计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220931
61(4): 929−954, 2024

数据库索引调优技术综述

赖思超[1] 吴小莹[1] 彭煜玮[1] 彭智勇[1,2]
[1](武汉大学计算机学院 武汉 430072)
[2](武汉大学大数据研究院 武汉 430072)
(sichaolai@whu.edu.cn)

Survey on Database Index Tuning Techniques

Lai Sichao[1], Wu Xiaoying[1], Peng Yuwei[1], and Peng Zhiyong[1,2]
[1](*School of Computer Science*, *Wuhan University*, *Wuhan* 430072)
[2](*Big Data Institute*, *Wuhan University*, *Wuhan* 430072)

Abstract Index tuning is an important problem in database performance tuning and has been studied consistently by worldwide researchers. Due to the theoretical complexity of index tuning as well as the advent of the big data era, manual tuning by DBA is no longer feasible for modern database systems, hence automated and intelligent solutions have been developed. With the development of machine learning techniques, more and more index tuning solutions have integrated with machine learning techniques for better performance and significant progress has been made recently. In this survey, we formulate the problem of index tuning under a search-based paradigm, and under this context, we analyze the main tasks and challenges of this problem. We categorize relevant studies into three main components of the search-based paradigm, namely the generation of the index configurations' search space, the evaluation of specific index configurations, and the enumeration or the search of index configurations. Then we discuss and compare the related work in each category. We further identify and analyze new challenges for the online index tuning problem where the workload is ad hoc, dynamic, and shifting. We summarize the existing solutions under the online feedback control loop framework. Finally, we discuss the state-of-the-art index tuning tools. Hopefully, with the thorough discussion and evaluation of current research, this survey can provide insights to interested researchers and conclude with future research directions for index tuning.

Key words database index; index selection; index tuning; performance tuning; machine learning

摘 要 索引调优是数据库调优的重要组成部分,一直受到广泛关注.由于索引调优问题的理论复杂性和大数据时代的到来,通过DBA手动调优的方案已经无法满足现代数据库的发展需求,调优方案逐渐开始向自动化、智能化的方向发展.随着机器学习技术的发展,越来越多的索引选择方案开始引入机器学习技术,并取得了一定的研究成果.将索引调优问题的解决方案归结为一种基于搜索的调优范式,归纳了其研究内容,阐述了其面临的挑战,对调优范式内的索引配置空间的生成、索引配置的评价以及索引配置的枚举与搜索3方面的研究成果进行了归纳、总结和对比.对动态工作负载下的索引选择问题(index selection problem, ISP)所面临的新挑战进行了分析,并基于在线反馈控制回路框架对其解决方案进行梳理.讨论了索引调优工具的发展与现状,通过对现有研究的分析论述,为后来研究者提供参考和研究思路,并对索

收稿日期: 2022-11-11; **修回日期**: 2023-06-13
基金项目: 国家自然科学基金项目(U1811263); CCF-华为数据库创新研究计划项目(CCF-HuaweiDBIR003A)
This work was supported by the National Natural Science Foundation of China (U1811263) and the CCF-Huawei Database Innovation Research Program (CCF-HuaweiDBIR003A).
通信作者: 彭智勇(peng@whu.edu.cn)

引选择方案的未来进行了展望.

关键词 数据库索引;索引选择;索引调优;性能调优;机器学习

中图法分类号 TP18

近年来,随着企业的业务数据和业务场景越来越复杂多样,一个固定的数据库系统配置往往不能再满足多变的业务需求,数据库系统的调优也变得越来越重要.这个调优工作以往通常被交给企业的数据库管理员(database administrator, DBA)处理.而据统计,DBA 的薪水支出占企业数据库系统近 1/2 的所有权总成本(total cost of ownership),且 DBA 将近 1/4 的工作时间用于数据库的调优[1],因此高效的数据库调优对企业的业务系统性能和运营成本至关重要.其中,数据库物理设计表达了从逻辑数据库到物理数据库的物化过程,涵盖所有影响存储介质上物理结构的设计问题,比如表的规范化、索引、物化视图、无共享分区、多维集群等问题,是影响数据库查询性能的主要因素之一[2].作为数据库物理设计中重要的物理结构,数据库索引记录了数据属性(被称为索引的搜索键,由一个或多个索引键(index key)组成)到数据记录间的映射关系,能帮助数据库快速找到所需的数据记录.合理的数据库索引配置能极大提升数据库的查询处理效率,是数据库实现高效查询处理的关键[3],因此索引调优是数据库调优的重要组成部分.

索引虽然可能提升工作负载的查询效率,但作为一种物理结构,也会占用磁盘/内存存储空间,且需适时进行维护,比如在处理数据操作语句导致数据库数据发生更新时.过多的索引可能导致磁盘/内存资源的浪费和索引维护代价的提高,抵消其带来的查询效率提升效果.因此,索引调优的核心内容是:如何选择一个合适的索引集合,在其存储空间占用和维护代价可控的情况下,最大化提升工作负载的查询效率.在学术论文中,这个问题通常也被称为索引选择问题(index selection problem, ISP)[4].

ISP 是数据库领域中一个历久弥新的问题.在学术界,其研究历史可追溯到操作系统文件的组织和索引机制[4-7].研究者们[8-9]发现索引可能给数据查询带来效率提升,也认识到 ISP 的难度和挑战.在工业界实践中,这个工作早期主要由 DBA 参考数据库的调优手册,并结合自身经验,手动完成调优.但在面对中大型规模的索引选择问题时,这种方式变得不太现实,所以主流商业数据库厂商开发了相关调优工具来辅助 DBA 进行调优,比如微软的 SQL Server 数据库调优顾问(database tuning advisor, DTA)[10]、IBM 的 DB2 设计顾问(DB2 design advisor, DB2Advis)[11-12]以及 Oracle 的 SQL 访问顾问(SQL access advisor)[13]等.随着机器学习技术的发展和数据库应用场景的拓展,ISP 的相关研究也重新获得越来越多研究者的关注,比如引入机器学习技术的索引选择方案[14]、数据库集群/云数据库上的 ISP[15-16]等.

根据索引键所包含的属性数量是否为 1,索引可被分为单列索引和多列索引.而根据索引的特性,索引也可被分为主索引(primary index)和二级索引(secondary index)(也成为辅助索引).其中,主索引的索引键是唯一的,且数据记录根据该搜索键值的顺序进行组织,所以 1 个关系表上最多只能有 1 个主索引;二级索引记录各索引键值及其对应数据记录的位置,数据记录顺序与索引键值顺序无关,所以 1 个关系表上可以有多个二级索引.在一些探索性数据分析任务中,一些研究者提出了自适应索引(adaptive indexing)技术[17-19],即根据查询语句自动生成索引结构并进行自适应调整,避开选择索引的问题,比如数据库分裂(database cracking)[17]、索引自适应合并(adaptive merging)[18]技术.主索引通常在数据库系统部署前的逻辑结构设计阶段被确定;而自适应索引技术大多依赖快速的属性列操作,比如属性列数据的高效移动,故相关研究大多集中在列存数据库背景下,不够通用.因此本文将综述范围聚焦在通用的二级索引调优技术上.文献 [2] 的综述范围为数据库物理设计技术;而文献 [20] 则主要对到 2010 年为止微软的数据库物理优化工作进行总结;文献 [21] 虽然研究的是索引选择问题,但该文献仅对部分传统方法进行描述和实验的对比分析,没有对现有研究方法进行系统性总结;且不涉及应用机器学习技术的相关新进展.虽然现有 ISP 的研究大多集中在磁盘数据库领域,但其主要研究的仍是逻辑层面上选择相关属性组成索引并构建索引配置的问题.文献 [22] 也指出内存数据库和磁盘数据库虽在索引结构设计方面存在不同,但在查询优化方面是类似的,而文献 [23] 则是关于磁盘数据库的索引选择相关方法在内存数据库下的应用,故我们认为相关理论方法

也适用于内存数据库，并在后文不再作区分.

ISP 通常被默认为离线（offline）ISP，即工作负载是静态的. 作为一个组合搜索/优化问题，ISP 在理论上被证明是 NP 难的[8]，也很难被近似[9]. 这意味着找不到一个多项式时间复杂度的精确算法对该问题进行求解，所以现有研究大多通过设计不同的近似算法进行求解.

通过对 ISP 的特点和现有研究的调研和分析，本文首先对 ISP 进行形式化定义，并将现有研究工作的解决方案归结为一种基于搜索的求解范式（见第 1 节），并随后依次对该范式内三大部分（或者称为步骤）的内容和方法分别进行总结、分析和讨论（见第 2~4 节）. 随着大数据时代的到来，企业的业务场景越来越多变，动态工作负载下的 ISP（也被特称为在线（online）索引选择问题）所面临的新挑战变得越来越突出，本文基于在线反馈控制回路（online feedback control loop）[24] 框架对动态负载下的索引选择方案进行梳理，并进行讨论和对比（见第 5 节）. 然后，结合不同时代对索引调优的不同需求，对工业界的数据库索引调优工具的发展和现状进行介绍（见第 6 节）. 最后，对 ISP 在静态和动态工作负载下的解决方案进行总结，并对未来的发展方向进行展望（见第 7 节）. 通过对现有研究的归纳、总结和分析，为研究者提供参考和研究思路.

1 索引选择问题

1.1 问题定义

优化器的查询优化目标在于为查询语句生成合适的物理执行计划，包括确定每个逻辑操作符的物理实现方式（即物理操作符），比如关系表的访问路径（access path）、关系表连接操作的物理实现方式、关系表连接操作的顺序等. 现有数据库管理系统（database management system, DBMS）优化器大多采用基于代价的查询优化策略，即通过代价模型估计不同候选物理执行计划的代价，并输出代价最小的物理执行计划. 数据库索引配置影响查询语句的查询优化结果（即生成的查询计划）和查询执行代价. 比如，如果查询语句过滤谓词的相关属性上定义了索引，那么该查询可利用索引扫描操作快速返回关系表上符合过滤谓词的记录，避免遍历该关系表的所有记录.

