软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(3):806−830 [doi: 10.13328/j.cnki.jos.005908]

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

学习式数据库系统:挑战与机遇*

柴茗珂[1,2], 范 举[1,2], 杜小勇[1,2]

[1](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)

[2](中国人民大学 信息学院,北京 100872)

通讯作者: 范举, E-mail: fanj@ruc.edu.cn

摘 要: 通用的数据库系统为不同的应用需求与数据类型提供统一的处理方式,在取得了巨大成功的同时,也暴露了一定的局限性:由于没有结合具体应用的数据分布与工作负载,系统往往难以保证性能的最优.为了解决这一问题,“学习式数据库系统”成为了目前数据库领域的研究热点,它利用机器学习技术有效捕获负载与数据的特性,从而对数据库系统进行优化.围绕这一方向,近些年工业界与学术界涌现出了大量的研究工作.首先提出了细粒度的分类体系,从数据库架构出发,将现有工作进行了梳理;其次,系统地介绍了学习式数据库各组件的研究动机、基本思路与关键技术;最后,对学习式数据库系统未来的研究方向进行了展望.

关键词: 数据库系统;机器学习;数据驱动;系统优化

中图法分类号: TP18

中文引用格式: 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. http://www.jos.org.cn/1000-9825/5908.htm

英文引用格式: 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). http://www.jos.org.cn/1000-9825/5908.htm

Learnable Database Systems: Challenges and Opportunities

CHAI Ming-Ke[1,2], FAN Ju[1,2], DU Xiao-Yong[1,2]

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

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

Abstract: Modern database systems provide a general design principle for various data types and application workloads. While gaining great success in the last decades, the principle has a limitation that a database system may not achieve superior performance, if the system cannot be “customized” to the specific data distributions and workload characteristics. To address the problem, learnable database systems have attracted much attention from both industrial and academic communities, with a novel idea of using machine learning to optimize database systems. Along with this direction, extensive efforts have been done very recently to advance the field of learnable database systems. This survey systematically reviews the existing studies from the perspective of database system architecture. A fine-grained taxonomy is provided by categorizing the existing works by their target learnable database components. To help readers better understand each type of learnable components their motivations are presented, demonstrating the insights and introducing the key techniques. Finally, a number of promising future research directions are outlined of learnable database systems.

Key words: database system; machine learning; data-driven; system optimization

* 基金项目: 国家自然科学基金(61632016, U1711261); 中国人民大学科研基金(18XNLG18)

Foundation item: National Natural Science Foundation of China (61632016, U1711261); Research Funds of Renmin University of China (18XNLG18)

本文由人工智能赋能的数据管理、分析与系统专刊特约编辑李战怀教授、于戈教授和杨晓春教授推荐.

收稿时间: 2019-07-20; 修改时间: 2019-09-10, 2019-11-25; 采用时间: 2019-12-18; jos 在线出版时间: 2020-01-10

CNKI 网络优先出版: 2020-01-10 13:34:51, http://kns.cnki.net/kcms/detail/11.2560.TP.20200110.1334.009.html1

自 20 世纪 70 年代开始,通用数据库系统取得了巨大的成功.这种通用设计使系统可以为不同的应用需求与数据类型提供统一的处理方式.然而,针对特定的应用需求,由于没有结合具体的数据分布与工作负载,通用设计往往难以保证性能上的最优.例如:对一组连续的整数进行查询,如果它们顺序存储在磁盘上,使用B树查询需要 $O(\log N)$操作,而使用键-地址映射函数查询只需 $O(1)$操作,因为键本身就对应着偏移量;针对 3 个小表连接操作这样极其简单的负载,穷举法可以较快地找到最优查询计划,而基于代价估计的启发式算法不一定最佳.类似的例子还包括为冲突率很低的事务负载选择并发控制策略,乐观并发控制或许比两阶段锁定法更加理想.因此,越来越多的研究者关注数据驱动的数据库优化,使数据库更适用于具体的应用场景,变得专用.

基于数据驱动的专用数据库可以根据数据与负载为各个组件选择最佳的处理模型,从而提高系统性能.但是实现专用并非易事,长期以来,数据库研究人员一直致力于基于数据驱动的系统优化[1].当给定负载和数据,通过穷举法可以查找出最佳模型,但是穷举需要耗费大量时间成本.另一种常用的方式是数据库管理员(DBA)根据经验为具体应用场景指定最佳模型,然而这种方法有以下的局限性:首先,数据库系统组件复杂、可配置的选项众多,这种方法需消耗大量人力且无法保证可靠性;其次,当数据分布或是负载发生变化时,数据库管理员难以第一时间对模型进行切换,这使该方法的适应性和可扩展性难以满足实际需求;最后,该方法虽然也是根据数据找模型,但从本质上看,更多的是根据模型在不同输入数据下的最终效果来进行选择,并没有深入使用数据本身的特征,如数据分布、数据模式等.因此,难以既快又准地找到最佳模型.

机器学习的迅猛发展给该问题的解决提供了巨大的机会[2].从上文的分析不难看出,专用数据库的核心是根据数据本身的特征或模式选择最佳模型.而机器学习极其擅长从数据中学习规律与模式,这一点已在语音识别、图像分类和自然语言处理等多个领域得到了充分的印证.随着计算能力的提升,机器学习也可以利用这一优势对数据库系统的核心组件进行优化.然而,该优化也存在着巨大的挑战.

- 第一,在语义层面:如何将数据库组件建模成机器学习问题,保证设计的模型提供和传统数据库组件相同的语义;
- 第二,在实现层面:系统应不局限于某一个或某几个特定的应用,因此,如何设计通用的机制,使数据库组件可以根据数据特征选择对应最佳模型,颇具挑战性;
- 第三,在优化层面,如何将机器学习模型与传统数据库优化技术相结合,以应对复杂的实际情况.

为了解决上述挑战,基于数据驱动的学习式数据库系统(learnable database systems)在近些年得到了工业界和学术界的广泛关注.在工业界,亚马逊开发了 OtterTune 系统[3,4],通过机器学习技术实现基于负载特性的自动旋钮配置.Oracle 公司也于 2017 年发布了“无人驾驶”的数据库,可以根据负载自动调优并合理分配资源[5].在 2019 年,华为发布了首款人工智能原生(AI-native)数据库[6],首次将深度学习融入分布式数据库的全生命周期.在学术界,也有多篇研究性论文.然而据我们所知,目前还没有一篇综述文章将这些最新的工作进行梳理.

本文将系统地对学习式数据库系统的现有工作进行调研与综述.首先,根据优化目标组件的不同,对现有研究工作进行细粒度的分类;其次,针对每一类研究工作,将重点介绍研究动机与应用场景,即:通过举例分析如何利用机器学习挖掘数据分布与工作负载中的基本模式,并利用数据模型对组件进行优化;同时,也将归纳典型方法的技术要点,分析它们如何建模机器学习问题、如何针对数据库系统做高效的实现、如何做深入的优化;最后,本文将对未来的研究方向进行展望.由此可见:本文的主要目标是将学习式数据库介绍给更广泛的读者,特别是数据库管理员与不同应用场景下的数据库终端用户,希望他们能够更加深入地理解如何通过机器学习的方法定制自己的专用数据库,以提高系统的性能.

(1) 学习式数据库技术的分类体系

本文基于传统数据库的体系结构对学习式数据库的现有技术进行分类.如图 1 所示,不同颜色表示数据库的不同层级,其中绿色表示语言翻译层、蓝色表示数据存取层、黄色表示数据存储层、橙色表示系统调优.在传统数据库的体系结构中,当用户输入查询时,查询编译器会将输入的查询语句编译成语法分析树,语法分析树经过查询优化器变成了最佳查询计划,执行引擎执行最佳查询计划,处理单个元组,处理期间触发了事务管理器,包括日志管理器和并发控制.执行引擎将处理数据页和系统缓冲区的请求传给了合适的索引、文件、记录

管理器,最后传给缓冲区管理器,缓冲区管理器从磁盘取出所需数据并返回给执行引擎,最后输出查询结果.此外,系统调优,如旋钮配置(knob configuration)调整着数据库的各个方面,如指定查询优化器枚举查询计划算法、并发控制策略、缓冲区大小、数据布局等.

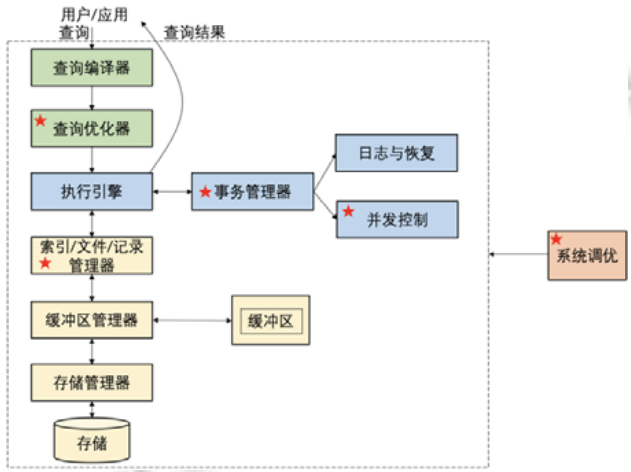


Fig.1 Overview of learnable database system components

图 1 学习式数据库系统组件概览

本文重点关注图 1 中标星的组件(剩余的组件也有基于机器学习的优化[7,8],但由于篇幅有限且现有工作不多,本文不对它们进行深入探讨).以数据库的体系结构自上向下的顺序,标星组件分别是查询优化器、事务管理器、索引结构和系统调优.这些组件在数据库系统中相对重要,同时,已有不少工作试图将它们变成学习式组件.具体来说,本文将现有工作归纳为以下 3 类.

- 学习式索引结构:索引是实现数据高效访问的重要途径,有助于快速得到键的相关信息,如地址、存在与否等.本文首先介绍如何使用机器学习中的回归模型建立起键值与数据之间的对应关系或分类模型判断键值是否存在,从而利用数据分布或特征对索引结构进行优化,使索引变得专用.本文进一步探讨机器学习模型如何与传统的 B 树、布隆过滤器等结构结合,进行效果优化;
- 学习式查询优化器:查询优化是数据库系统的经典难题,例如连接优化是一个 NP 难问题[9].使用机器学习解决查询优化问题的工作可以分为两类.
  - 第 1 类是基于现有的启发式算法框架,通过机器学习模型提高代价估计的准确率.然而,启发式方法往往需要 DBA 基于具体的应用进行调整.此外,执行结果不会向优化器反馈,优化器难以从错误中学习总结,“一错再错”;
  - 因此,第 2 类方法使用强化学习直接端到端输出最佳查询计划.该方法完全颠覆传统思路,利用强化学习多次试错的方式找到最佳查询计划;
- 学习式系统调优:针对具体的应用场景,对数据库系统进行调优十分重要.本文将介绍 3 类学习式系统调优工作.
  - 第 1 类是旋钮配置.数据库有数百个旋钮控制着系统的各个方面,糟糕的配置比好的配置在性能上可能差几个数量级,即使是 DBA 也无法保证准确调优.因此,学习式系统调优通过学习数据分布与工作负载,实现旋钮的自动配置;
  - 第 2 类是利用机器学习进行数据布局,从而决定数据在磁盘的组织存储方式,以更好地满足负载要求;
  - 第 3 类是学习式事务管理器,利用事务负载本身的特征以数据驱动的方式选择最优的并发控制策略.

(2) 与相关综述性文章的区别

学习式数据库系统方兴未艾,据我们所知,系统性地梳理相关文献的工作还不多.Wang 等人[1]和 Kumar 等人[10]曾也写过机器学习(深度学习)在数据库中应用的综述,然而它们的侧重点并非从数据库系统架构的角度去分析如何应用机器学习开发学习式组件.Kraska[7]等人设计了 SageDB 系统,其中大部分数据库系统的核心组件都被学习的组件所替换,例如索引结构、排序算法甚至查询执行引擎.此外,Pavlo 等人设计了 Peloton 系统[11],它基于负载选择数据库各个层次需要执行的操作,如数据布局采取行存或列存、对数据分配地址、对旋钮的配置等,而且该系统还实现了对于负载的预测,以更好地分配资源.但是由于上述两个系统重点在于提出学习式数据库系统的理念与系统实现架构,其侧重点并非梳理与对比实现每一种学习式组件的不同算法.

本文第 1 节介绍机器学习在索引上的应用.第 2 节讨论机器学习在查询优化模型上的应用.第 3 节阐述机器学习技术对数据库系统调优的优化.最后在第 4 节对机器学习在数据库中的应用进行总结,并对未来的研究方向加以展望.

1 学习式数据库索引

索引有助于快速得到键的相关信息,如地址、存在与否等.传统的索引结构有 B 树、哈希索引、布隆过滤器等,它们的设计原则都是通用性的,并不考虑具体数据分布或特征.机器学习可以根据数据分布或特征对索引结构进行优化.学习式索引的思想源自于对索引模型上的抽象和机器学习技术的应用,使索引模型的构建变成监督学习(supervised learning),如回归或分类问题.本节针对以上几种不同的索引结构,介绍机器学习在索引上的不同优化方法,即学习式 B 树索引及哈希索引、学习式布隆过滤器和学习式二级索引.

