查询负载. Ding 等人[5]引入网格树 (Grid-Tree) 根据查询负载划分数据, 实现分治, 并利用条件累积分布 (CCDF) 拟合二维相关性, 提出 Flood 的优化算法 Tsunami, 兼具网格和树的特点, 但无法解决网格结构难以拟合多维相关性的弊端. Davitkova 等人[32]提出 ML-index, 将多维数据聚类成多个不相交超球体再降至单维处理, 并利用学习型单维索引, 适用于较低维度的多维范围查询. Gao 等人[33]提出学习型单调空间填充曲线多维索引 LMSFC, 先使用学习型参数化可变 Z-order 曲线将多维数据降维到一维, 再进行分页优化和页内排序进一步提高数据筛选效果. Pai 等人[34]提出 WaZI, 在 Z-index 基础上提出查询负载感知的学习型多维索引, 先执行自适应分区令划分位置对齐高频查询边界, 再使用 Z-order 曲线进行分区内排序. 这两种方法都基于曲线降维法, 适用于低维情形, 难以拓展到较高维度的数据上. Li 等人[35]提出 LISA, 先使用网格划分多维数据, 再将各网格单元内数据通过映射函数降维至一维, 然后再使用回归模型分片和维护数据, 由于网格和映射函数都难以处理高维情形, 故该索引适用于低维查询.

比较两类学习型多维索引. 第 1 类解决二维地图上的查询, 缺点是各维度的局部单调性有一定损失, 且空间曲线拓展到更高维度计算量很大, 因此不适用于多维属性范围查询, 作为早期学习型多维索引, 其开启了后续进一步研究. 第 2 类适用于典型的多维数据上的多维属性范围/等值查询, 并在 TPC-H Lineitem[28]等属性维度在 5 左右的数据集上测试性能, 挑战在于查询性能的进一步提升, 尤其在更高维度的数据上.

9 总结与展望

本文提出一种查询感知的自适应学习型多维索引 LA-tree, 设计学习型空间划分多叉树结构, 数据划分同时满足均匀划分和查询感知, 并实现高效索引更新. LA-tree 具有 3 个关键技术点: 第一, 针对索引离线构建中数据划分的挑战, 提出多层次查询感知的数据划分方法, 设计的评分函数和多级搜索贪心算法分别解决了扫描比估计和全树节点维度选择的难题, 在多项式时间得出较优解, 数据划分优化效果好且索引离线构建效率高. 第二, 针对多维索引数据筛选效果差、大量数值比较增加在线查询用时的问题, 提出基于学习模型的高效在线筛选方法, 通过设计保序模型误差界限以及线性回归模型与分段线性回归模型混合的学习型空间划分多叉树结构, 分别保证了索引的准确性和高效性. 第三, 提出自适应增量式的索引更新方法, 以学习型树结构快速定位数据更新实现增量更新, 以快速实时合并新查询扫描比与节点评分函数值实现自适应更新, 保持低在线查询用时. 实验结果表明, 我们的 LA-tree 在多个数据集上在线查询性能均优于现有方法, 特别在高维情形下优势显著. 同时, 还通过纵向对比实验, 验证 LA-tree 索引自适应增量更新方法相比朴素更新方法, 不仅有效保持了低查询用时, 而且更新用时极短.

展望未来, 我们计划进一步优化 LA-tree 的实现. 学习型空间划分多叉树结构具有良好的并行化潜力, 由于子树之间的独立性, 我们首先可以尝试实现子树级并行的离线索引构建和在线查询子树级并行执行. 在此基础上, 进一步引入数据库事务中读写锁的机制, 实现多查询并行在线执行以及节点级并行的索引更新, 最终预期有效提高通用的并行事务的执行性能.