ISP 研究的是在特定的约束下，找到能给代表性工作负载（representative workload）带来最大代价缩减的最优索引配置的问题. 其中，代表性工作负载可由 DBA 设计和给定，比如基准测试工作负载，也可通过辅助工具生成，比如 DB2 的查询巡逻器（patroller）[12]、SQL Server 的跟踪探查器（profiler）[25] 和 Oracle 的数据库性能自动监控器（automatic database performance monitoring, ADDM）[26]. 作为一个组合搜索类型的优化问题，ISP 的搜索空间是离散的索引配置空间，目标是找到最优的索引配置[27].

为了对这个问题进行形式化定义，首先引入一些基本概念. 一个大小为 d 的数据库实例 D 可以表示为关系表的集合 $\{T_1,T_2,\cdots,T_d\}$，一个包含 m 个属性的关系表 T_i 可以表示为 $T_i=\{c_1,c_2,\cdots,c_m\}$，其中 c_i 表示关系表中的一个属性. 数据库工作负载通常指的是由一系列查询语句组成的集合，一个大小为 n 的工作负载 W 可表示为 $W=\{q_1,q_2,\cdots,q_n\}$，其中 q_i 表示工作负载中的一个查询语句. 索引配置指由索引组成的集合，一个大小为 k 的索引配置 I 可表示为 $I=\{i_1,i_2,\cdots,i_k\}$，其中 i_j 表示索引集合中的一个索引. 约束集指的是索引选择过程中所需遵循的约束的集合，一个大小为 j 的约束集即可表示为 $B=\{b_1,b_2,\cdots,b_j\}$，其中 b_i 表示约束条件集中的一个约束. 最常见的约束条件包括索引存储空间约束（索引配置中索引所占用物理存储空间的限额）和调优时间约束（调优算法运行时间限额）. $cost(W,I)$ 表示工作负载 W 在数据库索引配置 I 下的代价，可通过实际执行时间、外部代价模型估计、优化器代价估计等方式获取. 综合现有研究中的不同定义，本文对 ISP 进行形式化定义.

定义 1. 给定一个数据库 $D=\{T_1,T_2,\cdots,T_d\}$、一个代表性工作负载 $W=\{q_1,q_2,\cdots,q_n\}$ 以及一个约束条件集 $B=\{b_1,b_2,\cdots,b_j\}$，找到一个最优索引配置 I^*，使能在满足约束条件集 B 的情况下，最小化工作负载 W 的代价，即 $I^*=\arg\min_I cost(W,I)$.

其中最小化工作负载代价相当于最大化一个索引配置带来的工作负载代价缩减（相对初始配置），该缩减值也可看成这个索引配置给工作负载带来的收益，即 $benefit(W,I)=cost(W,I)-cost(W,I_0)$，其中 I_0 表示初始的索引配置，因此 ISP 的优化目标也可被等价表示为 $I^*=\arg\max_I benefit(W,I)$.

1.2 研究内容与基于搜索的调优范式

从该形式化定义出发，我们将 ISP 的主要研究内容总结为 3 点：1）从问题的输入来说，数据库的模式信息确定了索引的搜索键来源，代表性工作负载表达了用户的查询需求，需要找到一种有效的方法来

生成包含能提高该工作负载预期处理效率的候选索引配置的搜索空间; 2)从问题的目标来说, 为了界定最优的索引配置, 需要找到一种合理的方法来评价索引配置的优劣、对索引配置进行比较; 3)从问题的核心来说, 需要找到一种合理的策略能在索引配置搜索空间高效地找到最优的索引配置.

这种按照一定策略在索引配置搜索空间中搜索最优配置的方式是自然的, 因此本文将现有解决方案都归结为一种基于搜索的调优范式, 包含索引配置搜索空间的生成、索引配置的评价以及索引配置的枚举 3 个部分(步骤), 如图 1 所示.

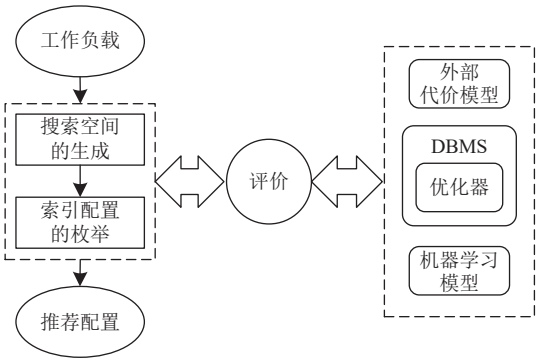


Fig. 1 Search-based index selection framework

图 1 基于搜索的索引选择框架

首先在该范式中, 搜索空间生成部分通过分析输入的代表性工作负载内容, 比如查询语句的特征和当前数据库状态, 生成候选索引及候选索引配置集合. 候选索引配置组成的搜索空间的大小影响着调优算法的代价, 更大的搜索空间往往意味着更高的调优算法代价上限. 其次, 为了评价候选索引配置的优劣, 需要对工作负载在各个候选索引配置下的执行情况进行评估和判断, 更好的索引配置能给工作负载的执行代价带来更大缩减. 图 1 中的评价模块展示了 3 种常见的索引配置评价方式, 即基于外部代价模型、基于优化器以及基于机器学习模型的方法. 现有大部分研究都采用基于优化器的评价方法, 这种方式利用优化器返回特定候选索引配置下的最优查询计划和代价估计(具体细节和方法对比参见本文第 3 节). 最后, 枚举和搜索策略确定搜索空间内的配置搜索路径, 一个好的枚举策略能让算法更快找到最优索引配置. 比如在图 2 中, 最优配置用实心圆表示, 在同样的搜索空间情况下, 右下角的搜索路径直到最后才找到最优配置, 而右上角的搜索路径能提前找到最优索引配置, 不用访问剩下(虚线所连接部分)的索引配置.

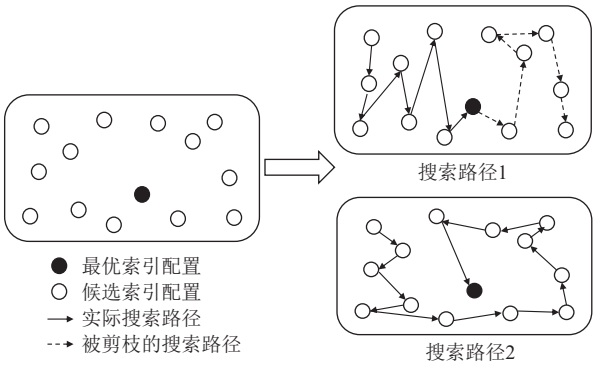


Fig. 2 The search for the best index configuration

图 2 最优索引配置的搜索

图 1 中这 3 个部分内部相对独立, 但外部联系紧密. 搜索空间中的候选索引配置是索引配置枚举部分的输入, 而索引配置枚举过程需要对索引配置进行评价以更快地找到最优配置. 不同的研究方案在这 3 个部分的具体设计存在不同, 但索引选择方案的效果和这 3 部分的整体设计相关. 大多数研究, 比如文献 [3, 11] 为了减小索引配置搜索空间会尽早对候选索引进行剪枝, 提高索引选择方案的效率. 但这种方式可能会错误地把一些可能后期被证明有用的索引剔除, 因此文献 [23] 选择尽量晚地对候选索引进行剪枝, 即不在候选索引配置搜索空间生成过程中剪枝, 而在其递归的枚举搜索策略中动态剪枝, 以保证索引调优方案的效果.

同时, 我们将现有索引选择方案中的方法与技术也按照该调优范式进行整理和总结, 如图 3 所示(对应本文第 2~4 节的内容).

1.3 存在的挑战

本文将 ISP 存在的挑战归纳总结为 3 点:

1)从问题性质来说, 索引选择问题这种组合优化问题是 NP 难的. 数据库关系表的每个属性上都可以建立索引, 且在不考虑主索引的情况下, 不同索引可任意组合形成不同的索引配置, 导致组合型(combinatorial)的索引配置搜索空间. 文献 [9] 通过假设存在一个能够准确计算任意索引配置对每个工作负载查询收益的函数(即能对任意索引配置进行优劣评价), 将 ISP 简化为一个纯搜索问题, 并将该简化的 ISP 规约为 NP 难的稠密 k 子图问题, 从而证明 ISP 肯定也是 NP 难的.

2)对于当前数据库和工作负载, 如何准确快速地判断一个索引配置的好坏是 ISP 的一大挑战. 文献 [9] 假设的准确的收益计算函数通常是不存在的, 且现有大部分索引调优方案代价的最大来源[28] 便是

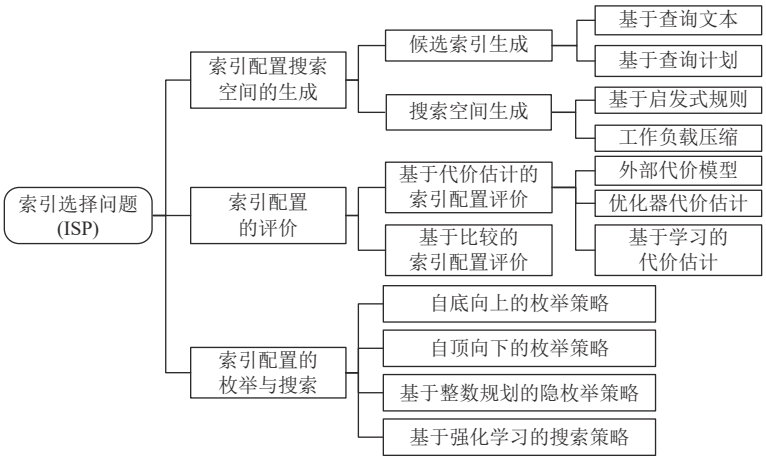


Fig. 3 Classification of related ISP methods

图 3 索引选择问题相关方法分类

索引配置评价过程. 虽然通过记录工作负载在不同索引配置下的实际执行时间能准确地通过数值来评价索引配置的优劣, 但实际执行的代价太过昂贵. 而使用优化器对工作负载在不同索引配置下的代价进行估计的方式更为轻量, 但存在估计不准的问题[29], 影响索引调优方案的效果. 因此, 实际的索引选择方案需要包含一个合理的索引配置评价机制, 平衡索引配置评价的准确性和代价.