1.1 学习式B树索引

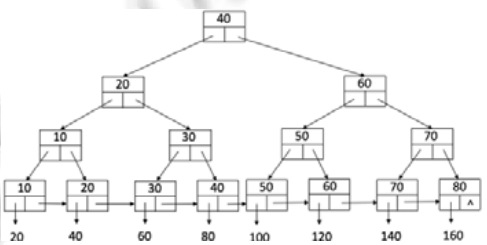
B 树(及其变种,如 B+树、B*树)是数据库系统中最常用的索引结构,给定键 K,索引可以输出对应数据在磁盘上的地址 Pos.由于其设计的通用性,不论数据分布如何,B 树查询时间的复杂度都是 $O(\log N)$.使用数据分布构造的索引可能可以提高查询效率.如图 2(a)为数据表 T,图 2(b)为数据表 T 中各个元组的物理地址.为价格列做索引,对应的 B+树如图 2(c)所示,查找任何一个数据的地址需要 3 次 IO 操作.但是根据数据分布很容易得出 $Pos=2\times K$,此时时间复杂度是 $O(1)$.学习式 B 树索引就是要根据类似或者更复杂的数据分布学到键到地址的映射函数,从而加速索引查找操作.

商品id	价格(元)	最低月销量(个)	最高月销量(个)
1	10	260	600
2	20	220	1200
3	30	180	360
4	40	160	480
5	50	150	600
6	60	140	720
7	70	90	210
8	80	85	240

(a) 数据表 T

商品id	地址
1	20
2	40
3	60
4	80
5	100
6	120
7	140
8	160

(b) 数据地址



(c) 表 T 的 B+树

Fig.2 Traditional B+ index for database systems

图 2 传统数据库系统的 B+树索引

1.1.1 学习式 B 树索引的实现

学习式 B 树索引的核心思想很简洁:根据数据分布学到键 K 到地址 Pos 的映射函数.然而,对于比较简单的数据分布,模型比较容易拟合出该映射函数.但是当数据库中的数据量很大或数据分布较为复杂时,如果还是只用一个模型拟合,该模型往往只可以拟合出总体趋势,但无法准确拟合出具体某一小部分数据的映射关系.如对数据表 2(a)中最低月销量列和对应地址的单个线性函数拟合 $Pos=-6/7\times K+1600/7$,则此时拟合的准确率只有 2/8=25%.如果我们将数据进行拆分,每一小部分数据用对应的模型拟合,那么效果会提升.如 85~90 的拟合函数

为 *Pos*=−4×*K*+500,140~150 的拟合函数为 *Pos*=−2×*K*+400,160~180 的拟合函数为 *Pos*=−*K*+240,220~260 的拟合函数为 *Pos*=−0.5×*K*+150,则此时准确率为 100%.

上述分段拟合的思想总结为 RMI 模型[12],它是建立学习式索引模型的核心.如图 3 所示,RMI 模型目标是将键 *K* 映射到它的地址 *Pos*.RMI 的基本结构是树,一般 2~3 层.树的结点是模型、边是引导到下层模型的判别式.根结点和中间结点的模型起引导作用,一直引导输入 *K* 到叶结点.叶结点模型根据其输入得出地址.RMI 的层数的递增表示数据范围的缩小,直至叶结点可以拟合最小范围的数据分布.这里的根结点往往采用简单神经网络模型,其他结点使用线性回归模型.如果数据分布过于复杂而无法拟合,叶子结点可以直接换成传统的 B 树结构.下面从 RMI 模型的查找与构建两个方面进行介绍.

(1) RMI 模型的查找.当输入键后,算法根据该键的取值找到 RMI 结构中最适合的叶结点,从而输出该键的地址.最后在预测的地址左右允许误差范围内查找是否存有该键:若存在,返回地址;若不存在,返回“空”.如图 3 为根据最低月销量和对应元组地址构建的 RMI 模型.当输入 *K*=90 时,模型 1.1 输出 90,引导至模型 2.1 输出 2×90=180,最后引导至模型 3.1 输出 *P*=(−2×180+500)=140,则地址为 *Pos*=140.在底层地址 140 附近指定误差范围内查到有地址 140 存了 *K*=90.同理,当输入为 *K*=100 时,最后输出为 *Pos*=100.在底层地址 100 附近指定误差范围内未查到 *K*=100,说明数据库没有键 *K*=100.假设模型 3.3 对 *K* 为 155~200 的数据拟合不佳,则可替换为 *K* 为 155~200 的传统 B 树;

(2) RMI 模型的构建.首先,需要确定好静态 RMI 结构,如 RMI 的层数、每个结点的孩子结点个数等;然后,把数据库中所有键-地址对根据数目均分到不同的叶结点内;最后,各个叶结点与其所有连接的上层结点根据其分配到的数据进行训练,使得当输入该数据范围内的键时,模型可以输出对应地址.输出的地址允许有一定误差,误差范围也被作为评价 RMI 模型好坏的指标.如若模型 3.1 输出的底层地址 140 上没有找到键 *K*=90,而在地址 141 或 142 找到,说明误差范围是 1.

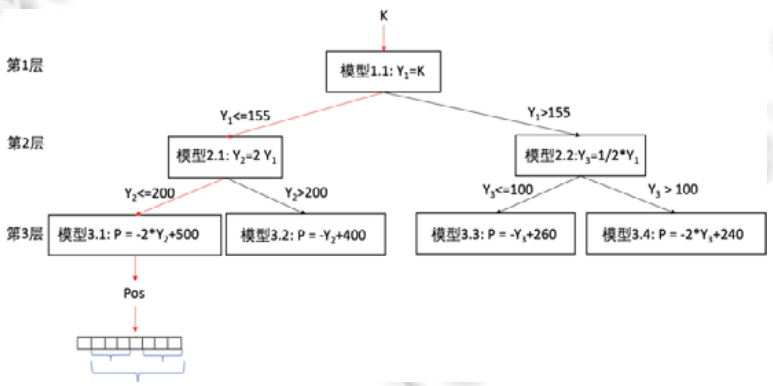


Fig.3 RMI model

图 3 RMI 模型

1.1.2 学习式 B 树索引的改进

基于静态 RMI 的学习式 B 树索引由 Kraska 等人提出[12],为机器学习对索引的优化开辟了先河.但是该模型存在以下 3 个问题:(1) 仅支持静态操作;(2) 数据划分不合理;(3) 仅考虑均匀的查询负载.为解决以上 3 个问题,近期不少学者提出了新思路[13−15].

- 首先,该索引仅支持静态操作.该索引的底层数据由紧密有序数组组织,这导致插入或删除操作需要移动大量数据.此外,由于 RMI 结构是静态且层次设计,当叶结点由于数据分布的改变需要重新训练时,该叶结点所有连接的上层结点都需要重新训练,训练成本过高,所以该模型仅支持查找操作,不支持插入、删除、更新操作.为使模型支持动态操作,Ding 等人[13]提出模型的底层数据不再由有序数组紧密排列,各数据之间存有一定空隙,使得插入操作无需移动过多的数据;其次,RMI 结构不再是静态的,若底层数据因动态操作而排列过于紧密或稀疏时,RMI 子模型可以进行拆分或合并.RMI 结构改变后,相

关子模型需重新训练,底层数据还会根据新模型的输入输出进行重排,以提高查询准确率.而 Li 等人[14]为每一个叶子模型配了一个缓冲区,以存放新插入的数据.当缓冲区过满或缓冲区和模型底层数据都为空时,做类似上述 RMI 子模型的拆分或合并操作.此外,为减少训练时间,Tang 等人[15]提出对以前训练好的模型可以再利用.所有训练好的模型经过数据分布归一化存在于缓冲区,当有新模型需要训练时,其初始模型定为缓冲区中归一化数据分布最相似的模型,以提高训练速度;

- 其次,该索引数据划分不合理.采用按个数均分的方式划分数据,而未考虑数据之间的关系远近.而模型往往擅长拟合关系较近的数据,所以数据之间的关系远近也应该纳入数据划分与否的指标之一.Li 等人[14]考虑了用数据之间的关系远近来更有效地划分数据,这里的远近用相邻数据点之间的欧式距离来评估;
- 最后,RMI 索引仅考虑均匀的查询负载.该模型假设对所有键的查询概率都相同,即没有热键.而现实查询往往存在热键,它们的查询准确率对模型的准确率影响最大,所以应该对热键有特殊的处理以提高模型性能.Tang 等人[15]对热键进行了特殊的处理以提高模型性能,包括增加热键在训练集的出现次数和增大热键和其他键之间的距离两种方式.如训练集中有 3 组键-地址对{(a,0),(b,1),(c,2)}且查询频率为 1:2:1.那么方式 1 更新训练集为{(a,0),(b,1),(b,1),(c,2)},增加对热键的拟合强度.方式 2 更新训练集为{(a,0),(b,1.5),(c,3.5)},提高相邻点距离以减少该子模型内数据量,减少拟合难度.

上述学习式 B 树索引的构建思想也可以使用在学习式哈希索引上.哈希索引利用哈希函数将键分配到介于 0 到(B–1)号哈希桶中,其中,B是桶的数目.不合适的哈希函数会提高哈希索引的冲突率,如当所有键都分配到了一个哈希桶中时,查找键就变成了 $O(N)$操作.若哈希函数利用数据分布进行设计,就可极大降低冲突率.学习式哈希索引就是要利用数据分布设计哈希函数.同学习式 B 树索引,学习式哈希索引也构建 RMI 模型学到键到地址的映射,即哈希函数.

1.2 学习式布隆过滤器

布隆过滤器用于判断给定键 K 是否存在于数据库,如图 2(a)的数据表中没有 5 元的商品但有 10 元的商品.传统布隆过滤器内部使用 m 维二元数组和 n 个哈希函数将一个键映射到 m 个数组位置中的 n 个.要向集合中添加元素时,将该键提供给 n 个哈希函数,并将返回位置的位设置为 1;要测试键是否是该集合的成员时,该键将再次输入到 n 个哈希函数中,如果返回位置的任何位是 0,则该键不是集合的成员.布隆过滤器确保不存在假阴性,但存在假阳性,这也是学习式布隆过滤器必须达到的基本要求.

1.2.1 学习式布隆过滤器的实现与优化

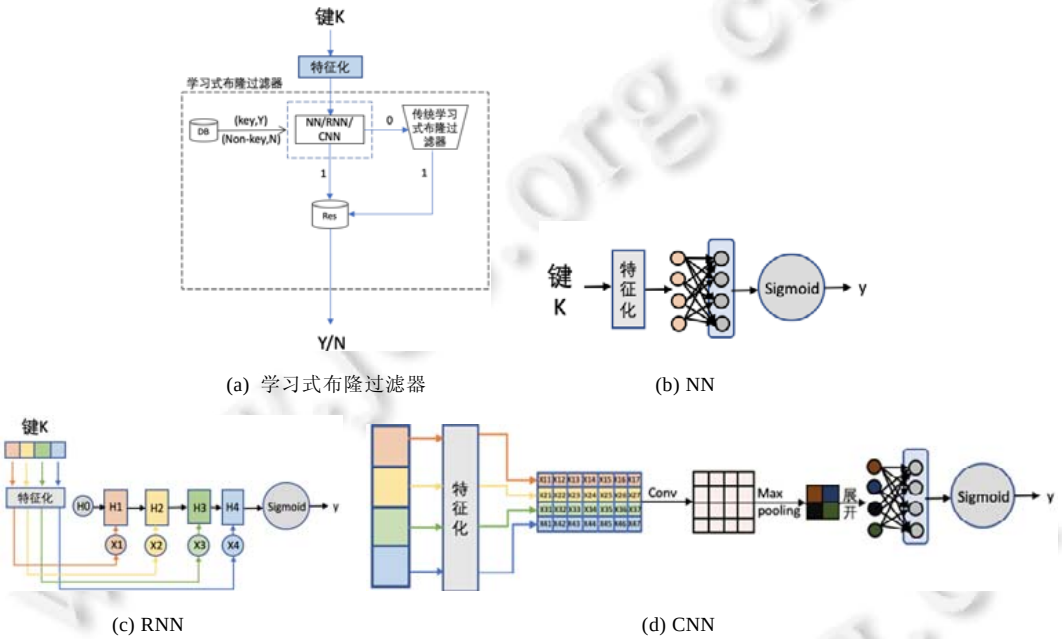
学习式布隆过滤器本质是机器学习中的二分类模型,即根据键的特征将键分为存在类或非存在类.学习式布隆过滤器与分类器示意如图 4 所示.

(1) 学习式布隆过滤器的构建.学习式布隆过滤器的本质是二元分类模型,数据集使用 $D=\{(x_i,y_i=1)|x_i\in E\}\cup\{(x_i,y_i=0)|x_i\in U\}$,其中,$E$ 是数据库键的集合,U 是的非键集合.数据集可根据数据库查询历史构造.二元分类模型可使用神经网络(NN)、递归神经网络(RNN)或卷积神经网络(CNN)[16,17].