References

[1] Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975, 18(9): 509–517. [doi: 10.1145/361002.361007]

[2] Meagher DJR. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-D objects by computer. New York: Rensselaer Polytechnic Institute, Image Processing Laboratory, 1980. https://www.researchgate.net/publication/238720460

[3] Yang ZH, Chandramouli B, Wang C, Gehrke J, Li YN, Minhas UF, Larson PÅ, Kossmann D, Acharya R. Qd-tree: Learning data layouts for big data analytics. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 193–208. [doi: 10.1145/3318464.3389770]

[4] Nathan V, Ding JL, Alizadeh M, Kraska T. Learning multi-dimensional indexes. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 985–1000. [doi: 10.1145/3318464.3380579]

[5] Ding JL, Nathan V, Alizadeh M, Kraska T. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. Proc. of the VLDB Endowment, 2020, 14(2): 74–86. [doi: 10.14778/3425879.3425880]

[6] 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. [doi: 10.1145/348.318586]

[7] Codd EF. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. 1993. https://cir.nii.ac.jp/crid/1570854175548221952

[8] Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Mining and Knowledge Discovery, 1997, 1(1): 29–53. [doi: 10.1023/A:1009726021843]

[9] Hyafil L, Rivest RL. Constructing optimal binary decision trees is NP-complete. Information Processing Letters, 1976, 5(1): 15–17. [doi: 10.1016/0020-0190(76)90095-8]

[10] Even Hallmark. Daily historical stock prices (1970–2018). 2020. https://www.kaggle.com/ehallmar/daily-historical-stock-prices-1970-2018

[11] TPC. TPC-H Version 2 and Version 3. 2019. http://www.tpc.org/tpch/

[12] Ding BL, Chaudhuri S, Gehrke J, Narasayya V. DSB: A decision support benchmark for workload-driven and traditional database systems. Proc. of the VLDB Endowment, 2021, 14(13): 3376–3388. [doi: 10.14778/3484224.3484234]

[13] Zhang JY, Su K, Zhang HC. Making in-memory learned indexes efficient on disk. Proc. of the ACM on Management of Data, 2024, 2(3): 151. [doi: 10.1145/3654954]

[14] Liu QY, Han SY, Qi YL, Peng JS, Li J, Lin LL, Chen L. Why are learned indexes so effective but sometimes ineffective? Proc. of the VLDB Endowment, 2025, 18(9): 2886–2898. [doi: 10.14778/3746405.3746415]

[15] Guibas LJ, Sedgewick R. A dichromatic framework for balanced trees. In: Proc. of the 19th Annual Symp. on Foundations of Computer Science. Ann Arbor: IEEE, 1978. 8–21. [doi: 10.1109/SFCS.1978.3]