3)索引间相互影响(index interaction)让最优索引配置的搜索变得更复杂[9,30]. 如果一个查询, 比如 SELECT *d* FROM *T* WHERE *a*=1, 能通过多个不同的索引完成执行, 比如$i_1(a,b)$和$i_2(a,c)$, 对于这个查询来说, i_1和i_2可看成是一种竞争关系(特别是在考虑存储空间限制和索引维护代价的情况下); 而如果一个查询, 比如 SELECT *d* FROM *T* WHERE *a*=1 AND *b*=2, 可以利用不同索引间的相互影响加速查询执行, 比如$i_3(a)$和$i_4(b)$, 那对于这个查询来说, i_3和i_4可看成是一种合作关系. 这些相互影响一方面让同一索引配置内的索引收益分配变得困难[9], 另一方面也让搜索过程变得更加复杂[30].

这些挑战展示了 ISP 的复杂性, 同时也吸引着众多研究者对这个问题的解决方案进行持续的研究和改进.

2 索引配置搜索空间的生成

2.1 可被索引属性和可被接受索引

索引配置由索引组成, 而索引被搜索键定义. 索引的搜索键可从数据库所有表的属性中进行选择, 如果考虑多列索引, 则可选择任意表上的若干属性进行排列组合, 形成以多个属性为搜索键的多列索引. 这些可能索引的数量是庞大的. 为了降低候选索引数量, 可以利用工作负载信息对索引键的属性搜索范围进行限制. 比如, 如果一个属性未出现在工作负载的查询语句中, 那么在索引的搜索键中包含这个属性将不能给工作负载带来收益, 反而可能导致索引存储和维护代价的增加, 因此可以预先将这些属性筛除.

现有研究工作常用 2 个概念筛选属性和索引: 可被索引属性(indexable columns)和可被接受索引(admissible indexes)[3,31-32]. 通常来说, 可被索引属性被定义为对构成索引有益的属性, 比如出现在查询语句 WHERE 子句中的属性, 有些文献中也称其为可行属性(plausible columns)[31]. 而根据可被索引属性生成的、对工作负载执行有潜在收益的索引则被称为可被接受索引.

为了抽取这些相关信息, 现有研究中的工作负载分析方法主要分为基于查询语句的方法和基于查询计划的方法. 基于查询语句的方法从查询语句文本抽取可被索引属性以生成可被接受索引. 比如, 抽取查询语句文本中的 SELECT, WHERE, ORDER BY, GROUP BY 子句出现的属性[3,33-35], 或通过规则和算法来筛选可被索引属性, 像 DB2Advis 的索引扫描智能属性枚举(smart column enumeration for index scans, SAFEIS)算法[11]. 这种方法只使用查询的文本信息, 简单高效. 基于查询计划的方法[31,36-37]则通过分析优化器生成的查询计划来获取可被索引属性和可被接受索引. 查询语句经过优化器的分析和改写, 生成的查询计划反映了真实的数据查询需求. 比如, 查询计划中 JOIN, AGGREGATE, SORT 算子的相关属性即可作为可被索引属性, 用于可被接受索引的生成. 这种方式需要额外调用优化器来生成查询计划, 但能

规避文本解析的局限性,比如优化器的查询重写过程可帮助消除空的子查询,排除干扰属性.

2.2 候选索引集合

从可被索引属性生成可被接受索引以作为候选索引的方式是可行的,但重点在于索引搜索键构建过程中的组合爆炸问题,即使通过可被索引属性概念限制搜索键的属性选择范围,可被接受索引的数量仍然很大,难以处理.很多早期研究工作只考虑单列索引[38-39]和/或单个关系表[8,38-41],虽不能完全避免组合爆炸问题,但在这些假设下,实际可被索引属性较少,可被接受索引数量也可控.然而这些假设是不切实际的:一个实际的数据库中不可能只有一个关系表,且多列索引在实际场景中也是十分常见且有效的.

针对这个问题,很多研究利用启发式规则或/和优化器进一步对可被接受索引集合进行限制,降低候选索引数量.比如,AutoAdmin[3]从单个查询出发,通过解析器(parser)抽取该查询的可被索引属性,生成可被接受索引.然后将这些可被接受索引作为该查询的候选索引,利用优化器和虚拟索引技术[25](将在3.2节进行更详细的介绍)找到对这个查询而言最好的索引配置(query-specific-best-index-configuration).以同样的步骤遍历工作负载中所有查询,将得到的这些单个查询的最优配置(即索引的集合)进行集合并集操作,形成当前工作负载的候选索引集合.此时,单个查询最优配置中的可被索引属性来自单个查询,而工作负载候选索引集合中的索引都来自这些最优配置,所以该方法有效降低了候选索引数量.和AutoAdmin通过解析器抽取可被索引属性并生成可被接收索引的方式不同,DB2Advis[11]首先在DB2优化器中执行SAFEIS算法,生成每个查询的有潜力索引(一种基于规则的索引生成算法,生成的有潜力索引集合是可被接受索引集合的子集).然后通过虚拟索引技术将这些虚拟索引注入DB2的模式信息模型,让优化器认为这些索引已被创建并可被用于查询计划的生成.最后在DB2优化器的RECOMMEND INDEXES模式下为工作负载的所有查询生成查询计划,并只将计划中被采用的虚拟索引加入候选索引集合,有效地减少了最终候选索引的数量.

然而这2种方法在候选索引的初始生成阶段只考虑面向单个查询的最优索引(配置),忽略了那些虽对任何一个查询都不是最优索引(配置)但可能对整个工作负载最优的索引(配置),导致有益的候选索引的疏漏,特别是在有存储空间限制或数据更新的情况下[20,42],这种情况在别的物理设计问题中也可能存在,比如数据库分区设计问题中[43].举个例子,如果一个工作负载W包含2个查询q_1,q_2,此时有3个可能索引i_1,i_2,i_3.其中i_1给q_1,q_2带来的收益分别为60和20,i_2给q_1,q_2带来的收益分别为30和70,i_3给q_1,q_2带来的收益分别为58和66,如果此时所剩空间只允许创建1个索引,那么对于工作负载W来说,上述2种方法可能都不会考虑索引i_3,因为i_3既不是q_1又不是q_2的最优索引,但i_3实际上却是对工作负载W的最优索引.针对这种情况,一些研究工作通过索引变换(index transformation)操作对候选索引集合进行调整[36,42,44],缓解候选索引生成过程中疏漏有益索引的情况.其中的索引变换操作包括合并(merging)、分割(splitting)、约减(reduction)和提升(promotion).具体来说,合并操作合并2个候选索引生成1个新索引,消除这2个索引的共有属性以降低存储代价,并尽量保留原有索引的查询加速能力,比如索引$i_1(a,b)$和$i_2(a,c)$可以被合并为$im_1(a,b,c)$或$im_2(a,c,b)$.分割操作则分离输入索引的公有属性和剩余属性以形成新索引,比如索引$i_3(a,b,e)$和$i_4(a,b,c,d)$的分割操作输出为$is_1(a,b)$,$is_2(c,d)$,$is_3(e)$.约减操作,有的文献中也叫前缀(prefixing)操作[36],通过取一个候选索引的搜索键前缀来形成新索引,比如索引$i_5(a,c,d)$可通过约减操作形成索引$ir_1(a,c)$和$ir_2(a)$.提升操作指的是把没有聚集索引(clustered index)和主索引的关系表上的特定索引提升为聚集索引的操作.和主索引一样,聚集索引所在关系表的数据会按照聚集索引搜索键值的顺序进行组织,但是其搜索键不需要是唯一的.提升操作会改变关系表数据的存储顺序,进而改变该表上其他索引和该索引的关系,影响该表上的访问路径选择和查询计划生成.这些变换操作考虑了索引在DBMS实际查询优化和执行过程的作用以及查询间、索引间的关系,能从当前候选索引集合出发,生成更多对工作负载可能有益的候选索引.

2.3 候选索引配置搜索空间

得到候选索引集合S后,如果不考虑存储空间限制,候选索引配置形成的搜索空间C可以看成候选索引集合S的幂集(power set),即$C=\mathcal{P}(S)=2^S$;如果考虑存储空间限制,搜索空间中那些所需存储空间大于给定限额的索引配置则需要被去除.根据这种思想,同时结合物理设计配置(比如索引配置和视图配置)对合并和约减操作的闭包性质,Bruno等人[44]提出一种形式化表示物理设计配置搜索空间的方法,也可直接被用于只考虑索引的情况.

ISP的搜索空间规模往往影响着索引选择方案的效果. 比如在决策支持系统(decision support system, DSS)场景下, 数据库中关系表及其属性的数量可能很多, 导致候选索引数量也非常多, 直接影响着候选索引配置的数量(即搜索空间的大小), 进而影响索引配置评价过程的代价和索引选择方案的总代价. 同时, 一个索引选择方案在不同搜索空间规模下的表现也是该方案可伸缩性的重要指标.

我们认为, 可以通过各种手段在保证质量的情况下尽量缩减搜索空间, 比如通过可被索引属性、可被接受索引这样的概念或自定义规则来排除一些候选索引. 还可以根据每个查询在整个工作负载中的代价占比情况, 比如在候选索引生成过程中只考虑工作负载中最昂贵的k个查询[12]或一些工作负载压缩方法(将在2.4节进行详细介绍), 通过减少工作负载中的查询数量来降低候选索引数量. 此外, 还可以选择直接对搜索空间进行剪枝, 比如利用存储空间约束条件将所需存储空间大于指定阈值的索引配置从搜索空间中删除; 或通过高效的枚举算法降低搜索空间规模的影响, 比如, AutoAdmin[3]中的迭代式枚举算法通过启发式规则来避免对整个搜索空间的遍历, 以实现对搜索空间的高效搜索.