- 若键为单字符或者数字类型,只可通过全连接神经网络进行分类.键经过特征化后(如 one- hot 编码)变成一个向量,该向量经过 NN 和 *sigmoid* 激活函数后产生一个概率,表示它属于存在类的可能性;
- 若键为多字符字符串,可以使用循环神经网络或卷积神经网络,捕捉字符之间的顺序特征,使分类模型更加准确.输入键经过其每个字符的编码后成为多个向量,将它们按序输入 RNN 模型,然后过 *sigmoid* 激活函数后产生属于存在类的概率.同理,输入键特征化后的多个向量按序填入 CNN 的起始二维向量,然后经过卷积层、pooling 层、全连接层,最后通过 *sigmoid* 激活函数产生概率.上述模型最后都会产生一个概率以决定输入是否属于存在类,并经过训练以最小化对数损失: $L=\sum_{(x,y)\in D} y\log f(x)+(1-y)\log(1-f(x))$;

(2) 学习式布隆过滤器的查找.模型输入查找键,若模型输出 1,则表示数据库有该查找键;若输出 0,为保证不存在假阴性,还需将该查找键过传统布隆过滤器,若过滤结果是该键不存在,才表示数据库无该查找键;

(3) 该模型实质是利用键的特征对键是否存在于数据库进行区分[12].若查找键和数据库中的键在特征上很类似,即使模型学得很好,假阳性比率也很高.为解决这一问题,Michael 等人[18]在过分类模型之前再加一个传统布隆过滤器,先过滤部分非数据库键.



(a) 学习式布隆过滤器 (b) NN

(c) RNN (d) CNN

Fig.4 Learnable Bloom filter

图 4 学习式布隆过滤器

1.3 学习式二级索引

二级索引描述了数据表中同元组不同列之间的数据关系,如普通列和主键列.在最常查询的列之上构建二级索引有助于加速查询.如对图 5(a)的数据表 *T*2 执行 SQL 语句 select*from *T*2 where *Sales*_min Between 18 AND 60.若在 *Sales*_min 列上有对主键的二级索引,即如图 5(b),则可以快速返回结果是 id 为 6,8,10,15 的元组.这些二级索引往往由数据库管理员创建或由查询优化器自动创建,具体表示为 B 树的形式.但是在数据库中管理多个二级索引会占用大量存储空间.传统优化方法有限制二级索引的个数和压缩二级索引,它们节省了有限的空间但是可能导致查找操作的高开销.

id	Sales_min(个)	Sale_max(个)
2	4	30
5	10	90
6	18	67
8	24	33
10	30	43
15	60	65
30	120	128

(a) 数据表 *T*2

id	Sales_min(个)
2	4
5	10
6	18
8	24
10	30
15	60
30	120

(b) *Sales*_min 二级索引

Fig.5 An example of secondary index

图 5 二级索引示例

1.3.1 学习式二级索引的实现与优化

学习式二级索引的核心是根据数据分布学到某列(基列 N)的键范围值到同元组下另一列(主列 M)键范围值的映射函数.类似第 1.1 节的学习式 B 树索引,如果数据分布很简单,用简单的单个模型就可以拟合该映射函数.但当数据分布比较复杂时,需要使用 TRS-Tree 模型[19]拟合.

如图 6 所示,TRS-Tree 的目标是将列 N 的范围值映射到同元组的列 M 的范围值.与 RMI 类似,TRS-Tree 的基本结构也是树.树的结点是模型、边是引导到下层模型的判别式.输入 K 根据根结点和中间结点一直被引导到叶结点.叶结点用特定范围内列 N 到列 M 的近似线性映射来表示它们的关系,即满足公式(1):

$$N=\beta m+\alpha\pm\epsilon \tag{1}$$

其中,n 是列 N 指定范围内的任意值;m 是同元组下列 M 的值;β,α和ϵ分别表示函数的斜率、截距和置信区间,它们都是待训练的参数.如图 6 为表示案例图 5(a)中 *Sales*_min 列对 *id* 列映射的 TRS-Tree 模型.当输入 6 时,即 *Sales*_min=6,引导至 Y_1=6,再引导至 Y_2=12,再引导至 Y_3=36,最后确定使用范围 A 的函数 n=2×m,则:

$$id=Sales\_min/2=3.$$

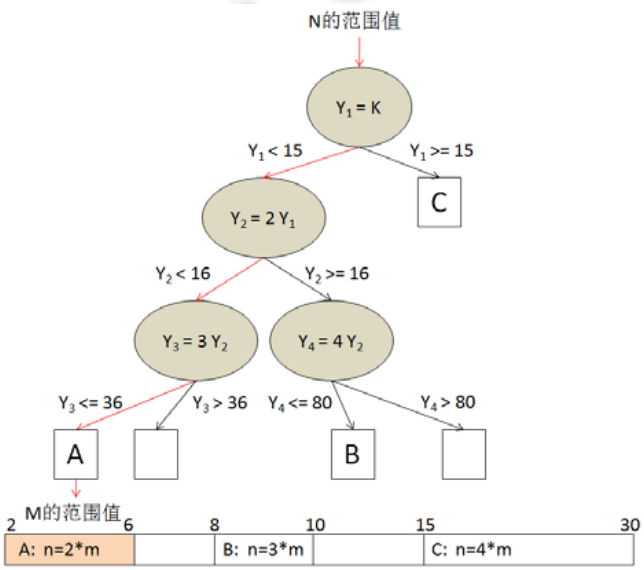


Fig.6 TRS-Tree model

图 6 TRS-Tree 模型

学习二级索引模型的构建:利用 FIFO(先进先出)队列以自上而下的方式构建 TRS 动态树.首先把主列和基列的所有数据放入根结点,利用线性回归模型拟合两列的数据关系,将不满足拟合函数的数据对存入异常缓冲区.当异常缓冲区过满时,则将该结点拆分成特定个数的等数据范围孩子结点.所有孩子结点进 FIFO 队列等待模型拟合,若拟合不佳则再次拆分.多次拟合和拆分,形成 TRS-Tree,其每个叶子结点可以为不同数据范围的基列映射至同元组下的主列.

1.4 总 结

如表 1 所示,不同的学习式索引根据其应用场景可以总结成不同的机器学习模型.学习式 B 树索引、学习式哈希索引和学习式二级索引都可以作为回归模型,而学习式布隆过滤器是典型的二分类模型.回归模型对键-地址或键-键进行拟合,对比较简单的数据分布,模型可以较好地拟合;相反,模型拟合效果较差.所以提出层次性树状结构(RMI 模型或 TRS-Tree 模型),每一个模型只需要拟合小范围数据,降低了拟合难度.分类模型根据键的特征,使用 NN,RNN 或 CNN 进行二分类.此外,虽然本文强调了机器学习在索引上的应用,这不意味着完全摒弃

传统索引模型,而可以相互结合.如:当 RMI 模型无法拟合键-地址对时,叶子结点可以换成 B 树.学习式索引模型还有很多开放式问题,例如:如何更好地应对数据的更新(插入、删除、修改),如何使用深度学习提高模型的预测能力,如何系统性地实现到现有数据库系统架构中,等等.

Table 1 Summary of learnable index model

表 1 学习式索引模型总结

模型	输入	输出	机器学习模型	核心
学习式 B 树索引、学习式哈希索引	键	数据库地址	回归模型	RMI
学习式布隆过滤器	键	是否存在于数据库	分类模型	特征分类
学习式二级索引	键	同元组下另一列的键	回归模型	TRS-Tree

2 学习式查询优化模型

查询优化是数据库系统的核心问题:针对同一个查询,不同查询计划的性能有可能相差几个数量级[20].查询计划由多个查询子计划组成,因此,查询优化可以抽象为查询子计划的选择和排序.传统的查询优化方法是对查询子计划的代价进行估计,进而用启发式算法,如动态规划法、贪婪法等对查询子计划进行选择和排序,从而形成最终的查询计划.然而,这些方法有以下两点局限性:第一,代价估计需要对查询子计划的基数或选择率进行准确估计,而传统基于数据库统计信息的估计模型往往无法保证估计的准确性;第二,使用启发式算法可能会错失最佳查询计划,因为它无法从执行结果的反馈中学习,可能每次都会选择相同的错误查询计划.为了解决这些问题,学习式查询优化器在近些年得到了广泛的研究.第 2.1 节介绍学习式代价估计方法,该方法使用机器学习中的回归或核密度估计的方法估计基数或选择率.第 2.2 节介绍强化学习查询优化模型,该方法完全颠覆传统思路,利用强化学习多次试错的方式找到最佳查询计划.

2.1 学习式代价估计模型

2.1.1 传统代价估计模型

查询代价是数据库系统执行查询时的 IO 代价、CPU 代价和内存代价的总和,通常主要取决于磁盘访问的 IO 代价.由于真实的查询代价只有在执行查询后才可以获得,因此传统数据库系统一般采用事先估计法,并将问题进一步转化为估计中间结果的大小,因为查询代价往往与中间结果的大小成正比.常见的方法有基数(cardinarlity)估计,即估计查询子计划的结果数量,或是选择率(selectivity)估计,即估计查询子计划结果数量占输入数据数量的比例.由于查询操作中连接操作最为复杂且时间占比最大,所以本节侧重介绍连接操作的基数(或选择率)估计方法.

传统基数估计方法主要有基于直方图和基于草图两种方式.基于直方图的估计方式[21]采用启发式规则.例如:表 R 和表 S 通过属性 Y 执行连接操作,则操作后结果数据的基数见公式(2):

$$T(R\triangleright\triangleleft S)=T(R)T(S)/\max(V(R,Y),V(S,Y)) \tag{2}$$

其中,$T(R)$和 $T(S)$分别为 R 表和 S 表的元组数,$V(R,Y)$和 $V(S,Y)$分别为 R 表和 S 表中 Y 属性的取值的个数.上述信息都可以由直方图获得.

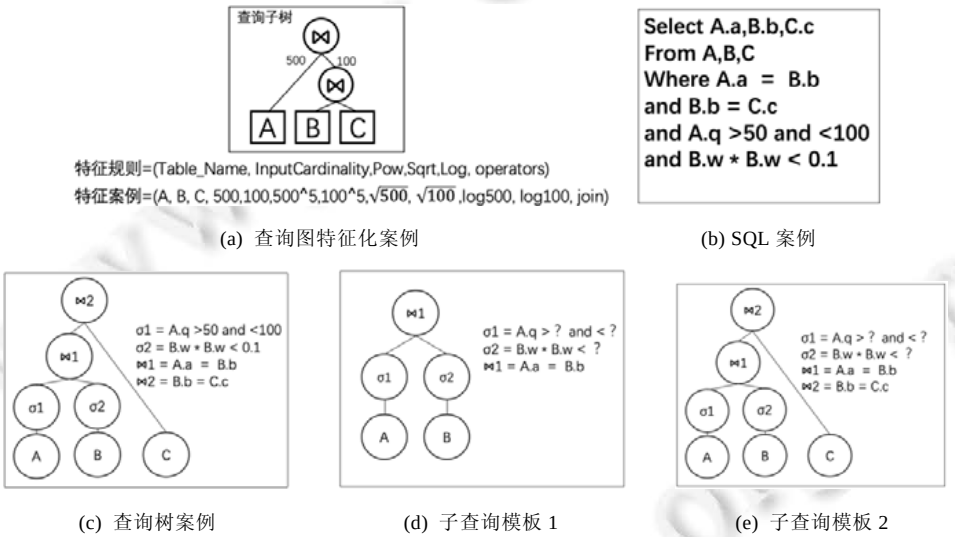
不难看出,上述估计方法基于的一些基本假设,如 Y 属性的取值在元组上均匀分布、两个表格的列之间相互独立,往往不符合实际的数据分布.因此,研究者又提出了基于草图的估计方法,包括 FM[22],LinearCount[23],Loglog[24],HyperLoglog[25]和 MinCount[26].它们将所有元组遍历映射到哈希位图,根据位图连续的零或一的次数推断出基数.这些方法可以有效地估计基数但不适合估算范围查询的基数.然而,上述方法构造代价都过大,因为构造直方图或哈希位图都需要遍历所有元组.

2.1.2 学习式代价估计模型

学习式代价估计模型是构建查询子计划到基数或选择率的回归模型.回归模型主要有两类,分别是回归函数和神经网络,它们的输入都是查询子计划对应查询图的特征化表示.使用查询图而非查询语句进行特征化表示,是因为查询图相比查询语句更加直观,而且可以形象地描述现在以及之前的查询操作.回归模型的训练集都

源于数据库中曾经执行过的负载与其对应基数或选择率对.此外,由于选择率相当于每个元组被选概率之和,所以对它的估计还可以利用元组采样和核密度估计(kernel density estimation,简称 KDE)的方法[27,28].

首先介绍基于回归函数的估计法.Wu 等人[29]采用泊松回归模型.该方法首先将每一个查询图进行特征化,表示成一个向量.如图 7(a)所示,查询图的特征使用数据表、根结点子树的基数等操作.进而采用泊松回归模型对基数进行估计.泊松分布是统计与概率学里最常见的离散概率分布,满足泊松分布的因变量可以利用泊松回归模型进行拟合,此时自变量为查询图向量、因变量为代价.然而,查询计划基数估计问题往往非常复杂,而泊松回归模型相对简单而导致拟合效果不佳.所以该方法进一步对查询计划子图进行基于模板的划分,每一个模板下的子计划的代价都由该模板对应的回归模型进行估计.如图 7 给出的例子,给定图 7(b)中的 SQL 语句,数据库自带的查询优化器输出的查询树方案如图 7(c)所示,根据该树可以拆分出两个子树模板(如图 7(d)和图 7(e)所示),分别对应不同的泊松回归模型.针对新输入的查询语句,该方法会找到所有可用的模板进行基数估计,然后以代价最小的计划作为其最佳查询子计划.不难看出,该方法的局限性在于往往需要过多模板,而且不支持未知模版查询语句的代价估计.