[16] Graefe G. B-tree indexes, interpolation search, and skew. In: Proc. of the 2nd Int'l Workshop on Data Management on New Hardware. Chicago: ACM, 2006. 5. [doi: 10.1145/1140402.1140409]
[17] Chierichetti F, Kumar R. LSH-preserving functions and their applications. Journal of the ACM, 2015, 62(5): 33. [doi: 10.1145/2816813]
[18] Fox EA, Chen QF, Daoud AM, Heath LS. Order-preserving minimal perfect hash functions and information retrieval. ACM Trans. on Information Systems, 1991, 9(3): 281–308. [doi: 10.1145/125187.125200]
[19] Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N. The case for learned index structures. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 489–504. [doi: 10.1145/3183713.3196909]
[20] Galakatos A, Markovitch M, Binnig C, Fonseca R, Kraska T. FITing-tree: A data-aware index structure. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 1189–1206. [doi: 10.1145/3299869.3319860]
[21] Kipf A, Marcus R, van Renen A, Stoian M, Kemper A, Kraska T, Neumann T. RadixSpline: A single-pass learned index. In: Proc. of the 3rd Int'l Workshop on Exploiting Artificial Intelligence Techniques for Data Management. Portland: ACM, 2020. 5. [doi: 10.1145/3401071.3401659]
[22] Ding JL, Minhas UF, Yu J, Wang C, Do J, Li YN, Zhang HT, Chandramouli B, Gehrke J, Kossmann D, Lomet D, Kraska T. ALEX: An updatable adaptive learned index. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 969–984. [doi: 10.1145/3318464.3389711]
[23] Lamb A, Fuller M, Varadarajan R, Tran N, Vandiver B, Doshi L, Bear C. The vertica analytic database: C-store 7 years later. Proc. of the VLDB Endowment, 2012, 5(12): 1790–1801. [doi: 10.14778/2367502.2367518]
[24] Samet H. The quadtree and related hierarchical data structures. ACM Computing Surveys, 1984, 16(2): 187–260. [doi: 10.1145/356924.356930]
[25] Gaede V, Günther O. Multidimensional access methods. ACM Computing Surveys, 1998, 30(2): 170–231. [doi: 10.1145/280277.280279]
[26] Guttman A. R-trees: A dynamic index structure for spatial searching. In: Proc. of the 1984 ACM SIGMOD Int'l Conf. on Management of Data. Boston: ACM, 1984. 47–57. [doi: 10.1145/602259.602266]
[27] IBM. The spatial index. 2021. https://www.ibm.com/docs/en/informix-servers/12.10.0?topic=data-spatial-index
[28] Wang HX, Fu XY, Xu JL, Lu H. Learned index for spatial queries. In: Proc. of the 20th IEEE Int'l Conf. on Mobile Data Management. Hong Kong: IEEE, 2019. 569–574. [doi: 10.1109/MDM.2019.00121]
[29] Qi JZ, Liu GL, Jensen CS, Kulik L. Effectively learning spatial indices. Proc. of the VLDB Endowment, 2020, 13(12): 2341–2354. [doi: 10.14778/3407790.3407829]
[30] Wang XL, Chen HH. ZFT INDEX: Learning multi-dimensional index based on piecewise linear regression. Computer Applications and Software, 2024, 41(10): 24–31 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-386x.2024.10.005]
[31] Idreos S, Kersten ML, Manegold S. Database cracking. In: Proc. of the 3rd Biennial Conf. on Innovative Data Systems Research. Asilomar, 2007. 68–78.
[32] Davitkova A, Milchevski E, Michel S. The ML-index: A multidimensional, learned index for point, range, and nearest-neighbor queries. In: Proc. of the 23rd Int'l Conf. on Extending Database Technology. Copenhagen: OpenProceedings, 2020. 407–410. [doi: 10.5441/002/EDBT.2020.44]
[33] Gao J, Cao X, Yao X, Zhang G, Wang W. LMSFC: A novel multidimensional index based on learned monotonic space filling curves. Proc. of the VLDB Endowment, 2023, 16(10): 2605–2617. [doi: 10.14778/3603581.3603598]
[34] Pai S, Mathioudakis M, Wang YH. WaZI: A learned and workload-aware Z-index. In: Proc. of the 27th Int'l Conf. on Extending Database Technology. Paestum: OpenProceedings, 2024. 559–571. [doi: 10.48786/EDBT.2024.48]
[35] Li PF, Lu H, Zheng Q, Yang L, Pan G. LISA: A learned index structure for spatial data. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 2119–2133. [doi: 10.1145/3318464.3389703]

附中文参考文献

[30] 王小丽, 陈华辉. ZFT 索引: 基于分段线性回归的学习型多维索引. 计算机应用与软件, 2024, 41(10): 24–31. [doi: 10.3969/j.issn.1000-386x.2024.10.005]

作者简介

刘佳伟, 博士生, 主要研究领域为智能数据库技术, 机器学习.
范举, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为智能数据库技术, 人在回路的数据准备, 大数据管理与分析.
张超, 博士, 副教授, CCF 高级会员, 主要研究领域为云原生 HTAP 数据库系统, 智能数据库技术.
杜小勇, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为数据管理技术, 语义网技术, 智能信息检索技术.