2.4 工作负载的压缩

索引选择方案的复杂性随工作负载大小的增长而呈二次方增长[45], 所以在工作负载驱动的问题中, 可通过工作负载压缩(也称为摘要[32])等预处理操作, 减少工作负载中的查询数量, 提高搜索空间的生成效率, 缩减搜索空间大小.

数据库工作负载压缩方案的常用评价指标[32,45-47]有3个, 即压缩结果的覆盖性、代表性和压缩方案的效率. 其中, 覆盖性保证压缩后的工作负载可覆盖原工作负载中大部分重要查询, 代表性保证压缩后工作负载应尽量保留原工作负载的特征(比如查询分布), 这2个指标要求压缩过程要保证压缩质量. 而压缩方案的效率则影响压缩方案的可用性, 保证压缩带来的收益显著超过压缩所需的代价(比如压缩执行所需的时间和资源等). 一个合理的压缩方案需要根据任务的具体要求在这3个指标上找到平衡. 数据库工作负载压缩最直接的一种方法, 即不考虑查询顺序, 将工作负载表示为查询和查询频率的对, 减少索引选择方案要处理的查询对象. 另外一种直接的方法则用查询模板[15,48-51]对工作负载进行压缩. 这2种直接的方法只使用查询语句的文本信息, 得到工作负载后可快速完成压缩, 压缩结果的覆盖性和代表性相对可控. 也有研究利用查询计划的模板[52-53]来提高索引选择方案的效率. 虽然这种直接模板化的方式能加快搜索空间生成的速度, 但不能减小索引配置的搜索空间, 因为查询所包含的可被索引属性并没有减少. 同时, 在索引配置评价过程中, 查询模板也不能代替查询实例, 因为查询模板不能够直接在DBMS中执行, 且查询模板的一个实例通常并不能代表该模板下的所有查询实例. 比如, 当属性数据不是均匀分布的时候(实际中的大部分情况), 同一索引配置对同一查询模板下不同查询实例的收益是存在差异的, 即一个索引配置对一个查询实例的收益不能够代表这个索引配置对该模板下所有查询实例的收益情况. 此外, 实际场景中的查询是多样的, 工作负载包含的查询模板数量可能仍然很多[32], 导致压缩效率没有保障.

为了进一步提高工作负载压缩方案的效率, Chaudhuri等人[46]系统性地提出了一种数据库工作负载压缩的思路: 其目标是找到一种工作负载压缩算法, 使解决方案在输入变为压缩后工作负载时仍能得到质量相近的结果(隐含了对压缩结果的覆盖性和代表性的要求), 提高工作负载驱动任务(比如索引选择问题)解决方案的效率. 其中心思想是将工作负载的压缩看成是一个查询聚类问题, 在解决方案质量损失最大允许范围内, 尽量降低压缩后工作负载中的查询数量. 该方法需要事先对聚类过程中的距离函数进行定义, 但有些场景下的距离函数并不容易被定义. 针对这个情况, Chaudhuri等人[54]又设计了一种适应大部分场景的通用工作负载分析语言WAL. WAL在SQL语法基础上设计并集成了2种新原语(primitives): 支配(dominance)和表示(representation), 通过这种声明式语言的形式可定制地、轻量化地筛除工作负载中的冗余查询, 支撑工作负载的摘要和压缩任务. 比如, 支配原语可用来描述查询语句谓词集合间的子集关系, 通过过滤工作负载中那些被支配的查询语句, 对工作负载进行压缩. 但这种通用压缩方法[46,54-55]没有考虑到索引选择问题的特殊性, 压缩后可能还会保留很多发生频率高但索引收益上限低的查询. 针对这个问题, Siddiqui等人[32]提出了一种在索引选择背景下的查询表征方法, 用来计算索引感知(indexing-aware)的查询相似性, 实现工作负载的压缩, 提高索引选择方案的效率.

2.5 小 结

本节内容聚焦如何生成候选索引和候选索引配置. 现有研究方法通常通过启发式规则从工作负载

查询文本或从查询计划抽取相关的属性和索引(比如可被索引属性和可被接受索引),形成候选索引和候选索引集合.其中基于查询语句的方法简单高效,而基于查询计划的方法能生成更为准确的候选索引集合,但这种方式需要调用优化器生成查询计划,代价相对更大.为了提高候选索引配置生成的效率,还有很多研究通过压缩工作负载来减少需要处理的查询数量,基于模板的方法能加快搜索空间生成的速度,但不能缩减索引配置的搜索空间,同时在查询模板数量很多的实际场景下不能保证压缩效率.基于聚类的方法和把压缩操作抽象为 SQL 原语的方法则能进一步提高工作负载的压缩效率.我们认为,未来的查询相似性度量和工作负载压缩机制的设计应关注文献 [32] 中的索引感知性质,使得到的查询聚类结果更具有针对性,减少因工作负载压缩引起的索引选择方案效果的损失.

3 索引配置的评价

索引配置的评价是索引选择方案的核心内容之一.其思路一般可分为 2 种,其中最自然、使用最多的一种思路是用数值记录索引配置对工作负载代价的影响程度.比如,用工作负载在特定配置下的代价来评价该配置,工作负载在一个索引配置下的代价越低,则代表该索引配置对当前工作负载越好.另一种思路则通过构建索引配置间的偏序关系来评价索引配置的优劣.如果一个配置比其他配置都好,那么这个配置显然也是最优配置.在前一种思路中,最直接的方法就是将工作负载的实际执行时间作为工作负载代价,但这种方式太过昂贵.特别是在处理联机分析处理(online analysis process, OLAP)型工作负载情况下,一个长查询的执行可能耗费几个小时,且索引选择方案需要在多个不同的索引配置下对相同的工作负载进行代价估计.因此,大多数研究选择用优化器估计工作负载在特定配置下的代价.但优化器的代价估计是不准确的,可能导致次优的索引配置推荐,甚至数据库性能的退化[29].如何设计一种高效且相对准确的代价估计方法是这种思路的关键.而在后一种思路中,由于不需要确定工作负载代价的具体数值,索引配置间准确偏序关系的构建成为该思路的重点,比如通过机器学习建立配置偏序关系的模型,能快速且准确地在当前工作负载情况下对索引配置进行比较,得到最优索引配置.本节首先对大多数研究所采用的第 1 种思路的代价估计方法进行归纳和总结,然后对使用机器学习进行索引配置比较的第 2 种思路进行分析介绍,最后对索引的更新维护代价问题进行论述.

3.1 外部代价模型与优化器估计

很多早期研究采用轻量的外部代价模型[38,41,56-57]估计特定索引配置下工作负载代价以评价索引配置,但这些代价模型通常会对工作负载和数据库状态做出诸多假设,比如数据库中只有一个表;各属性值的数据分布是独立的,实用性不强.而且这种方式让索引选择过程和优化器脱节[3,31],即使外部代价模型更准确,得到的索引配置仍可能不会给查询执行带来预期收益.因为在不改动优化器的情况下,优化器仍按照其内部代价模型生成计划,所以可能并不会采用被推荐的索引.如果想要外部代价模型产生较好效果,在保证外部代价模型准确的基础上,还需要侵入式地对 DBMS 优化器的查询计划生成模块进行修改,确保优化器能采用被推荐的相关索引.但这也意味着更高的工程实现难度和更低的可迁移性(portability).针对这种情况,Finkelstein 等人[31]在研究数据库物理设计问题时,提出直接使用优化器内部代价模型来估计工作负载代价的思想和使用优化器实现同步(in-sync),也让索引选择过程能契合目标数据库优化器特性.大部分后续相关研究[3,9,11,23,44,52,58-59]都采用优化器对工作负载代价进行估计,大多数 DBMS 也实现并暴露了相关接口,比如 PostgreSQL, SQL Server, DB2 中的 EXPLAIN 命令.

3.2 what-if 优化与虚拟索引机制

虽然优化器可对特定索引配置下的工作负载代价进行估计,但是一些索引配置调整操作的代价是昂贵的,比如索引创建操作,所以先实际调整当前索引配置到目标索引配置再进行代价估计的方式是低效的.现有研究通常采用 what-if 优化(what-if optimization)技术[25]对物理设计的调整进行模拟,让优化器支持在虚拟的物理设计配置下估计工作负载代价,避免对配置进行实际调整.在 ISP 中即意味着对索引结构的模拟,其可行性依赖优化器的查询优化机制:优化器在生成查询计划过程中不需要使用索引的实际数据(在查询执行过程才需要使用索引的实际数据),只需要使用索引的源信息,比如索引键的信息、索引键所在表的信息、索引大小的信息、是否是聚集索引等.根据这种思想,Chaudhuri 等人[25]在 SQL Server 7.0 的虚拟配置模拟模块实现了虚拟索引机制.别的商业数据库一般也实现了类似的虚拟索引机制,比如在 DB2 的 RECOMMEND INDEX 模

式[11]、Oracle 的 SQL 访问路径顾问模块[13] 中. 它们实现的基本功能为虚拟索引的管理, 比如创建、删除及考虑虚拟索引的查询计划生成功能.

一个值得关注的重要问题是 what-if 优化过程的代价问题. 虽然相比实际执行, what-if 优化技术确实能大幅度降低调优代价, 但 what-if 优化过程中对优化器的调用, 也被称为 what-if 调用, 仍占据索引调优方案大约 90% 的调优时间[28]. what-if 调用次数主要受需要进行 what-if 优化的候选索引配置数量的影响. 关于如何减少 what-if 调用的问题, Finkelstein 等人[31] 提出了原子配置(atomic configurations)概念, 限制每个表上最多只有 1 个索引, 且仅在对工作负载在原子配置下的代价估计时进行 what-if 优化, 并利用原子配置下的代价信息推导非原子配置下的代价, 减少配置评价过程中的 what-if 调用. AutoAdmin[3] 考虑到优化器特点, 对原子配置概念进行发展, 提出单连接原子配置(single-join atomic configurations)概念, 限制每个表上最多同时使用 2 个索引的交集以及面对多表查询时最多只考虑 2 个表上的索引, 进一步降低了需要进行 what-if 优化的配置数量.