(a) 查询图特征化案例 (b) SQL 案例

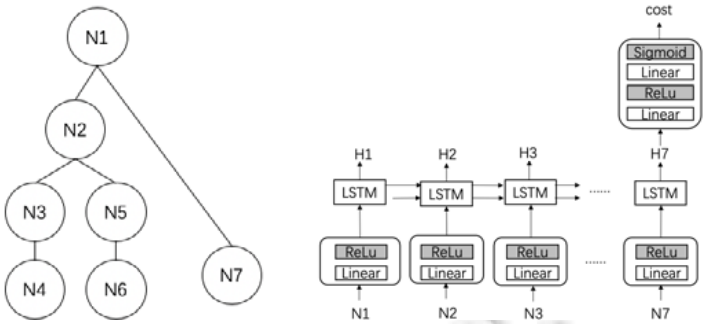
(c) 查询树案例 (d) 子查询模板 1 (e) 子查询模板 2

Fig.7 Query subplan template based cardinality estimation

图 7 基于查询子计划模板的基数估计案例

为避免使用多模型,一些研究[30,31]提出用深度学习解决该问题.Dutt 等人[30]将查询图向量直接输入神经网络进行回归.Sun 等人[31]使用了更复杂的 LSTM 模型,虽然也考虑查询树,但是其思路是对树中每一个结点做特征化和嵌入化,再按照树的深度优先(DFS)的顺序将结点的嵌入化表示(embedding)输入到 LSTM 中,最后输出基数.例如还是执行图 7(b)中的 SQL 语句,且仍旧考虑图 7(c)中的查询计划.该方法采用类似图 7(a)中的特征提取方法,将每个结点都根据谓词操作、元信息等进行特征抽取形成向量.进而,将结点按照树 DFS 排序,得到节点序列 $N_1,N_2,\ldots,N_7$,如图 8(a)所示.然后,按照图 8(b)所示,将排序后的节点按序输入进 LSTM 模块,最后在估计层神经网络输出归一化基数.

还有一些研究侧重选择率的估计,将选择率建模成元组满足查询的概率分布函数的积分.例如,Heimel 等人采用核密度估计(KDE)的方法,基于采样对概率密度函数进行拟合[27,28].图 9 给出了该方法的示意,针对数据表 A,图 9(b)所示的 SQL 查询可以筛选出 5 个元组(表中标红),因此真实的选择率为 5/9.KDE 方法首先对表 A 进行采样,并将样本中满足查询的元组记为 $S=\{s_i\}$,比如本例中可能得到 $S=\{t_1,t_4,t_7\}$.基于 S 集合,该方法拟合出如图 9(c)所示的概率密度函数.可以看出,样本点 S 对应的概率密度函数相对较高.最后,针对概率密度函数进行积分(对应求图中阴影部分面积),则得到选择率的估计.



(a) 排序后结点 (b) 基于 LSTM 的基数估计模型

Fig.8 LSTM based cardinality estimation

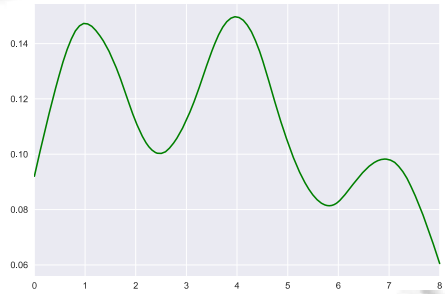
图 8 基于 LSTM 的基数估计

A.id	A.a	A.b
0	2	8
1	20	12
2	6	44
3	80	28
4	20	17
5	13	14
6	24	16
7	50	5
8	80	60

(a) 数据表 *A*

```
Select *
From A
Where A.a  > 10
and A.b <20
```

(b) SQL 案例



(c) KDE 拟合

Fig.9 KDE based selectivity estimation

图 9 KDE 拟合选择率案例

形式化地,为了对概率密度函数进行估计,KDE 的方法首先将元组建模成 d 维空间上的点,表示为 $\vec{t}$.进而,在采样出的 S 集合基础上,使用公式(3)估计概率密度函数:

$$\hat{p}_H(\vec{t})=\frac{1}{|S|\cdot|H|}\sum_{i=1}^{|S|}K(H^{-1}[\vec{s}_i-\vec{t}]) \tag{3}$$

其中,$|S|$是 S 集合的大小;K 是核函数,定义了局部概率分布的形状;矩阵 H 是带宽矩阵,控制了局部概率分布的扩展; $\vec{s}_i-\vec{t}$ 表示样本点 s_i 与元组 t 在 d 维空间上两个向量的差.

对于表 A 中的所有元组,表示为 $\Omega=\{\vec{t}_0,\vec{t}_1,...,\vec{t}_7\}$,SQL 查询的选择率可以用积分公式来(4)估计:

$$\hat{p}_H(\Omega)=\int_{\Omega}\hat{p}_H(\vec{x})\mathrm{d}\vec{x} \tag{4}$$

通过实验发现,核函数 K 的形状一般对估计质量影响不大.但是带宽矩阵 H 会影响选择率的估计:带宽过小,则拟合曲线过于尖锐;反之过于平缓.所以带宽矩阵是模型训练时需要调整的参数,并通过训练得到(最小化真实选择率和估计选择率的差值).

KDE 也可以进一步应用到多表操作上[32].以两表操作 $Q=\sigma_{c_1}(R_1)\triangleright\triangleleft_{R_1.A_1=R_2.A_1}\sigma_{c_2}(R_2)$ 为例,选择率可以使用公式(5)估计:

$$\widehat{Sel}(Q)=\frac{1}{|R_1|\cdot|R_2|}\cdot\sum_{v\in A}\hat{p}_1(A_1=v\wedge c_1)\cdot\hat{p}_2(A_1=v\wedge c_2) \tag{5}$$

其中, $\hat{p}_1(A_1=v\wedge c_1)\cdot\hat{p}_2(A_1=v\wedge c_2)$ 的估计即可转化为单表 $R_1(R_2)$上的选择率估计.由于使用采样进行概率分布估计依赖于样本的好坏,MullerMK 等人[32]进一步提出同时使用直方图和采样以提高估计准确率.

2.2 强化学习查询优化模型

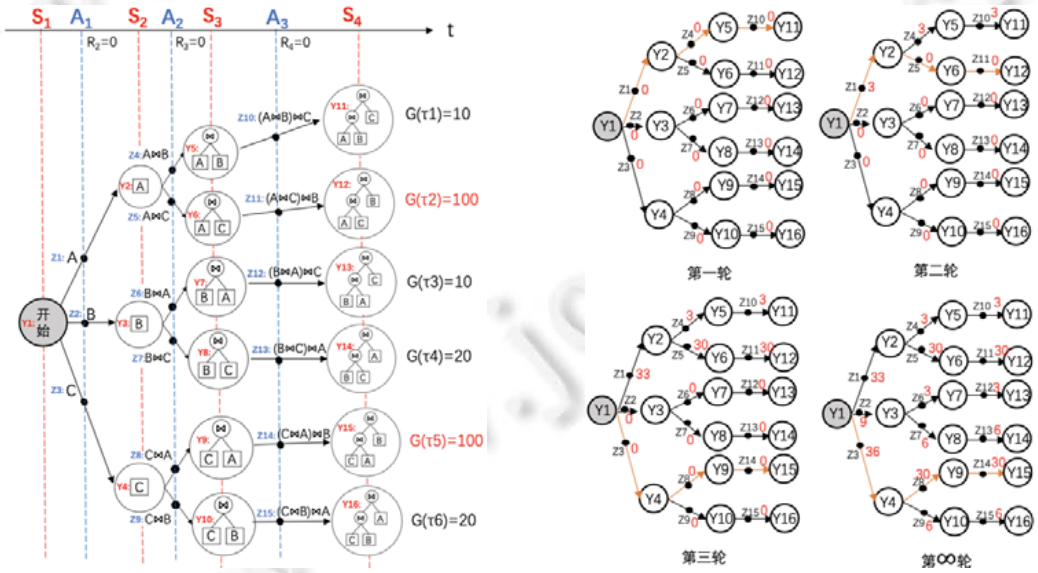
2.2.1 查询优化的强化学习模型

强化学习是指代理(agent)通过试错的方式不断地与环境交互学习,最大化与环境交互过程中获得的累计奖励,从而找到解决问题的最优策略的方法[33].几乎所有的强化学习问题都可以转换为马尔可夫决策过程(MDP).MDP 是由⟨S,A,P,R,G⟩构成的一个五元组,其中,

- S 表示有限状态集,对任意时刻 t 的状态 S_t,都满足 $S_t \in S$;
- A 表示有限动作集,对任意时刻 t 的状态 A_t,都满足 $A_t \in A$;
- P 是策略函数,其中,$P_s^a = P(A_t = a \mid S_t = s)$,即 t 时刻下状态为 s 时取动作 a 的概率,如随机选择.策略函数通过强化学习的不同决策算法生成;
- R 是奖励函数,若 t 时刻下状态为 s 时取动作 a,则 t+1 时刻获得的奖励为 $R_{t+1} = R_s^a = R(S_t = s, A_t = a)$,具体奖励取值由环境根据其变化给定;
- G 表示收获,即代理完成一个完整的状态序列 τ(即从开始状态到终止状态)所收获的累计奖励,$G(\tau) = \sum_{t=1}^{t=T-1} \gamma^t R_{t+1}$.其中,为权衡立即回报和将来长期回报之间的重要性,引入 $\gamma \in (0,1]$ 作为折扣因子.强化学习目标就是要找到一个序列 τ,使得累计奖励最大.

数据库系统查询优化可以使用马尔可夫决策过程进行建模.例如,针对关系表 A,B 和 C 的三表连接操作,可以将其建模为如图 10(a)所示的马尔可夫过程.

- 每个空心圆圈表示一个状态,所有的状态构成了状态集 $S=\{Y_1,Y_2,\ldots,Y_{16}\}$;每个状态表示一种可能的查询子计划,如状态 Y_{10} 表示将表 C 和表 B 做连接后的查询计划;
- 每个实心点表示一个动作,所有的动作构成了动作集 $A=\{Z_1,Z_2,\ldots,Z_{15}\}$;每个动作表示做连接操作,如 Z_{15} 表示在状态 Y_{10} 的结果上再与关系表 A 做一次连接操作.



(a) 查询优化马尔可夫决策过程 (b) 查询优化强化学习过程

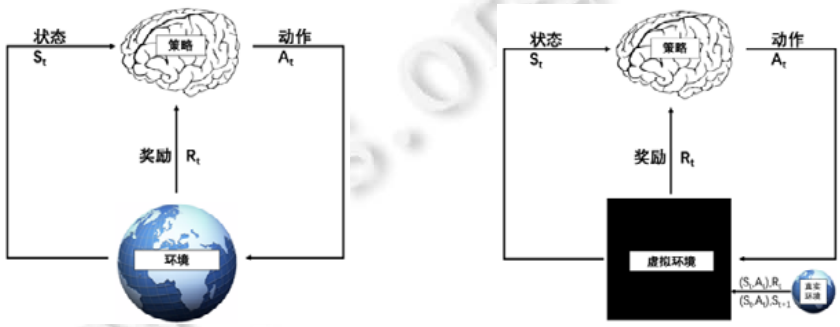
Fig.10 Query optimization process case

图 10 查询优化过程案例

在起始状态 $S_1=Y_1$ 时,根据策略函数从 Z_1,Z_2,Z_3 中选择一个作为执行动作 A_1,假设 $A_1=Z_1$.由于此时相应的查询子计划没有在数据库真实运行,所以数据库(环境)反馈的的 $R_2=0$,且状态由 Y_1 变为 Y_2,即 $S_2=Y_2$.同理,再根据策

略函数选择 A_2,假设 $A_2=Z_4$,则 $R_3=0$ 且 $S_3=Y_5$.最后选择 $A_3=Z_{10}$,则 $S_4=Y_{11}$.此时已经生成了完整的查询计划,数据库运行 Z_{10},根据运行时间等数据库指标反馈 $R_4=10$.取 $\gamma=1$,则该查询计划序列 τ_1 的收获为 $G(\tau_1)=10$.如果策略函数比较理想,则会生成序列 τ_2 或 τ_5,它们的收获更大.

强化学习就是要与环境不断地交互,根据反馈 R 更新策略函数,使得代理可以既快又准地找到最佳序列.如图 11(a)所示,强化学习有 6 个基本要素$\langle S,A,P,R,F,E\rangle$,前 4 个分别对应了 MDP 的 S,A,P 和 R.后面的两个要素:环境 E 用于执行动作 A 并反馈奖励 R 和生成新状态 S,决策算法 F 用于更新策略函数 P.类似 MDP,强化学习的过程如下:在任意时刻 t,代理根据当前状态 S_t,利用策略函数 P 选择要执行的动作 A_t,环境 E 执行 A_t 并反馈奖励 R_{t+1},决策算法 F 利用 R_{t+1} 更新策略函数 P,同时进入下一状态 S_{t+1}.反复上述操作,直至到达终止状态.上述过程会经历多轮,直至策略函数 P 收敛或时间结束,此时认为选择的序列是最佳序列.