除了通过原子配置概念对需要进行 what-if 优化的索引配置进行筛选, 还可以利用索引选择方案的巧妙设计减少 what-if 调用. 比如, DB2Advisor[11] 的候选索引生成策略能在 1 次对 DB2 的优化器调用中同时进行单个查询的候选索引生成工作和该查询的最优索引配置选择工作, 使后续步骤不再需要调用优化器对该查询进行查询优化, 降低 what-if 调用次数. 因为对这个查询来说, 优化器认为的最优索引配置已被优化器选中并用于查询计划的生成, 或通过缓存 what-if 调用结果[58,60] 来避免对优化器的冗余调用.

还有研究工作[12,61] 通过减少工作负载中所需处理查询数量来减少需要进行 what-if 优化的配置数量, 比如工作负载压缩, 或利用索引配置间关系动态降低 what-if 优化所需考虑查询的数量[23,44], 比如, 如果 1 个待评估索引配置和 1 个已被评价索引配置只差 1 个索引, 那么只需重新估计和该索引相关的查询在待评价索引配置下的代价, 即可通过相关查询在不同配置间的差值推导出工作负载在待评价索引配置下的代价.

3.3 基于机器学习的配置评价

随着机器学习技术的发展, 越来越多数据库优化问题的解决方案引入了机器学习技术[62]. 本节将分析讨论使用机器学习评价索引配置的思路和方法.

3.3.1 基于机器学习的代价估计

基于优化器的代价估计可能存在不准确的情况, 影响索引选择方案的效果[29]. 现有 DBMS 的代价估计通常由优化器在计划生成阶段完成, 即通过表、(子)表达式的基数估计和代价模型计算查询的代价. 其错误可以分为基数估计引入的错误和代价模型引入的错误, 且相比于代价模型, 基数估计对代价估计的影响更大[63], 所以很多研究致力于研究准确估计各计划节点基数的方法. 基于摘要的传统基数估计方法, 比如直方图法、采样法可能经常会出现不准确的情况, 特别是在数据库表或属性列间存在着关联关系时[64], 而机器学习强大的数据抽象能力能捕获查询和数据的特征, 帮助更准确地估计基数.

一种经典的基于机器学习的基数估计方法是 Kipf 等人[65] 提出的 MSCN 模型. 该模型会抽取工作负载查询中所涉及到的表信息、查询的连接信息以及谓词条件信息, 并通过集合卷积(set convolution)操作整合这 3 种特征信息以捕捉对基数估计存在影响的特征. 整合后的特征向量被输入到一个 2 层的全连接神经网络, 输出即为估计的基数值. 而 Sun 等人[66] 观察到即使通过机器学习模型获得了较准确的基数估计结果, 但还是可能由于代价模型的错误而得到不准确的代价估计结果, 所以提出了一个端到端的、能同时对基数和代价进行估计的学习模型. 该模型通过收集查询的物理计划、计划代价以及计划基数, 对影响查询代价的 4 个主要因素, 即物理计划操作信息、谓词信息、源信息以及数据信息进行特征编码. 然后输入到一个树形模型进行多任务训练, 得到每个计划节点的稳定表征. 最后输入到一个 2 层全连接神经网络对(子)查询的基数和代价进行估计.

现在也有索引选择方案使用基于机器学习的代价估计模型. 比如, Zhou 等人[51] 通过收集数据库内核中能捕获到的查询的 I/O 和 CPU 代价作为特征, 以查询的历史代价值为标签, 训练一个深度回归模型来预测查询代价. Siddiqui 等人[67] 以查询模板为单位, 收集该模板下所有实例的模板参数选择率和索引配置信息, 并进行向量化. 然后训练一个树形机器学习模型对查询在特定索引配置下的代价进行预测. Gao 等人[68] 通过结合查询计划信息和相关索引信息, 训练一个图卷积神经网络来对查询代价进行估计, 并用于索引调优. 在类似的视图选择问题上, Yuan 等人[69] 提出一种结合计划、关联表信息的查询表征方法, 并设计了一种 Wide-Deep 模型预测查询在特定视图配置下的代价. 该模型的核心思想是用一个宽的线性模型来编码数值型特征, 用一个深的神经网络模型来

编码文本型特征. 结合这些特征编码作为模型的输入进行训练，得到可预测查询代价的机器学习模型.

基于机器学习的代价估计在索引选择方案中的应用仍不多. 我们认为，可能的原因包含 2 个方面：一方面，基数估计/代价估计的模型准确性和稳定性问题使得相关研究仍属于早期起步阶段，相关研究虽然探索了用机器学习方法预测查询代价，但仍没有文献对其中的代价估计模型进行详细的对比分析；另一方面，和外部代价模型一样，为了得到预期收益，仍避免不了对数据库优化器代码进行侵入式修改. 但我们认为，为了获得更好的索引调优结果，这仍是一个具有前景的方向.

3.3.2 基于机器学习的索引配置比较

ISP 的目标是找到最优索引配置，实际上并不关心在每个候选索引配置下工作负载代价的具体数值，只关心能最小化工作负载代价的索引配置，所以通过索引配置的偏序关系来评价索引配置的思路也是可行的. 据我们所知，该思路现有研究只有 Ding 等人[29]的工作(本文称为 DingIdxAdvis). 该工作指出优化器估计不准导致索引调优效果变差的问题，并用分类模型实现索引配置的比较和评价. 具体来说，文献 [29] 从生成的查询计划中抽取物理操作符(比如索引扫描、表扫描、哈希连接等)信息作为特征，并附加这些操作的执行模式信息(以行还是批为执行单位)和并行信息(是否是并行执行)，形成〈Physical Operator〉_〈Execution Mode〉_〈Parallelism〉形式的物理操作符标识键，比如 Scan_Batch_Parallel, HashJoin_Row_Serial. 这个标识键会被赋予特定数值，用来评估这个物理操作符的计划完成量，比如当前操作在计划中的预期代价或计划结构信息加权的行数总和. 通过整合查询计划中所有操作的特征即可形成一个计划的向量表示. 对于 2 个不同的索引配置下产生的不同查询计划，将它们的向量表示进行差值整合作为分类器的输入，分类器的输出为从一个计划变为另一个计划后查询性能变化类别的预测结果，比如性能是提升了还是退化了，亦或是不确定(不确定时使用优化器估计进行比较)，通过对比不同索引配置下所产生计划的性能来对索引配置进行比较. 最后在索引配置搜索过程中，只保留那些能带来非退化性能的索引配置，并不断更新已搜索到的最优索引配置.

3.4 索引的更新维护代价

当工作负载中包含插入、删除以及更新类型的语句时，数据库相关表的数据会更新，此时定义在这些表上的索引也需要进行更新维护. 在 OLAP 场景下，业务数据变化频率低，工作负载的执行时间较长，所以经常选择忽略索引的维护代价. 这也是大部分 ISP 的研究背景为数据仓库和 OLAP 场景的原因之一. 而在联机事务处理(online transaction processing，OLTP)场景下，工作负载需要频繁且快速地对数据库数据进行读取、写入和更新，如果一个索引被定义在需要频繁更新的表上，那么这个索引可能需要经常进行维护，产生大量维护代价，也不能再被忽略. 所以需要设计策略以减少索引的维护代价，比如，通过存储空间约束[52,70]或索引合并[42]等手段尽量减少索引配置中的索引数量，降低维护发生的概率.

对于索引更新维护代价的估计，现有研究大多先会对查询语句进行分割，形成语句的 SELECT 部分和 UPDATE 部分[3,36,59,61]，并根据 UPDATE 部分计算索引的更新维护代价. 其中，SELECT 部分和普通的 SELECT 语句类似，UPDATE 部分(也称为更新壳，update shell[30,36,44])则对 SELECT 部分选中的记录进行更新操作. 前期的一些索引选择方案[4,7,38]会对各类型语句的概率进行建模，并建立外部模型来计算更新维护代价[39]，但这些方法在数据库、工作负载等方面做出诸多假设，比如只考虑 1 个表[7,38]. 同时索引的更新代价还与查询执行机制[51]、索引配置中其他索引有关[30]，让索引更新代价的准确估计变得更难. 针对这个情况，Zhou 等人[51]以索引更新操作的 I/O 代价、CPU 代价和数据查找代价作为输入特征，以查询处理代价作为输出，训练一个深度回归模型来学习索引更新操作代价中 I/O 代价和 CPU 代价间的比例关系以及各输入的代价特征和查询处理代价之间的关系，对更新代价进行考虑.

3.5 小 结

本节聚焦如何高效准确地评价索引配置的优劣. 早期基于外部代价模型的方式虽然效率高，但为了模型可用性做出诸多假设，且和优化器脱节，所以已很少在索引选择方案被直接使用. 现有主流方法采用优化器和 what-if 优化机制来估计工作负载在特定索引配置下的代价，实现对不同的索引配置的比较. 这种方式和优化器同步，但不准确的优化器代价估计可能导致索引选择方案效果出现偏差，甚至造成数据库系统性能的退化. 同时，为了保证索引选择方案的效率，还需要注意 what-if 优化的代价问题，可以通过原子配置概念、what-if 调用结果的缓存与复用或通过减少工作负载中所需处理查询数量来减少需要 what-if 优化的配置数量(这些方法可结合使用). 而随着机器学习技术的发展，越来越多的研究者开

始研究用机器学习技术提升索引配置评价的准确性和效率，虽然该部分方法在代价估计的平均误差低于传统优化器，但要达到可用，仍需关注机器学习模型的训练（包括更新）代价和稳定性问题，并对数据库内核中优化器部分代码进行修改.

4 索引配置的枚举与搜索

如2.3节所述，虽然利用启发式规则可以缩减索引配置搜索空间，但仍不能改变该搜索问题的NP难属性，且索引配置评价过程的代价也不可忽略，因此通过暴力枚举、评估、比较所有候选索引配置来找到最优配置的精确算法是不切实际的，需要设计一种精巧的枚举或搜索算法，牺牲一定的精度和确定性，保证代价相对可控的情况下，更快地找到“最优”配置. 针对这个问题，本文将当前范式下的枚举/搜索策略总结为4种：自底向上（bottom-up）的枚举策略、自顶向下（top-down）的枚举策略、基于整数规划（integer programming, IP）的隐枚举策略和基于强化学习（reinforcement learning, RL）的搜索策略.

4.1 索引间的相互影响

虽然在不考虑主索引和存储空间约束的情况下，一个索引配置可包含任意多个不同的索引，但是索引间不是完全独立的，如果存在正向或负向相互影响的索引出现在同一索引配置中，会产生相互影响[30]. 这种相互影响与工作负载中查询、索引搜索键及优化器查询优化机制相关，影响索引调优方案的效果.

现有工作负载驱动的索引选择方案隐式地[3,11,23,36,39]或显式地[9,30,71]在索引推荐过程中考虑索引间的相互影响. 隐式的方法通常与枚举/搜索策略的设计相关，而显式的方法则直接对索引间的相互影响进行量化评估. 比如，Schnaitter等人[30]提出索引间交互度（degree of interaction）概念来衡量索引间的相互影响. 具体来说，交互度衡量的是一个索引在另一个索引加入索引配置前后工作负载代价的变化幅度，交互度大则意味着索引之间关联紧密. Bruno等人[71]则提出有用度（usefulness）概念来衡量索引间的相互影响，并用于索引收益的计算. 其核心思想在于比对索引搜索键间的包含、前缀等关系. 比如，如果索引i_1搜索键中不包含索引i_2搜索键中任一属性，那么索引i_1完全不能实现索引i_2在关系表访问加速上的功能；而如果索引i_2搜索键就是索引i_1搜索键的前缀，索引i_1则完全可以被用来替代索引i_2来完成关系表访问加速的功能，虽然替代后可能导致更高的索引访问代价和索引更新代价.

4.2 自底向上的枚举策略

自底向上的枚举策略从空的或当前索引配置出发，逐渐向这个配置依照一定策略加入新索引、形成新配置，直到达到设定的终止条件，比如任意新索引的加入都会导致违反存储空间约束条件停止.

AutoAdmin[3]提出一种自底向上的枚举方法*greedy*(m,k). 该算法首先暴力遍历所有由m个索引组成的索引配置（$m \leqslant k$），隐式地考虑存在重要相互影响的索引，让那些有强正向相互影响的索引能提前被加入索引配置中. 然后进入迭代的索引配置扩展阶段：迭代每一步找一个能给工作负载代价带来最大缩减的索引加入索引配置，直到配置中索引数量达到k时停止. DB2Advisor[11]则提出了一种基于背包问题的ISP建模：将配置内的索引当成背包里的物品，索引的大小看成物品的重量，索引收益（索引的存在与否带来的工作负载代价差异）看成物品的价格. 这个问题是NP难的，所以DB2Advisor设计了一种启发式规则来近似处理，即以每单位存储收益对搜索空间内的索引进行排序，并依次选取剩余排序最高的索引加入索引配置，直到加入下一个索引会导致违反存储空间约束时停止. 但在这个过程中（特别是索引收益的计算过程），文献[11]没有充分考虑索引间的相互影响，导致索引选择方案效果受到影响. Chaudhuri等人[9]针对这个情况提出了BEN_KNAP方案，迭代地缩放各索引的收益值范围，使索引配置内索引的收益值能尽量满足线性关系（索引配置收益等于配置内所有索引收益之和），实现在考虑索引间相互影响的情况下计算各索引的收益，提升索引调优效果.

Extend[23]则提出一种递归的索引配置构建策略. 该方案不预先对候选索引进行剪枝，防止过早排除可能有用的索引. 虽然该方案仍是自底向上扩充索引配置，但和AutoAdmin[3]，DB2Advisor[11]方案不同，该方案第1步会选择收益最大的单列索引加入空的初始索引配置，后面的每一步有2种可能选择：第1种是从剩下未被选中的单列索引中选择收益最大的单列索引；第2种则在从当前索引配置中所有单列索引上扩展一个属性形成扩展索引，并选择能带来最大收益的那个扩展索引. 最终选择这2种可能选择中收益更大的那个. 该方案在每一步对索引配置的扩展中考虑了索引配置中已有索引的情况，隐式地对索引间的相互影响进行了考虑.

4.3 自顶向下的枚举策略

与自底向上的枚举策略相反，自顶向下的枚举

策略首先从一个不考虑约束条件的初始配置出发，逐渐对索引配置进行变换，直到能满足约束条件. 初始配置的选择影响后续配置枚举的范围，可能的选择包括所有可能由索引组成的索引配置[39]、通过启发式规则得到的候选索引组成的索引配置[36,72]等.

Whang[39]提出了一种自顶向下的枚举策略 DROP. 该策略从一个包含所有可能索引的初始索引配置出发，每次挑选k个索引(初始的k设为 1)，使删除那k个索引后可以给工作负载代价带来最大缩减. 当每k个索引的删除不再能给工作负载代价带来缩减时，给k的值加 1($k=k+1$)，重复上述过程，直到达到终止条件(比如达到设定的最大k值)停止. Bruno 等人[36]则提出了一种 Relaxation 枚举策略. 其初始配置为一个不考虑存储空间约束的最优配置，然后开始迭代：首先检查初始/当前索引配置是否满足存储约束条件，并评估这个配置的工作负载代价；然后选择一个能带来最小松弛惩罚，即每存储单位减少带来的工作负载代价增加的索引变换操作对当前配置进行变换，得到存储开销更低但更低效的松弛(relaxed)配置，直到满足存储约束条件时停止.

关于自底向上和自顶向下的枚举策略，结合文献 [21] 进行分析：当存储限额较低时，满足条件的索引配置中索引数量通常较少，自底向上的枚举策略能较快地找到最优索引配置；而自顶向下的枚举策略由于初始配置的大小和索引变换操作的多样性，需要评价的候选配置数量相对更多，算法运行时间更长，但调优效果一般更好. 而如果存储限额较高，自顶向下的枚举策略可用更少的迭代次数和配置评价次数，更快地使松弛索引配置满足存储约束条件，提高效率.

4.4 基于整数规划的隐枚举策略

如 4.2 节所述，索引选择问题可以被建模为背包问题. 而整数规划也是解决背包问题的经典方法之一. 这类方法将索引(物品)是否存在于索引配置(背包)中的状态建模为 0-1 变量，然后通过这些变量表达 ISP 的目标和约束条件，得到一个最优化问题的数学模型. 其解空间即对应索引配置的搜索空间，其求解过程则对应索引配置的枚举/搜索，通过检查一部分的变量组合来找到最优解.

比如，Caprara 等人[73]将索引选择问题看成广义无容量限制设施定位问题(generalized uncapacitated facility location problem, GUFLP)，用 0-1 线性规划建模，并提出一种基于分支定界法的精确解法和一些基于松弛和启发式规则的近似解法. Papadomanolakis 等人[59]则将 GUFLP 的建模应用到商业数据库的实际场景中，消除了 GUFLP 中的 1 个查询只能使用 1 个索引的假设[31,74]，即不支持索引交集技术，并用线性优化中的敏感性分析(sensitivity analysis)技术获取参数区间(intervals)，分析特定索引是否应该被加入索引配置，使该数学模型能被更高效地求解. CoPhy[52]方案观察到 ISP 解空间的良好结构，证明了 ISP 在结合类似 INUM[58]和 C-PQO[28]等快速代价估计技术后仍可被建模为 0-1 整数规划问题，提高了求解效率.

对于小规模的 ISP，基于整数规划的方法能够很快地返回该背包建模的理论最优解(即最优索引配置). 但随着问题规模的增大，整数规划系统中变量的数目由于组合爆炸而指数级增加. 虽然随着整数规划研究的持续发展，已存在很多优化解答器(optimization solver)能对这种较大规模的整数规划问题进行高效求解，比如 Gurobi optimizer[75]，CPLEX optimizer[76]，但其最优性保证建立在准确计算每个候选索引的大小和收益值的基础上，面临索引间相互影响带来的挑战[9].

4.5 基于强化学习的搜索策略

随着强化学习特别是深度强化学习技术的兴起，越来越多强化学习技术被用于数据库优化问题，比如 DBMS 的参数调优[77-78]、分区[79]、连接顺序选择[80-82]、视图的物化选择[83]、索引选择[15,49-51,84-88]问题，给优化策略提供从错误中学习的能力. Schaarschmidt 等人[89]也将数据库优化任务抽象为强化学习任务，并提出示教学习(learning from demonstration)和代价模型自展(cost model bootstrapping)2 种可能提升强化学习方案实用性的策略以及一种应对规模增长的增量学习(incremental learning)方法，为强化学习算法在数据库优化任务中的应用提供指导.

从 4.2 节和 4.3 节可以看出，自底向上和自顶向下的枚举策略依赖启发式规则的设计，但这些启发式规则是固定的，在面对不同的数据库和工作负载时常常可能导致错失最优索引配置，且没有从错误的选择中进行学习的能力. 这意味着这些策略未来遇到相同状况还是会犯一样的错误. 而索引选择方案这种逐步向索引配置加入合适索引的过程可以看成是一个序列决策问题，利用强化学习理论来进行建模和求解，即通过试错法(trial-and-error)训练索引调优智能体，使智能体能根据数据库状态和工作负载情况，自动做出索引调优决策，调整索引配置.

具体来说，为了将 ISP 建模为序列决策问题(该过程也可被称为强化学习建模)，需要将 ISP 上下文

映射为强化学习任务的基本元素，即环境（包括奖励、状态）和智能体（包括动作、策略）. 最直接的一种强化学习建模就是将 ISP 建模为一个情节性强化学习（episodic reinforcement learning）[90] 任务，将 DBMS 作为环境，将索引选择策略作为智能体，智能体在每一个情节（episode）中完成一次完整的索引配置调优：对于一个工作负载，智能体通过观测 DBMS，使用策略函数计算所需采取的动作，比如创建一个特定的索引，然后采取动作对现有（虚拟）索引配置进行变更；DBMS 通过评价新配置的好坏，比如工作负载代价的缩减给智能体一定数值的奖励，新配置越好，奖励值越大. 智能体在不断与环境的交互过程中更新其策略函数，以获得更大的情节回报（episode return）. 其中，可将约束条件用作情节的终止条件，比如索引数量限额，也可将约束因素，比如存储空间约束中的索引存储空间作为惩罚项，用于调整智能体动作的奖励值.