(a) 传统强化学习架构 (b) 基于模型的强化学习架构

Fig.11 Architecture of reinforcement learning

图 11 强化学习架构

决策算法更新策略函数时常会使用一些价值函数,如状态价值函数 $V(S)$和行为价值函数 $Q(s,a)$.$Q(s,a)$反映了在状态 s 下执行动作 a 对收获 G 的影响:$Q(s,a)$越大,说明在状态 s 下执行动作 a 后可能得到的收获越大.同理,$V(s)$反映了状态 s 对收获 G 的影响.策略函数会随着价值函数的改变而改变,所以决策算法通过更新价值函数来更新策略函数.如例 10(b)是图 10(a)对应的强化学习过程,使用 $Q(s,a)$来完成策略函数的更新.假设环境 E 为数据库;最开始,所有的 Q 值都为 0;策略函数 P 是在给定状态下以 p 的概率选择 Q 值最大的动作、$(1-p)$的概率随机选择;决策算法 F 是在未执行过现生成的查询计划的条件下,对当轮执行序列上所有访问过的 Q 值加上奖励 R_t 与序列长度的比值.则强化学习过程(包含策略函数的更新)如下.

- 第 1 轮:由于所有的 Q 值都为 0,策略函数随机选择序列,假设选择 τ_1(橙色路径),数据库执行后收获为 $G(\tau_1)=10$,则该序列上所有 Q 值都更新为 10/3=3;
- 第 2 轮:由于 $Q(Y_1,Z_1)>Q(Y_1,Z_2)=Q(Y_1,Z_3)$,所以策略函数有较大概率选择动作 Z_1.但在 Y_2 状态时,虽然 $Q(Y_2,Z_5)<Q(Y_2,Z_4)$,但由于有$(1-p)$的概率随机选择,所以策略函数选择到了动作 Z_5,最终选择序列 τ_2,数据库执行后收获为 $G(\tau_2)=100$,τ_2 序列上所有 Q 值都加 100/3=30;
- 第 3 轮:由于有$(1-p)$的概率随机选择,所以选择了序列 τ_5;
- 第∞轮:如果时间充裕,可以进行无穷轮,那么所有 Q 值都被更新且收敛.此时很明显,τ_5 是最佳序列.但强化学习常常会有时间限制,所以不能更新无穷轮,如仅更新到第 2 轮,那么 τ_2 就被认为是最佳序列.

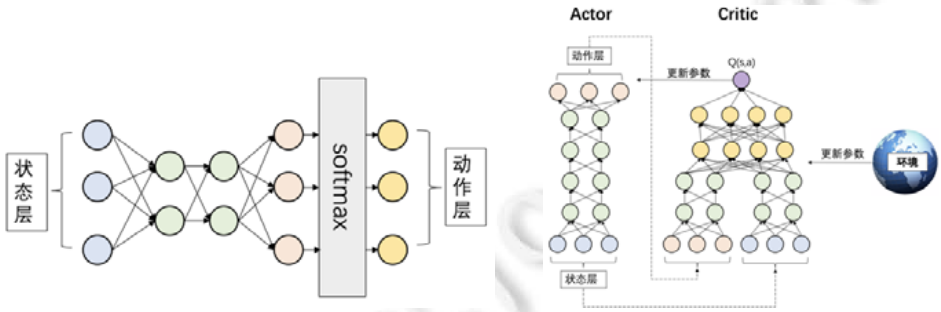
上述仅是比较简单的决策算法,强加学习有各种更复杂的决策算法,不同的决策算法会以不同的方式更新策略函数,最终选择不同的序列.本文重点介绍 Q-Learning[33],Policy Gradient[34]和 DDPG[35],本文仅涉及算法利用部分,不探讨算法探索部分.而且这些算法没有明确地孰好孰坏,根据实际应用场景而定.

- Q-Learning:典型的基于值的算法.Q 指 $Q(s,a)$,即在某一时刻的状态 s 下执行动作 a 后能够获得的收益期望,这里的收益期望不是采取动作后的立即奖励 R,而是行为价值函数.算法会以状态为行、动作为列

构建一张二维 Q 表,用于存储 Q 值.一开始,所有 Q 值都为 0.虽然原始 Q-Learning 算法使用 Q 表对 Q 值进行存储,但其实也可以利用回归模型实现状态和动作到 Q 值的映射,以减少基于表的存取操作开销.算法每次都会执行当前状态 *S_now* 下 Q 值最大的动作 *A_now*,执行完毕后,根据立即奖励 R 和下一状态的最大 Q 值 $\mathrm{Max}_a Q(S_next,a)$利用时间拆分算法更新当前的 $Q(S\_now,A\_now)$,经过多轮更新,Q 表会收敛.此时,每一个 Q 值都接近真实值.最后,基于 Q 表每次选择最大 Q 值的动作即构成了最佳动作序列.但是 Q-Learning 算法作为基于值的算法,每次会取 Q 值最大的动作,即一旦 Q 表收敛每次选择的动作唯一,所以它无法支持类似剪刀石头布这种需要随机选择动作的游戏;

- Policy Gradient:基于策略的算法.不同于基于值的算法,它无需先学 Q 值,再取最大的动作,Policy Gradient 直接学习策略 π_θ即在任意状态下采取各个动作的概率,其中,θ是策略参数.如图 12(a)所示,当前状态经过特征化后输入至神经网络,最后经过 *softmax* 输出至动作层,动作层中的每个神经元代表一个潜在动作的发生概率(所以不支持连续动作空间).每一轮的神经网络架构不变,但参数θ可能不同.但在一轮之内θ也不变,即在一轮内所有时刻的状态向量都输入至参数且架构相同的神经网络.等一轮结束后,θ再根据最后的收益使用梯度下降法更新,在更新函数中,有状态价值函数 $V(s)$;
- DDPG:基于策略(actor)和值(critic)的算法.在选择动作时,DDPG 使用和 Policy Gradient 一样的方式直接学习策略,以得到下一步要执行的动作;但在更新策略时,相比于 Policy Gradient 直接用带有状态价值函数 $V(s)$的梯度下降函数更新,DDPG 使用 $Q(s,a)$.与上述两个算法相比,其原始算法就支持状态和动作空间都连续的情况.如图 12(b)所示,分别有 Actor 和 Critic 两个神经网络,Actor 负责给定状态输出最佳动作(此时的网络输出层是动作的向量表示,而非每个动作的执行概率),Critic 负责给当前的状态与动作打分 $Q(s,a)$,打的分数再用来更新 Actor 网络参数,而 Critic 网络参数的更新同 Q-Learning 方法,根据环境给的奖励和下一状态的最大 Q 值更新.

此外,如图 11(b)所示,强化学习架构中的环境可以使用虚拟环境(回归模型和概率分布模型)替代而不需要真实环境,以减少实际交互开销.具体而言,给定当前状态和动作下,该回归模型和概率分布模型分别要学习奖励和下一个状态的状态分布概率函数以找到最有可能的下一状态.



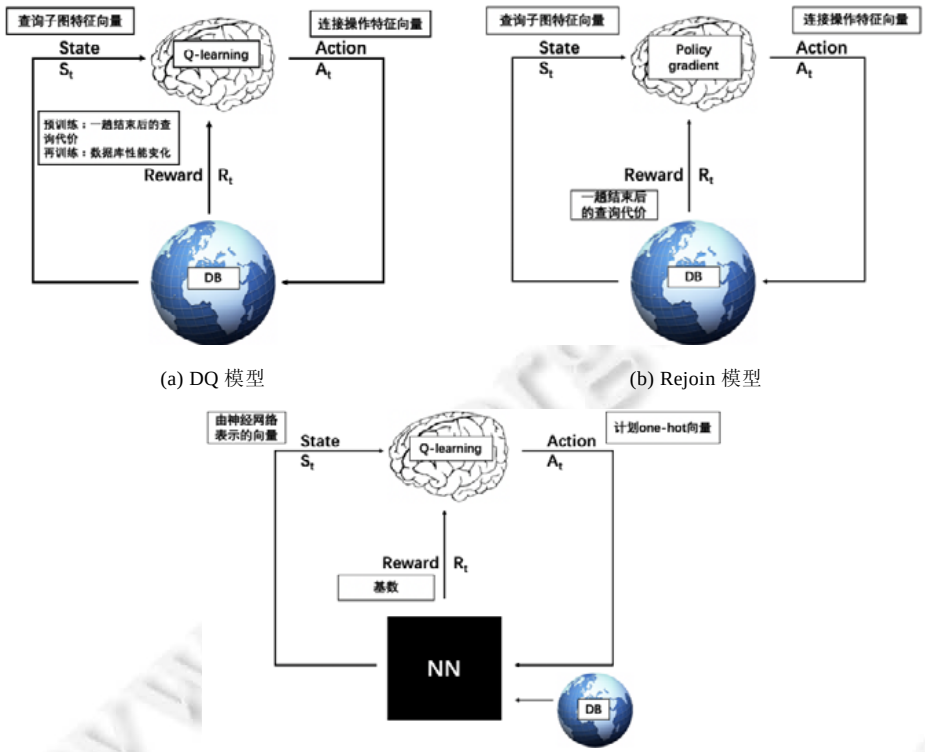
(a) Policy gradient 架构 (b) DDPG 架构

Fig.12 Decision algorithms in reinforcement learning

图 12 强化学习的决策算法

2.2.2 强化学习查询优化模型的实现与优化

如上节所述,查询优化过程可以抽象为马尔可夫决策过程,并使用强化学习进行优化.因此,强化学习查询优化模型的实现最重要的是对模型中各个部件的设计,具体包括状态 S、动作 A、环境 E、奖励 R、策略函数 P 以及使用的决策算法 F.由于策略函数根据决策算法更新,所以只要确定决策算法即可确定策略函数.图 13 展示了强化学习查询优化模型的 3 种典型方法.



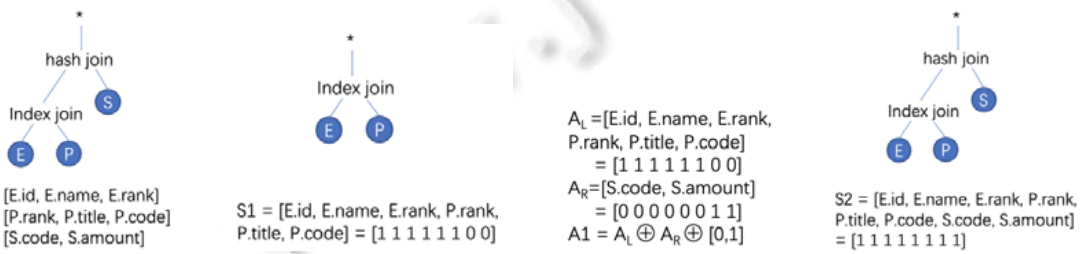
(a) DQ 模型 (b) Rejoin 模型

(c) LSR 模型

Fig.13 Reinforcement learning based query optimization methods

图 13 强化学习查询优化方法

DQ 模型[36]和 Rejoin 模型[37]的建模方式比较类似,状态和动作分别用查询子图特征向量和连接操作特征向量表示.以 DQ 为例,查询子图特征向量是查询子图涉及的属性的 one-hot 编码;连接操作特征向量是查询子树左树使用属性的 one-hot 编码、查询子树右树使用属性的 one-hot 编码和连接操作的物理操作 one-hot 编码之和.如图 14(a)所示,对表 E,P,S 做连接操作,假设当前状态为 $E \triangleright\triangleleft P$,如图 14(b)所示,则状态表示为[1,1,1,1,1,1,0,0].通过策略函数动作为$(E \triangleright\triangleleft P) \triangleright\triangleleft S$,如图 14(c)所示,则动作表示为[1,1,1,1,1,1,0,0]⊕[0,0,0,0,0,0,1,1]⊕[0,1].执行动作 1 后,数据库(环境)将其性能的变化传给决策算法以更新策略函数,同时查询子树(状态)也发生了变化,如图 14(d)所示,从状态 1 变成了状态 2.



(a) 查询树案例 (b) 状态 1 (c) 动作 1 (d) 状态 2

Fig.14 Implementation of reinforcement learning based query optimization

图 14 强化学习查询优化模型具体实现

DQ 和 Rejoin 的不同之处在于它们使用的决策算法不同,分别是 Q-Learning[33]和 Policy Gradient[34].最后,在奖励设计上,模型训练时,DQ 和 Rejoin 都考虑了一趟结束后查询计划的查询代价,如执行时间;模型用于实际场景时,Rejoin 仍使用一趟结束后的查询代价,而 DQ 使用数据库性能变化作为奖励,如吞吐量、延迟等.

LSR[38]不同于以上两种强化学习设计,使用了基于模型的强化学习架构,环境不再是真实的数据库,而是神经网络.不论是下一个状态还是奖励,都由神经网络学习得到.这里的奖励具体设为基数,所以神经网络可以看成是一个学习式代价估计模型.由于它使用 Q-Learning[33]作为决策算法,所以每次会选择 Q 值最大的而非基数最大的,这就与基于基数的贪婪法计划枚举有所区别.但是 LSR 的动作是用 one-hot 编码的,选当前要执行的动作为 1、已执行的动作为–1、其余动作为 0,所以动作空间大小非常有限,且动作向量非常稀疏.

强化学习查询优化模型需要不断地试错,而在学习初期容易选择一些比较差的查询计划.这些查询计划都需要在数据库中进行真实查询,耗时可想而知.因此,Marcus 等人[39]就尝试使用先验知识,如现有的成本模型,以快速度过这尴尬的学习初期.虽然现有的成本模型使用启发式规则进行预测,其准确性并无法保证,但是它携带了直方图等先验知识,所以让强化学习查询优化模型在初期先拟合现有的成本模型,这会比初期随机选择模型更加智能.

2.3 总 结

本节重点介绍了学习式查询优化模型,见表 2.学习式代价估计模型根据 SQL 输出查询代价,包括基数和选择率,该模型在性能上优于传统启发式估计模型,特别是当 SQL 语句有用户自定义函数或复杂表达式时.但是它们本质是监督学习中的回归模型,即需要有标注信息并且依赖于大规模高质量训练样本.强化学习查询优化模型旨在通过不断地试错,端到端地找到最优查询计划.其重点在于状态、动作、环境、奖励、策略函数以及决策算法的具体设计.从实验结果来看,强化学习运用在查询优化模型上的确可以使模型较快且准地找到最优查询计划.学习式查询优化还有以下挑战性问题亟待解决:首先,强化学习的 3 种决策算法 Q-Learning, Policy Gradient 和 DDPG 应用在查询优化问题上,尚没有一个系统性的实验对比与性能研究;其次,虽然强化学习查询优化模型可以优化没有任何先验知识的查询,但是如果有先验知识,这些知识也可以加以运用使模型更快地找到最优的查询计划.然而,如何结合先验知识是个不小的挑战.

Table 2 Summary of learnable query optimization model

表 2 学习式查询优化模型总结

模型	输入	输出	机器学习模型	核心
传统代价估计模型	SQL	查询代价(基数、选择率)	回归模型	回归函数、神经网络、KDE 拟合
查询优化的强化学习模型	SQL	查询计划	强化学习	强化学习建模

3 学习式数据库系统调优

针对具体的应用场景,对数据库系统进行调优(tuning)是非常重要且实际的问题.目前主流方法仍是由经验丰富的 DBA 进行人工调整,这带来了很高的人力成本.传统的数据库系统会提供一些辅助 DBA 进行调优的工具,例如 IBM 提出了 DB2 配置向导[40],通过向 DBA 提问的方式收集应用场景的反馈,并基于规则进行调优.微软的 SQL Server[41]和 MySQL 系统[42]也有类似的工具.但是随着数据库系统变得越来越复杂,这些方式已无法保证配置的有效性.本节介绍学习式数据库系统调优,该方法通过机器学习根据负载和数据库当前状态对系统进行自动配置.本节从 3 个方面进行介绍:第 3.1 节介绍学习式旋钮配置(knob configuration);第 3.2 节介绍如何利用机器学习优化数据的布局(按行或是按列存储或更复杂的混合存储);最后,第 3.3 节介绍学习式的方法选择分布式数据库的并发控制策略.

3.1 学习式旋钮配置

旋钮配置是系统调优的核心部分,它通过调整指定旋钮,使当前数据库性能最佳.这里的旋钮包括:数据库或系统配置参数,如缓冲池大小;物质资源分配的旋钮,如 CPU 的使用占比;用于负载准入控制的旋钮,如多编程

级别[43].对它们的配置极大影响着数据库的性能和可扩展性,而且这种影响因硬件平台、工作负载和数据属性而异.此外,不同旋钮对数据库影响不同且互相并非独立.鉴于旋钮配置问题的复杂性,许多组织都会聘请 DBA 或专家来调整旋钮.但随着数据库和应用程序在规模上的扩大和复杂性方面的提高,准确调整旋钮已经超越了人类的能力[44].BestConfig[45]尝试基于某些给定原理搜索最佳旋钮,但是搜索需要花费大量时间.

学习式旋钮配置方法根据负载在线调优,找到最佳旋钮配置,解决了传统方法无法根据负载实时调优的缺点.现有的方法分为两类:第 1 类基于监督学习,将旋钮配置问题建模为变化曲面拟合的问题;第 2 类基于强化学习,通过不断地试错,找出最优的旋钮配置.

(1) 基于监督学习的方法

旋钮调优的核心挑战在于拟合出数据库系统性能在不同旋钮调整时的变化曲面,再根据曲面选择出最优的配置方案.图 15 是一个变化曲面的案例:给定查询为 TPC-H Q18[46],假设仅考虑两个旋钮——寄存器大小和缓冲区大小.在它们取不同数值的时候,数据库系统执行查询会得到不同的性能,如:当寄存器大小和缓冲区大小分别设置为 250MB 和 140MB 的时候,平均执行时间最小,即系统取得了最佳的性能.然而,上述变化曲面的构造是十分昂贵的,因为旋钮可能配置的搜索空间很大,而且每尝试一种配置都需要实际执行一次查询.针对这一难题,一些方法[3,43]提出通过监督学习的方法,首先对部分配置进行采样,进而基于样本点拟合出变化曲面.iTuned[43]是第一个使用变化曲面来调整数据库配置参数的实用工具.对于查询 Q 和 d 个旋钮 $x_1,x_2,\ldots,x_d$,iTuned 首先采样 n 组旋钮配置,然后利用高斯过程拟合(GRS)拟合变化曲面.初始变化曲面会有一个性能最佳点,即此处估计的性能最佳.以此点作为新采样点,即在该点所在旋钮配置下重新执行 Q.采样后更新变化曲面,再找到新最佳点,采样并更新变化曲面.通过不断循环采样最佳点和更新变化曲面,变化曲面会拟合得和实际情况越来越相近.而且每次仅对最佳点采样,使得模型可以尽快找到最佳旋钮配置.上述循环直至性能到达满意值后结束.上述方法通过在不同旋钮配置下反复执行同一个负载,对旋钮进行在线调整,但它并没有从其他负载的变化曲面中学到知识.OtterTune[3]尝试从其他负载的变化曲面中学到知识.数据仓库存有很多旧负载的变化曲面,它们根据曾经在数据库运行过的负载基于采样得到.每当来了一个新负载,新负载利用执行情况映射到最相似的旧负载.将加噪声的旧负载变化曲面作为初始变化曲面,然后再通过与 iTuned 一样的采样和更新变化曲面方法最终找到最佳旋钮配置.以其他负载的变化曲面作为初始变化曲面,减少开始时大量采样旋钮操作,有效提升了旋钮调整的速度.

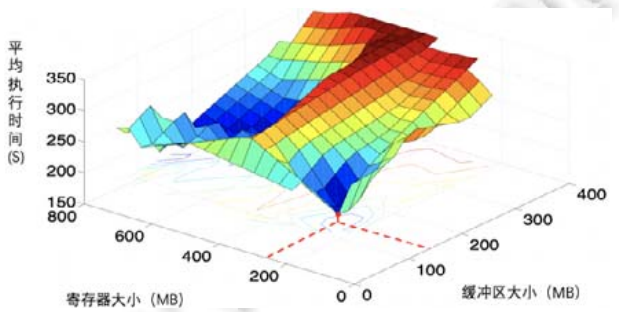


Fig.15 Trends of database performance with the changes of knob configuration[47]

图 15 数据库性能随旋钮配置的变化曲面[47]

(2) 基于强化学习的方法

上述基于曲面拟合的方法存在局限性:不仅需要采集大规模高质量训练样本,而且需要高维高斯过程拟合.就算拟合出精确的变化曲面,在曲面上找到最优解实际上还是 NP 难问题[3].所以有学者开始尝试使用强化学习对旋钮进行配置[47],不需要先验知识,也无需复杂的采样或变化曲面拟合.与查询优化类似,旋钮配置也可抽象为马尔可夫决策过程.在执行负载的过程中,数据库的状态发生着变化,在每一个时刻策略算法根据当前数据库状态(状态)可以选择要调整的旋钮并调整至某个值(动作);调整后,数据库状态变为新状态,再根据新状态选择

新动作.如此反复,直至负载执行结束.使用强化学习反复上述过程多轮,即可找到最适合该负载的数据库旋钮配置.如图 16 所示,CDBTune 模型[47]是典型的基于强化学习的学习式旋钮配置模型.状态使用向量化的 63 个指标表示数据库当前的状态,如页大小、上锁解锁时间等;决策算法使用 DDPG 算法[35]选择当前要调整的旋钮,即可满足高维度的状态和动作;动作即为 one-hot 编码的旋钮向量;奖励选择调整旋钮后数据库发生的性能变化,具体表现为数据库的吞吐量和延迟变化.例如对例 16 找到最佳旋钮配置,由于该案例重在缩短平均执行时间,所以奖励设为平均时间的倒数.

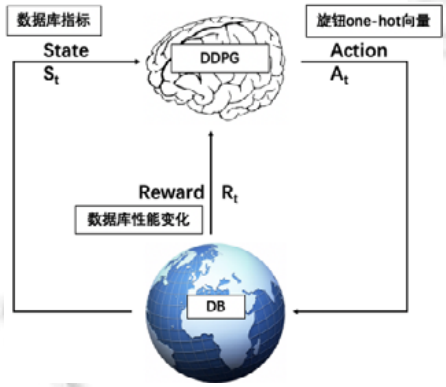


Fig.16 CDBTune: RL-based knob tuning

图 16 基于强化学习的调优 CDBTune

为方便举例,状态仅选择上锁解锁时间(二维向量);旋钮即选择寄存器和缓冲区的大小,假设寄存器大小仅可从 0 到 800MB 以 50MB 为间隔选(即 0,50,...,750,800),缓冲区大小仅可从 0 到 400MB 以 20MB 为间隔选,则旋钮向量则可表示为 30 维向量,每一维表示某旋钮的一个取值.如:当寄存器大小和缓冲区大小分别取 50,20 时,则该旋钮向量第 2 维和第 11 维取 1、其余取 0.则 CDBTune 强化学习过程如下.

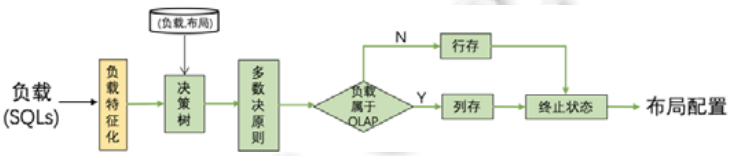
- 第 1 轮:在旋钮未调整过之前,上锁、解锁时间分别为 10,100(状态),策略函数根据当前状态选择寄存器大小和缓冲区大小分别 50,20(动作),重新执行负载,上锁、解锁时间变为 10,90(新状态)且平均执行时间降为 250,数据库反馈奖励 1/250 给 DDPG,以此来更新策略函数;
- 第 2 轮:策略函数根据状态(10,90)选择寄存器大小和缓冲区大小分别 100,40,再重新执行负载,新状态变为(10,75)、平均时间降低为 180,数据库反馈奖励 1/180 给 DDPG,以此来更新策略函数;
- 第无穷轮:最终模型收敛,当寄存器大小和缓冲区大小分别 250,140 时,平均执行时间最短.

3.2 学习式布局配置模型

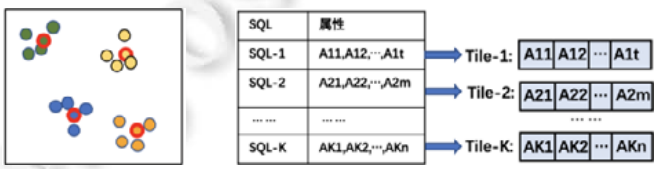
数据布局是指关系表中的数据以何种组织方式存储在磁盘中,常见的有按行存储(行存)、按列存储(列存)与混合存储.不同的布局有不同的优缺点,如行存写操作效率高但读操作效率低、而列存正好相反.所以布局的选择非常依赖于负载的特性,如 OLTP 负载访问的数据适合行存、OLAP 负载访问的数据适合列存.若选择不合适负载的布局,会影响数据库的读写效率.传统的布局配置由 DBA 根据一定规则事先选定,而没有根据真实负载动态选择.而布局作为旋钮之一,也可以使用之前介绍的学习式旋钮配置模型进行配置.但是,如果 DBA 已经总结出一些基于负载的布局配置规则,可以利用这些规则设计出更加专用的学习布局配置模型.

学习式布局配置模型利用负载的特性和启发式规则选择合适的布局.Elnaffar 等人[48]提出了基于决策树的策略,如图 17(a)所示.该方法将问题转换为特征工程和分类问题,对负载进行特征化,利用决策树判断负载属于 OLAP 还是 OLTP:若属于 OLTP,则采用行存布局;否则,使用列存布局.决策树利用曾经在数据库运行过的负载以及对应最佳布局方式构造.通过评测多种布局方式下执行同一个负载后的数据库性能,取性能最佳的布局方式作为该负载对应的最佳布局方式.随着负载的复杂度增加,单一的行存或列存已经无法满足复杂的负载需求.因

此,Pavlo 等人[11,49]提出了支持混合存储的学习布局配置模型,如图 17(c)所示.底层由“tile”组织以满足行列混存.当有新负载要执行时,负载内所有 SQL 根据属性重合度进行 *K*-means 聚类.如图 17(b)中标红的 SQL,每一类中都有一个 SQL 作为该类的 SQL 代表,再从这些 SQL 代表中选择对 IO 影响最大的 Top-*K* 个 SQL,最后根据 Top-*K* 个 SQL 访问的属性进行布局.如图 17(c)为布局配置案例,最后筛选出得 Top-*K* 个 SQL 分别是 SQL-1,SQL-2,..., SQL-*K*,其中,SQL-1 需要访问 *A*11,*A*12,...,*A*1*t* 这 *t* 个属性,它们可以来自不同的表,由于这些属性常常被一块访问,所以把它们各个列的属性值都存在一个 tile 内,即 Tile-1.同理,SQL-2 需要访问的属性列也存在一个 Tile-2 内.同一个属性列可能会以副本的形式存在多个 tile 内.负载的变化可能导致数据布局的重组,现有的方法是增量性重组[48],即以数据项为单位逐渐将数据迁移至新布局,要求不可迁移热数据.