比如，Sharma 等人[85] 提出一种基于深度强化学习的单列索引推荐方案 NoDBA. 智能体动作为一个特定索引的创建，其状态包含数据库属性列的选择率信息和 DBMS 现有索引配置信息. 奖励函数为索引创建后带来的工作负载代价缩减，其使用交叉熵方法进行训练. 但 NoDBA 建立在一些过度简化的假设上，比如只考虑 1 个关系表，且只能推荐单列索引，这使得该方案很难在实际场景中被应用. Licks 等人[91] 提出一种基于强化学习的索引选择方案 SmartIX，虽然考虑了多表情况，但也只考虑单列索引的推荐. Lan 等人[86] 采用一种基于深度 Q 网络[92]（deep Q-network，DQN）的深度强化学习算法的索引推荐方案 LanIdx-Advis，且设计了启发式规则来生成候选索引，缩减动作空间. 但这种基于规则的候选索引生成方法可能误删一些实际有益的索引[23]，导致次优的推荐结果. Lai 等人[87] 把单列和多列索引的推荐融入强化学习建模，并探索了利用基于近端策略优化（proximal policy optimization，PPO）的深度强化学习方法实现索引配置的推荐. Wu 等人[88] 在考虑 what-if 调用次数的情况下，使用基于蒙特卡罗树搜索（Monte Carlo tree search，MCTS）的强化学习方法对索引选择问题进行求解（本文称该方案为 MCTS_IT）. 与经典的强化学习方法不同，MCTS 是一种决策时规划（decision time planning）方法，不会维护一个全局策略函数，只会维护已知状态的局部策略函数. 在遇到新状态时通过蒙特卡罗试验得到该状态下的动作，并反向更新已知状态的策略函数[90]. MCTS_IT 将索引配置自底向上枚举的过程映射成搜索树生成的过程，不断通过动作策略生成新的索引配置，形成搜索树，最后通过贪婪算法遍历搜索树找到最优索引配置.

我们认为，基于强化学习的索引选择方案在序列决策的每一步从当前索引配置出发，隐式地将索引间相互影响纳入了考虑. 同时这种逐步扩充索引配置的方式也可看成是一种自底向上的搜索策略. 但和传统自底向上的枚举策略不同，在基于强化学习的索引选择方案中，索引配置的每次扩充对应着智能体策略函数控制的一次动作，不需要重复地枚举、评估和比较所有可能的扩充索引配置. 智能体的策略函数相当于一个灵活可变的启发式规则，在与 DBMS 的交互过程中学习（包括从错误中学习），不断改进.

4.6 方法对比

4.6.1 常用实验设置与评价指标

除了真实的 DBMS 数据和工作负载，现有索引选择方案最常用的公共数据库数据和工作负载来源为 TPC-H[93]，TPC-DS[94]，JOB[63]，均为决策支持型测试基准，这些基准测试套件中都包含相应的数据库数据填充程序和基于模板的查询生成程序.

索引选择方案的常用评价指标为推荐索引配置下的工作负载代价和调优时间. 比如代表性工作负载在特定索引配置下的执行时间越低配置越好；或与初始配置下执行时间相比的代价缩减（率）越高配置越好；调优时间则越低越好.

4.6.2 代表性方法对比

本节对现有索引选择问题的代表性方法进行整体的分析对比. DROP[39] 作为早期的代表性方法，设计了外部代价模型评估工作负载代价. 但 DROP 对 ISP 做出了一些简化假设，比如，为了让文献 [39] 提出的代价模型可用，假设查询语句只包含简单的等值谓词条件，且该方法没有考虑工作负载的特征，将数据库中所有可能的索引都作为候选索引，使得该方案的时间复杂度很高.

AutoAdmin[3] 和 DB2Advisor[11] 均基于规则和优化器生成搜索空间，但是它们对规则和优化器的使用方法不同. AutoAdmin 通过启发式规则迭代选择每个查询的最优索引配置，然后合并这些查询的最优索引配置作为工作负载的候选索引集合，并以相同的启发式规则选择整个工作负载的最优索引配置；而 DB2Advisor 针对每个查询调用 1 次 DB2 优化器，同时完成候选索引的生成（SAEFIS 算法）和选择（基于虚拟索引注入和优化器计划生成机制）. 此外，AutoAdmin

还利用原子配置概念和代价推导规则减少优化器的what-if调用, 其*greedy*(*m*,*k*)算法先确定一个由*m*个索引组成的最优种子配置再根据当前配置逐步加入一个最佳新索引的做法也隐式地将索引间相互影响考虑在内. 而DB2Advisor在其枚举阶段没有进行what-if调用, 导致枚举过程中不能考虑到索引间的相互影响, 只能通过设计一个额外的扰动阶段, 将枚举得到的索引配置进行索引随机替换, 有限地考虑索引间的相互影响.

DingIdxAdvis[29]和AutoAdmin[3]的主要差别在于前者在索引配置评价阶段使用了基于分类的机器学习模型对索引配置进行比较. BEN_KNAP[9]针对索引选择问题背包建模中索引收益计算通常没有考虑索引间相互影响的问题(比如DB2Advisor[11]中), 提出了一种显式根据索引间相互影响对配置内的各索引的收益进行分配/推导的机制, 提升索引选择方案效果. 该索引收益分配机制可以和各种搜索空间生成方法兼容, 所以文献[9]没有限定搜索空间的生成方法(实验部分使用的是AutoAdmin[3]的生成方法).

Extend[23]递归式的索引配置构建方法不预先确定一个固定的索引配置搜索空间, 而在每一步根据现有配置和规则确定一个动态的单步搜索空间. 通过what-if优化和最大每单位存储代价缩减确定索引配置的扩充动作, 隐式地考虑现有索引和新索引间的相互影响, 并在索引配置占用存储达到限额后停止.

Relaxation[36]是当前经典的自顶向下索引选择方案. 首先在不考虑存储空间约束情况下, 通过优化器分析工作负载中每个查询的索引请求, 并通过规则生成该查询的最优索引配置. 然后对这些最优索引配置进行并集操作, 形成工作负载的候选索引集合. 最后通过启发式规则选择集合中一个索引配置进行松弛如删除特定索引或2.2节所提到的合并、约简等那些能降低索引配置空间占用的索引变换操作, 并迭代更新最优配置为满足存储空间约束且代价最低的松弛配置, 直到达到调优时间限额后停止. 文献[36]的方法只对被变换索引的相关查询进行代价的重新估计, 并设计了一种根据现有配置查询计划及其代价推导松弛配置下代价上界的方法, 减少了what-if调用次数.

CoPhy[52]是现有基于整数规划枚举方案中最好的方案. 其搜索空间是由变量及约束条件构成的解空间, 只用简单的规则进行剪枝, 并交由整数规划解答器进行可行解的高效枚举. 其配置评价过程使用INUM[58]中的代价推导方法减少what-if调用, 并支持在该规划系统中处理那些能改写为线性不等式的软性约束(soft constraint). 同时, 文献[52]的方案可以随整数规划解答器求解而自然结束, 也可以在可行解质量足够高的情况下提前结束求解, 该决策可自动或引入DBA判断, 减少求解时间, 这得益于现有大多数解答器的性质: 求解效率较高且能够在任意时间点上判断出当前探索到的可行解和最优解之间的距离.

NoDBA[85], LanIdxAdvis[86], MCTS_IT[88]是3种基于强化学习的代表性方案. NoDBA作为一个早期的探索方案, 只考虑单列索引, 且其奖励计算是基于工作负载实际执行时间的缩减, 每个情节中智能体步数与索引数量限额相等. 考虑到效率问题, LanIdxAdvis通过启发式规则生成候选索引, 限制动作空间. 同时在训练过程中使用what-if优化在当前动作所形成的索引配置下估计工作负载代价, 并将相对于前一时刻的工作负载代价的缩减值作为智能体的奖励来源. 为了在探索和利用间平衡, 在智能体策略函数中添加了ε-greedy因子(有ε的随机选择动作的概率), 终止条件为存储空间约束或索引数量限额约束. MCTS_IT采用和AutoAdmin类似的方式生成候选索引(对应动作空间), 并根据索引配置的单调性假设: 一个工作负载在一个索引配置的子集配置下的代价不小于此工作负载在该索引配置下的代价, 推导出一个索引配置下的工作负载代价为其所有子集配置所能达到的最小工作负载代价. 其奖励函数设计主要考虑相对于起始时刻的代价缩减率(代价缩减值与无索引情况下代价的比值), 其智能体策略为了探索和利用的平衡, 添加了简化的ε-greedy因子, 终止条件为索引数量限额.

表1从搜索空间的生成策略、索引配置的评价手段、配置的形成策略、最终索引配置形成过程中的导向指标(即根据什么指标确定添加或删除的索引)、索引调优的终止条件、是否考虑索引间相互影响这6个方面对这些代表性方法的特点进行了展示, 文献[21]也对部分传统方法进行了实验对比分析.