(a) 基于决策树的布局配置模型



(b) SQL 聚类 (c) 布局配置案例

Fig.17 Learnable data layout configuration models

图 17 学习式数据布局配置模型

3.3 学习式并发控制策略选择

事务管理器保证事务的正确执行,即满足 ACID 的特性.并发控制是事务管理器的最重要功能之一,因为不合理的并发控制会导致存取和存储不正确数据、破坏事务的隔离性与数据库的一致性.然而,处理并发控制开销很大,频繁的并发控制已被证明大约是传统数据库中 OLTP 工作负载的 CPU 开销的 30%[50].因此,一些研究工作侧重从并发控制策略选择的角度开发学习式事务管理器.

这些工作的基本想法是:学习出查询负载的模式,从而对现有的并发控制策略进行最优的选择.现有的并发控制策略有很多,代表性的有 H-Store 使用的有基于分区的并发控制(PartCC)[51]、基于 Silo 的乐观并发控制(OCC)[52]和基于 VLL 的无等待两阶段锁定方法(2PL)[53,54]等.

- PartCC 作为并行数据库管理系统 H-Store 使用的并发控制策略,特别处理多个事务在多个站点同时执行的情况.它使用时间同步协议保证集群内所有机器的时间都一致,使用时间戳模型进行并发控制.为保证事务的执行是可串行化隔离的,它事先分析好事务使用数据的情况,如果两个事务的两个进程可能存在冲突,则不让它们在同一个时间执行;
- OCC 是乐观的并发控制协议,即:在事务运行时不申请对资源上锁,直接使用资源.在事务提交之前,每一个事务都会验证其他的事务是否修改了它所读取的数据.如果检测存在冲突修改,提交事务将被回滚并重新开始执行;
- 2PL 指每个事务的执行都分为加锁阶段和解锁阶段,读写数据前必须对数据加锁而非一次把所有数据都加锁、加锁请求都先于解锁请求.那么对这些事务的并发调度都是可串行化的.

经过实验发现:所有并发控制协议都有其最佳性能的事务场景,没有一个协议对所有场景都是理想的,并且错误的协议可能会导致显着的性能下降.如:多个事务在多个站点执行时,PartCC 是最佳选择;当冲突率很低时,OCC 工作得最好;而当冲突率很高时,2PL 最为理想.学习式事务管理策略选择模型就是要以机器学习的方式挖

掘出事务本身特性,以选择最佳的并发控制策略.如图 18 所示,学习并发控制选择策略根据当前事务从有限个并发控制策略中选择出最合适的一个.Tang 等人[55]对事务进行特征化,再利用决策树选择合适的并发控制策略.事务总共有 4 种特征,分别是读操作比率(ReadRatio)、事务访问的平均记录数(TransLen)、读取或写入相同记录的并发事务的概率(RecContention)和所有跨群集事务的成本(CrossCost).前三者记录了跨事务的冲突,这是影响 PartCC,OCC 和 2PL 性能的关键因素,第 4 个特征影响了 PartCC 的适用性.决策树使用 Zipf 分布事务和对应最高吞吐量的并发控制策略来训练.Zipf 分布事务使用综合事务,即包括各种场景,可以有效地探索特征空间,使训练好的模型可靠地预测最佳并发控制策略.

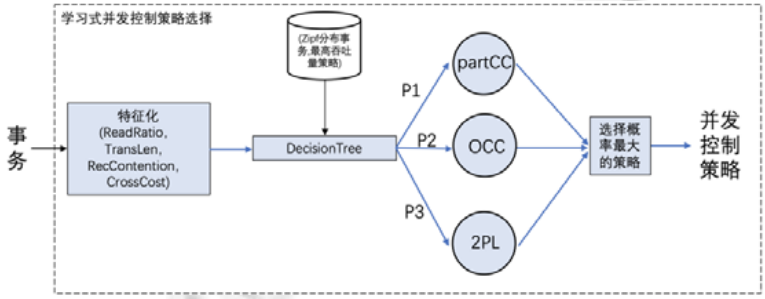


Fig.18 Learnable selection model for concurrency control strategies

图 18 学习式并发控制策略选择

3.4 总 结

本节主要阐述了学习式系统调优(见表 3),包括学习式旋钮配置模型、学习式布局配置模型和学习式并发控制策略选择.旋钮配置是找到一系列旋钮使得数据库性能最佳,学习式旋钮配置模型细分为基于监督学习和基于强化学习两种方法:前者基于采样和 GRS 拟合生成变化曲面;而后者通过不断尝试不同的旋钮组合,利用数据库性能作为奖励以引导旋钮重组,找到最优配置.布局配置探讨如何基于负载对数据的组织方式进行优化,现有工作的重点在于负载的分类,进而选择更好的布局方式.本节还介绍了学习式并发策略选择模型,根据事务的特性来选择合适的策略,以结合多种并发控制策略的优势.尽管现有的研究工作已经取得了不小的进展,但学习式数据库调优还有以下挑战性问题亟待解决:首先,现有的学习式调优方法多从查询负载中学习出数据模式,当切换应用场景时,由于积累的数据有限,这些方法难以在短期取得很好的效果;其次,我们发现机器学习算法在事务管理上的应用非常少,这方面的应用还有待探究;同时,还需进一步探究如何与 DBA 的经验相结合.

Table 3 Summary of learnable konb configuration model

表 3 学习式数据库系统调优模型总结

模型	输入	输出	机器学习模型	核心
学习式旋钮配置	负载	旋钮配置	回归模型	变化曲面拟合、强化学习
学习式布局配置模型	负载	布局配置	分类模型	特征分类
学习式并发控制策略选择	负载	并发控制策略	分类模型	特征分类

4 总结与展望

系统设计的通用性和面向应用的定制性,是数据库系统领域长期以来一直探讨,乃至争论的一对矛盾.

- 一方面,单纯追求系统的通用性,难免牺牲服务具体应用时系统的性能.目前,数据库领域已对“不是所有情况都适用于一种模式”(one size does not fit all)这一结论形成了广泛的共识;
- 另一方面,针对具体应用对数据库系统进行定制需要数据库管理员与开发人员的深度参与,费时费力不谈,在应用需求如此复杂多变的今天,也显得越来越捉襟见肘.

学习式数据库系统为解决这对矛盾提供了一种全新的思路:基于模型的系统定制.具体来讲,数据库系统充

分使用机器学习技术基于数据建模的能力,从负载与数据中学习出面向具体应用的特征或模式,从而为不同的应用需求定制出不同的模型.更为重要的是:通过“学习式”数据库核心组件的研发,使数据库系统具备利用模型做决策的能力,保证一定的通用性.

但由于数据库的数据与传统机器学习使用的数据有所差别,机器学习在数据库上的应用并不简单.传统机器学习多用于多维数据并基于独立同分布假设,而关系数据有更复杂的语义.首先,在数据类型上,传统机器学习处理数据的类型相对单一,而数据库中同一条记录内可能同时包含数值型、范畴型、文本型等多种数据类型的属性;其次,在不同维度的关联性上,传统机器学习假设同一条数据的不同维度之间不存在关联性(或通过主成分分解 PCA 等手段处理维度之间的关联性),而数据库中不同属性可能存在更为复杂的语义关联,如函数依赖或因果关系;最后,在数据间的关联性上,传统机器学习的数据基于独立同分布假设,而数据库的记录分布更复杂,并不一定满足独立同分布.上述差异为机器学习技术的应用带来新的难题:首先,数据类型的差异导致数据库数据作为机器学习模型输入时需要进行数据的表示学习,关系数据的表示学习相比单纯的文本或图片更为困难;此外,数据库数据存在记录间和属性间的依赖性导致训练出的模型通用性较低,这种过拟合现象的优点是适合当前的应用场景,但缺点是不利于数据库的后期维护,难以设计出同时兼顾通用性和专用性的机器学习模型;最后,若数据频繁更新,如何实现模型的实时更新,权衡机器学习应用的效果与成本,也是机器学习应用的一大难点.

本文以数据库系统架构的角度系统地综述了学习式数据库的研究工作,探讨了如何使用机器学习技术对数据库系统的核心组件进行优化.重点介绍了学习式数据库索引、学习式查询优化器和学习式系统调优这 3 个方面.学习式数据库索引的核心是利用监督学习对数据的分布进行拟合,形成有效的键值-数据映射函数,并由此提升数据查找的效率.学习式查询优化器分为基于监督学习的代价估计模型与基于强化学习的新型查询优化架构,其主要的发展方向是通过从过去的经验中学习,指导查询优化器做出更好的决策.学习式系统调优的基本思路是从历史的查询负载中学习模式或是对不同配置下的系统性能进行采样,从而对系统的配置进行调节与优化.需要强调的是:学习式数据库系统并不完全摒弃传统数据库组件,这些组件可以作为模型的一个构件以备机器学习无效时使用,或作为初始模型加快训练.

对不同类型的数据库,机器学习在它们的数据库组件上应用范围不同.表 4 中,数据库主要分为事务型数据库(TP)和分析型数据库(AP),它们在业务范围、主要数据库操作、操作对象、是否必须满足 ACID 特性、优化重点和适合配置等不同维度上都有不同[56].这些差异导致机器学习在它们的数据库组件上应用范围不同.

Table 4 Difference of TP and AP

表 4 TP 和 AP 的差异

维度	TP	AP
业务范围	事务操作、数据更改	数据查找和数据分析
主要数据库操作	增删改操作	复杂查询
操作对象	单表操作	多表操作
是否必须满足 ACID 特性	是	否
优化重点	写优化	读优化
适合配置	B-tree 索引、行存	位图索引、视图、星模式、列存

下面围绕本文介绍的几类学习式数据库组件做进一步的分析.

(1) 索引上,基于机器学习构造的索引会同时提高 TP 和 AP 数据存取的速度,但是由于 TP 的主要数据库操作为增删改操作,索引需要高频率更新,因此机器学习再训练成本过高,这给机器学习的应用带来了性价比的问题;

(2) 查询优化器上,机器学习可以找到最佳查询计划,利用使用本文介绍的强化学习查询优化模型进行连接顺序的优化.由于 AP 大多涉及多表的复杂查询而 TP 查询相对简单,通常涉及单表,所以此时机器学习会让 AP 有更多收益;

(3) 布局配置上,学习式布局配置模型的基本思路是:使用机器学习来预测未来的查询负载偏重 TP 还是

AP,以便提前对数据的存储形式做出调整.具体是:若查询负载偏 TP,则调整为行存;若偏 AP,则调整为列存,以满足两种类型的负载在写优化及读优化方面的不同侧重.旋钮设置上,机器学习让 TP 和 AP 都受益.最后,相比于 AP,TP 因会触发更多的事务冲突而更需要机器学习为其选择合适的并发控制策略,因此在并发控制方面,机器学习会更让 TP 受益.

对数据库而言,应用机器学习技术的性价比需要综合考虑数据库吞吐量和延迟方面的性能提升,以及机器学习模型在训练和执行上的时间成本.但是对不同类的数据库,其指标有不同的侧重.对 TP 而言,由于查询相对简单且有较多写操作,对事务的并发控制提出了严峻的挑战,所以机器学习技术的性能主要通过自动化选择并发控制策略后,并发事务处理能力的提升来衡量.而频繁的写操作涉及较多的模型更新,所以机器学习技术的成本主要由再训练的时间衡量.而 AP 的主要业务是复杂查询,所以查询速度是机器学习性能的主要衡量指标,而在寻找最佳查询计划时,越复杂的模型也许能找到越好的查询计划但是执行时间较长,如 LSTM 模型比泊松回归模型更好地解决复杂基数估计问题,但模型进行预测的开销也更大.所以需要用模型执行时间来衡量机器学习成本.

根据应用机器学习的性价比决定,并非所有场景都适合使用学习式数据库:首先,学习式数据库难以承担过多更新操作,因为它意味着频繁重新训练机器学习模型,训练代价过大;其次,若数据库数据过少或分布单一,传统数据库组件已经可以较好地处理该情况,而使用机器学习技术又增加了额外开销,还很容易过拟合而无法保证其通用性;最后,若数据本身并没有任何规律,机器学习技术难以挖掘出数据分布并构建合适的组件模型.

总的来说,机器学习为解决数据库问题提供了新思路,利用底层数据分布和事务特征设计出的专用数据结构有效地节省了时间和空间开销.此外,传统数据库系统仍需大量人为参与,如旋钮配置等,这增加了大量人工成本且无法保证其可靠性.但机器学习技术根据数据分布做出决策,降低人工成本的同时又保证了可靠性.最后,机器学习可以从历史数据以及数据库表现上吸取经验教训.在带来上述优势的同时,机器学习技术也为数据库带来了以下弊端.

- 第一,相比传统数据库组件,机器学习构建的组件难以适应业务迁移,可能存在冷启动问题.如学习式索引模型由于再训练成本过高,所以无法适应数据频繁更新的情况;
- 第二,当数据库数据分布、事务特征都非常简单时,使用机器学习性价比过低.如做查询优化时,若查询非常简单,查询计划全枚举也许快于使用强化学习查询优化模型;
- 第三,学习式数据库系统性考虑不足,并不考虑数据库系统中组件之间的牵连性,这导致学习式数据库系统的性能可能只能达到局部最优而非全局最优.如当内存大小给定时,缓冲区使用内存大小和索引使用内存大小是相互制约的,即:为缓冲区分配更多内存意味着为索引分配更少内存,那么在减少缓冲区中数据调用时间的同时又增加了从磁盘取数据的 IO,整体的性能未必改善.

最后,学习式数据库系统仍存在着很多开放式研究问题,这里归纳为 6 点.

- 模型构建:传统的机器学习技术一般假设数据独立且同分布,但是实际的数据库数据分布复杂,不一定满足上述假设.例如:若数据库键与键的特征并非相似,则学习式布隆过滤器会面临失效;其次,在基于数据驱动的新模型构造初期,所需要的数据通常不足或需要实时采样,这导致构建模型存在冷启动问题,如 iTuned 在构造初始变化曲面时,得先采样多组旋钮、执行多次负载;再次,某些模型的构建用了启发式规则,例如,基于决策树的布局配置模型使用了“OLTP 适合行存布局、OLAP 适合列存布局”,这些启发式规则可以简化模型,但是并非完全可信,如何处理它们使学习式组件更加准确尚无定论;最后,学习式组件和传统数据库组件究竟要以什么方式结合,若学习式组件与传统数据库组件的输出结果发生冲突时,应该更倾向于谁的结果.这些问题都是模型构建时应该考虑的问题;
- 系统整合:本文阐述了机器学习在数据库系统部分组件上的优化,但是目前尚没有对所有学习式组件进行整合的系统.由于组件之间的牵连性,一个组件性能的改善也许意味着另一个组件性能的恶化.例如:当内存大小给定时,给缓冲区分配更多内存,就意味着给索引分配了更少的内存,那么虽然减少了调用缓冲区中数据的时间,但是增加了从磁盘取数据的 IO,整体的性能未必改善.因此,如何系统地将各

个学习式组件进行整合,各司其职、物尽其用,是形成完整学习式数据库亟待解决的一步;

- 数据更新:基于数据驱动的学习式数据库只适合于训练模型时使用的数据或负载,但是数据和负载随着时间可能会发生变化.例如:基于现有数据构建的RMI模型(第2节)会随着更新而难以对数据进行有效的拟合;基于历史查询预测的负载趋势也可能随着查询的变化而变得不再适用.因此,现有的方法采用阶段性更新学习模型,如每月更新一次.然而,若更新过于频繁,会带来很大的开销;反之,学习式数据库可能不再适用.此外,数据与负载的变化并非规律性的,所以很难找到模型更新的适宜频率.所以更新的时间、频率与方式都需要进一步的研究;
- 评测基准:对任何一个学习式组件都可以由不同的模型去替代或优化,但是通过调研,本文发现:对大部分学习式组件而言,现如今并没有一个评测基准于评测这些模型.这样就无法对比不同的模型性能,也不知道模型具体适合什么应用场景.传统数据库的评测基准有TPC系列;对个别组件有一些评测基准,如查询优化使用的JOB[57].然而,系统性地探讨如何对学习式数据库系统进行评测,并构建评测基准的工作目前尚属空白.这不利于对不同的模型进行效果评测,从而进行对比;
- 未来预测:相比传统数据库组件只能实现基于当前数据与负载的基本功能,学习式数据库也许还可以使用机器学习的预测能力对未来的数据和负载进行预测,从而成为一个智能型学习式数据库.一旦有了这种预测能力,学习式组件可以自动更新而无需采用当下的阶段性更新法.此外,还可以根据未来数据及负载对资源及时地合理分配,如对内存、缓冲区分配执行负载所需的内存空间,对数据合理地分配到不同站点从而减少并发处理的冲突率;
- 在线机器学习(OLML)[58]:OLML是以在线构建模型为主的应用.区别于传统TP和AP类应用,即以业务为中心、便于提高工作人员的业务效率的应用,OLML以用户为中心,用户可以直接在数据平台使用而不受过多技术限制.OLML需求多样,呈现出不同的负载和数据模型,底层需要新型数据库系统来支撑,学习式数据库系统是一种可行的解决方案.

References:

[1] Wang W, Zhang MH, Chen G, *et al*. Database meets deep learning: Challenges and opportunities. SIGMOD Record, 2016,45(2): 17−22.

[2] Ré C, Agrawal D, Balazinska M, *et al*. Machine learning and databases: The sound of things to come or a cacophony of hype? In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2015. 283−284.

[3] Van Aken D, Pavlo A, Gordon GJ, *et al*. Automatic database management system tuning through large-scale machine learning. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. ACM, 2017. 1009−1024.

[4] Joshi A, Pavlo A, Butrovich M, *et al*. External vs. Internal: An essay on machine learning agents for autonomous database management systems. IEEE Data Engineering Bulletin, 2019,42(2):31−45.

[5] Oracle database contributors. Oracle database 18c. https://www.oracle.com/technetwork/cn/database/index.html

[6] Li SH, Li GL, Zhou XH. Xuanyuan: An AI-native database. IEEE Data Engineering Bulletin, 2019,42(2):70−81.

[7] Kraska T, Alizadeh M, Beutel A, *et al*. Sagedb: A learned database system. In: Proc. of the 9th Biennial Conf. on Innovative Data Systems Research (CIDR 2019). 2019.

[8] Pavlo A, Jones EPC, Zdonik SB. On predictive modeling for optimizing transaction execution in parallel OLTP systems. Proc. of the VLDB Endowment, 2011,5(2):85−96.

[9] Krishnamurthy R, Boral H, Zaniolo C. Optimization of nonrecursive queries. In: Proc. of the 12th Int'l Conf. on Very Large Data Bases (VLDB'86). 1986. 128−137.

[10] Kumar A, Boehm M, Yang J. Data management in machine learning: Challenges, techniques, and systems. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data (SIGMOD Conf. 2017). 2017. 1717−1722.

[11] Pavlo A, Angulo G, Arulraj J, *et al*. Self-driving database management systems. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research (CIDR 2017). 2017.

[12] Kraska T, Beutel A, Chi EH, *et al*. The case for learned index structures. In: Proc. of the SIGMOD Conf. ACM, 2018. 489−504.

[13] Ding JL, Minhas UF, Zhang HT, *et al*. ALEX: An updatable adaptive learned index. CoRR, abs/1905.08898, 2019.

[14] Li X, Li JD, Wang XL. ASLM: Adaptive single layer model for learned index. In: Proc. of the 24th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2019). 2019. 80−95.

[15] Tang CZ, Dong ZY, Wang MJ, *et al*. Learned indexes for dynamic workloads. CoRR, abs/1902.00655, 2019.

[16] Graves A. Generating sequences with recurrent neural networks. CoRR, abs/1308.0850, 2013.

[17] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proc. of the Advances in Neural Information Processing Systems 27: Annual Conf. on Neural Information Processing Systems 2014. 2014. 3104−3112.

[18] Mitzenmacher M. A model for learned bloom filters, and optimizing by sandwiching. CoRR, abs/1901.00902, 2019.

[19] Wu YJ, Yu J, Tian YY, Sidle R, Barber R. Designing succinct secondary indexing mechanism by exploiting column correlations. CoRR, abs/1903.11203, 2019.

[20] Ono K, Lohman GM. Measuring the complexity of join enumeration in query optimization. In: Proc. of the 16th Int'l Conf. on Very Large Data Bases. 1990. 314−325.

[21] Ioannidis YE. The history of histograms (abridged). In: Proc. of the 29th Int'l Conf. on Very Large Data Bases. Very Large Data Bases, 2003. 19−30.

[22] Flajolet P, Martin GN. Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences, 1985,31(2):182−209.

[23] Whang KY, Vander Zanden BT, Taylor HM. A linear-time probabilistic counting algorithm for database applications. ACM Trans. on Database System, 1990,15(2):208−229.

[24] Durand M, Flajolet P. Loglog counting of large cardinalities (extended abstract). In: Proc. of the 11th Annual European Symp. on Algorithms (ESA 2003). 2003. 605−617.

[25] Flajolet P, Fusy E, Gandouet O, *et al*. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In: Proc. of the Conf. on Analysis of Algorithms, 2007. 127−146.

[26] Giroire F. Order statistics and estimating cardinalities of massive data sets. Discrete Applied Mathematics, 2009,157(2):406−427.

[27] Heimel M, Kiefer M, Markl V. Self-tuning, GPU-accelerated kernel density models for multidimensional selectivity estimation. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. 2015. 1477−1492.

[28] Kiefer M, Heimel M, Breß S, Markl V. Estimating join selectivities using bandwidth-optimized kernel density models. Proc. of the VLDB Endowment, 2017,10(13):2085−2096.

[29] Wu CG, Jindal A, Amizadeh S, Patel H, Le WC, Qiao S, Rao S. Towards a learning optimizer for shared clouds. Proc. of the VLDB Endowment, 2018,12(3):210−222.

[30] Nazi A, Dutt A, Wang C, *et al*. Selectivity estimation for range predicates using lightweight models. 2019.

[31] Sun J, Li GL. An end-to-end learning-based cost estimator. CoRR, abs/1906.02560, 2019.

[32] Müller M, Moerkotte G, Kolb O. Improved selectivity estimation by combining knowledge from sampling and synopses. Proc. of the VLDB Endowment, 2018,11(9):1016−1028.

[33] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 1998.

[34] Sutton RS, McAllester DA, Singh SP, *et al*. Policy gradient methods for reinforcement learning with function approximation. In: Proc. of the Advances in Neural Information Processing Systems 12. MIT Press, 2000. 1057−1063.

[35] Lillicrap TP, Hunt JJ, Pritzel A, *et al*. Continuous control with deep reinforcement learning. In: Proc. of the 4th Int'l Conf. on Learning Representation (ICLR 2016). 2016.

[36] Krishnan S, Yang ZH, Goldberg K, *et al*. Learning to optimize join queries with deep reinforcement learning. CoRR, abs/1808.03196, 2018.

[37] Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration. In: Proc. of the 1st Int'l Workshop on Exploiting Artificial Intelligence Techniques for Data Management. 2018.

[38] Ortiz J, Balazinska M, Gehrke J, *et al*. Learning state representations for query optimization with deep reinforcement learning. In: Proc. of the 2nd Workshop on Data Management for End-to-end Machine Learning. 2018.

[39] Marcus R, Papaemmanouil O. Towards a hands-free query optimizer through deep learning. In: Proc. of the 9th Biennial Conf. on Innovative Data Systems Research (CIDR 2019). 2019.

[40] Kwan E, Lightstone S, Storm A, *et al*. Automatic configuration for IBM DB2 universal database. Technical Report, IBM, 2002.

[41] Narayanan D, Thereska E, Ailamaki A. Continuous resource monitoring for self-predicting DBMS. In: Proc. of the MASCOTS. 2005. 239−248.

[42] Mysql tuning primer script. https://launchpad.net/mysql-tuning-primer

[43] Duan SY, Thummala V, Babu S. Tuning database configuration parameters with ituned. Proc. of the VLDB Endowment, 2009,2(1): 1246−1257.

[44] Bernstein P, Brodie M, Ceri S, *et al*. The asilomar report on database research. ACM SIGMOD Record, 1998,27(4):74−80.

[45] Zhu YQ, Liu JX, Guo MY, *et al*. Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. In: Proc. of the 2017 Symp. on Cloud Computing (SoCC 2017). 2017. 338−350.

[46] TPCH Contributors. Transaction processing performance council for decision support benchmark. http://www.tpc.org/tpch/

[47] Zhou K, Zhang J, Liu Y, *et al*. An end-to-end automatic cloud database tuning system using deep reinforcement learning. 2019.

[48] Elnaffar S, Martin TP, Horman R. Automatically classifying database workloads. In: Proc. of the 2002 ACM CIKM Int'l Conf. on Information and Knowledge Management. 2002. 622−624.

[49] Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Proc. of the 2016 Int'l Conf. on Management of Data (SIGMOD 2016). 2016. 583−598.

[50] Harizopoulos S, Abadi DJ, Madden S, *et al*. OLTP through the looking glass, and what we found there. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2008). 2008. 981−992.

[51] Kallman R, Kimura H, Natkins J, *et al*. H-Store: A high-performance, distributed main memory transaction processing system. Proc. of the VLDB Endowment, 2008,1(2):1496−1499.

[52] Tu S, Zheng WT, Kohler E, *et al*. Speedy transactions in multicore in-memory databases. In: Proc. of the ACM SIGOPS 24th Symp. on Operating Systems Principles (SOSP 2013). 2013. 18−32.

[53] Yu XY, Bezerra G, Pavlo A, *et al*. Staring into the abyss: An evaluation of concurrency control with one thousand cores. Proc. of the VLDB Endowment, 2014,8(3):209−220.

[54] Ren K, Thomson A, Abadi DJ. Lightweight locking for main memory database systems. Proc. of the VLDB Endowment, 2012,6(2): 145−156.

[55] Tang DX, Jiang H, Elmore AJ. Adaptive concurrency control: Despite the looking glass, one concurrency control does not fit all. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research (CIDR 2017). 2017.

[56] Michael S, Cetintemel U. "One size fits all": An idea whose time has come and gone. In: Proc. of the 21st Int'l Conf. on Data Engineering (ICDE 2005). IEEE, 2005.

[57] Leis V, Gubichev A, Mirchev A, *et al*. How good are query optimizers, really? Proc. of the VLDB Endowment, 2015,9(3): 204−215.

[58] Du XY. OLML. http://zb.gywb.cn/276/index.html

柴茗珂(1996－),女,浙江衢州人,硕士,主要研究领域为数据库与大数据.

范举(1984－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为数据库与大数据,众包数据管理,数据准备.

杜小勇(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为数据库与大数据,智能信息检索,知识工程.