5 动态工作负载下的索引选择问题

5.1 在线索引选择问题

在学术上, 根据处理的工作负载情况是否可变, ISP又可细分为离线ISP和在线ISP. ISP通常默认为离线ISP, 处理的是不变的代表性工作负载. 而实际生产环境中的工作负载可能是动态变化的, 在线ISP

Table 1 Comparison of Representative Methods

表 1 代表性方法对比

方法	搜索空间生成	配置评价	配置形成策略	导向指标	终止条件	索引间影响的考虑
AutoAdmin[3]	规则+优化器	what-if+推导	自底向上	代价缩减	索引数量限额	隐式
DB2Advis[11]	规则+优化器	what-if	自底向上	每单位存储收益	存储空间限额	有限
BEN_KNAP[9]	规则+优化器	what-if+推导	自底向上	每单位存储收益	存储空间限额	显式
Extend[23]	规则	what-if	自底向上	每单位存储代价缩减	存储空间限额	隐式
DROP[39]	完整	外部代价模型	自顶向下	最低代价	最低代价	隐式
Relaxation[36]	优化器	what-if+推导	自顶向下	松弛惩罚	调优时间限额	隐式
CoPhy[52]	规则	what-if+推导	IP	IP	求解结束	隐式
DingIdxAdvis[29]	规则+优化器	机器学习模型	自底向上	代价缩减	索引数量限额	隐式
NoDBA[85]	单列索引	实际执行	自底向上（RL）	代价缩减	索引数量限额	隐式
LanIdxAdvis[86]	规则	what-if	自底向上（RL）	代价缩减+ε-greedy	约束破坏	隐式
MCTS_IT[88]	规则+优化器	what-if+推导	自底向上（RL）	代价缩减+ε-greedy	索引数量限额	隐式

处理的即是这种动态的工作负载，所以离线 ISP 也可看成在线 ISP 的特例，即在一个时间段内的在线 ISP. 现有离线索引选择方案通常需要 DBA 启动集成索引推荐模块的调优工具，然后根据工具返回的提示信息和自身经验做出最终的调优决策. 但现代业务场景越来越复杂多变，这种完全离线的方案变得越来越局限. 为了解决这个问题，在线 ISP 研究如何在动态环境中有效地对数据库索引配置进行调优维护（比如索引的创建、删除、更新等），保证数据库系统的稳定和高效.

综合现有代表性在线索引选择文献 [34, 37, 50−51, 71, 95] 中的描述，本文对在线 ISP 进行如下形式化定义：

定义 2. 给定一个数据库 $D=\{T_1,T_2,\cdots,T_d\}$、一个查询的序列 $(q_1,q_2,\cdots,q_n)$ 以及约束条件集 $B=\{b_1,b_2,\cdots,b_j\}$，找到一个最优的索引配置序列 $S^*=(I_1,I_2,\cdots,I_n)$，使能在满足约束条件集 B 的情况下最小化查询序列的代价和索引配置转变代价的总和，即 $S^*=\arg\min\sum_{i=1}^{n} cost(q_i,I_i)+transition(I_{i-1},I_i)$.

其中 $cost(q_i,I_i)$ 表示查询 q_i 在索引配置 I_i 下的代价，而 $transition(I_{i-1},I_i)$ 表示索引配置从 I_{i-1} 变更为 I_i 的代价（比如创建/删除索引的代价）. 值得注意的是，有的研究工作[34,50-51] 由于应用场景或假设不同，仍以工作负载序列而不是查询序列为单位维护索引配置. 同时，除了离线 ISP 中常见的存储空间约束和调优时间约束外，还可能涉及由于动态工作负载带来的新约束，比如索引配置变更的提升和代价间差值（也称为索引配置变更收益）的阈值约束，即只有索引配置转变的收益与代价间差值达到一定程度才会考虑变更索引配置.

5.2 在线 ISP 的新挑战

在线索引选择处理的是动态工作负载，工作负载的偏移（workload shift）可能导致在原工作负载下推荐的索引配置在新工作负载发生收益衰减，甚至产生负收益. 为了更好地处理动态工作负载，在线索引选择方案可建模为一个在线反馈控制回路[24,37,96-97]，即监控—调优—调整的循环回路，如图 4 所示. 通过监控和观测工作负载和数据库系统指标，对现有数据库系统和不同的候选索引配置进行评价和诊断，得到包含推荐配置的调优建议，然后实际调整现有数据库系统索引配置到推荐配置，完成一次控制反馈循环.

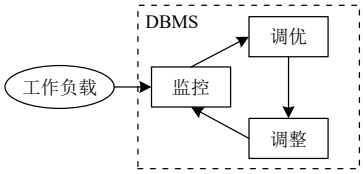


Fig. 4 Online index selection framework based on online feedback control loop

图 4 基于在线反馈控制回路的在线索引选择框架

大多离线索引选择方案的输出仅为调优建议，对应在线反馈控制回路中的调优部分，因此在线 ISP 不仅继承了离线 ISP 的既有挑战，还面临着新的挑战：

1）需要高效的系统监控. 为了保证不会显著影响数据库系统的正常业务运行，系统监控过程需要足够高效，并能及时归集相关数据，使能对系统进行及时的诊断调优工作. 比如，通过将相关监控功能代码嵌入数据库内核（工程方面的挑战）或根据调优算法的需求，提高原始监控数据预处理的能力，比如提

高工作负载表征的准确性和效率.

2)需要高效的索引调优. 由于工作负载的动态变化, 在线索引选择方案需要快速地生成和部署索引调优建议, 避免出现索引调优建议部署完成后所面对的工作负载特征已经发生变化的情况, 造成实际调优效果的下降. 因此在实际设计在线索引调优方案时, 应充分考虑索引调优主体算法的代价和索引配置变更的代价, 比如新增对索引调优算法代价和索引配置变更收益值等目标的考虑, 更多的目标也提高了该多目标优化问题的求解难度.

3)需要高效的索引配置变更. 索引配置变更操作的调度计划(即索引操作的序列)应该足够高效, 特别是考虑到索引间相互影响对索引操作代价影响的情况下. 比如, 如果待删除索引i_1搜索键是待创建索引i_2的搜索键的前缀, 此时可以利用索引i_1降低索引i_2的创建代价, 即在变更部署计划中先创建索引i_2, 再删除索引i_1. 文献[44]将该问题建模为调度(scheduling)问题, 并证明了该问题是NP难的.

值得注意的是, 也有研究[35,98]提出半自动化的在线索引选择方案, 利用DBA提供的调优反馈让调优过程更人为可控[99]. 但我们认为, 这种半自动的方案更适合在特定的探索性数据分析场景下使用. 一方面, 在企业数据管理和业务环境下, 完全自动的方案能进一步降低企业的数据库所有权成本, 在算法和调优机制将变得越来越智能的未来更有前景. 另一方面, DBA在复杂多变的数据库应用场景下提供反馈信号的可靠性也是半自动化方案的一大瓶颈.

5.3 数据库系统的监控

现有研究对在线ISP输入的处理一般可分为2种: 一种是将输入的查询切分为一个一个等时间或等数量的窗口, 一个窗口内查询的集合即形成一个工作负载, 如果不同窗口间工作负载的查询分布发生偏移, 则可能需要进行调整[34]; 另一种则将输入当成以查询为单位的查询流[37,71], 实时地对索引配置进行调整. 但不管使用哪种处理方式, 都离不开对数据库系统的监控.

5.3.1 监控与在线索引选择

动态工作负载, 比如查询需求发生了改变, 可能导致原来输入/生成的代表性工作负载变得不再具有代表性, 原来推荐的索引配置也可能变得不再高效. 为了捕获这种变化, 数据库系统需要持续监控流入的查询语句、索引利用情况以及系统性能等信息.

针对这个问题, 很多数据库厂商都在其数据库产品中实现了这种监控功能. 比如, Oracle的自动工作负载仓库(automatic workload repository, AWR)模块[13]能自动地从内存视图中收集系统性能数据的快照, 用来简单识别工作负载中代价昂贵的查询或传入像ADDM[26]这样的诊断模块进行更复杂的调优工作. 微软则将监控机制集成在SQL Server服务端内部, 提出了一个持续监控的通用框架SQLCM[100], 能以较低代价实现对数据库系统的监控. 结合其事件—条件—动作(event-condition-action, ECA)规则引擎, 可以给像索引选择这样的数据库优化任务提供支持. DB2数据库也实现了其查询巡逻器组件, 能对流入的工作负载进行监控和管理[12]. 在学术界, Thiem等人[101]提出了一种集成式性能监控的思想, 通过将性能监控代码集成到数据库内核中, 尽可能减小细粒度监控给数据库正常运行带来的影响.

5.3.2 工作负载/查询的表征

数据库工作负载或查询的表征反映了数据库系统用户的数据需求, 对代表性工作负载(或查询)或预期工作负载(或查询)的正确认识是数据库系统设计和优化的关键之一. 通过对DBMS的监控, 可以抽取工作负载(查询)相关的原始信息[13,100-101], 比如查询文本、查询相关数据表/列的统计信息、DBMS在各时刻的实时性能信息等. 但这些原始信息过于庞杂, 如果能利用这些原始信息形成工作负载(查询)的有效表征, 则可更好地支持后续数据库的诊断和调优工作. 离线ISP处理的代表性工作负载是静态的, 且对调优时间容忍度较高, 所以工作负载通常被简单表示为查询语句的集合或查询语句-频率对的集合, 但是这种粗粒度的表征方式忽略了查询语句中的特征信息. 合理的工作负载表征能帮助挖掘工作负载中的隐含知识, 比如工作负载的查询分布或工作负载、索引以及数据库性能之间的关系, 还能帮助进行工作负载摘要、查询预测等预处理工作, 提升索引选择方案的效率和效果.

Calzarossa等人[102]对批处理系统、数据库系统、分布式系统中的通用工作负载表征技术进行了综述. 对于数据库工作负载, 可以根据数据库系统的追踪(trace)记录信息, 利用聚类、统计分析等手段实现对工作负载的描述和分类. 很多商业DBMS会默认记录这些信息, 比如Oracle和SQL Server的系统性能快照[26,100]、DB2的追踪记录[12]. 针对关系型数据库系统, Yu等人[103]提出了一个工作负载的分析和表征工具REDWAR, 通过分析工作负载中查询语句的结构、复杂性、查询语句的实际执行以及关系(视图)构成情况, 形成该工作负载的描述性统计信息并进行展示,

帮助 DBA 了解当前工作负载. Elnaffar 等人[104]则通过抽取工作负载特征来构建决策树，判断工作负载的类型是 OLTP 型还是 DSS 型，定性地对工作负载进行描述. 这种定性的、描述性的方式能给数据库系统设计和优化任务提供参考信息，但不能提供精细的建议.

同时，不同任务场景下所需的工作负载表征方法是不同的. 比如在系统资源/容量的规划任务和数据库参数调优这些任务中，一种直接的解决思路就是通过历史工作负载的性能表现信息对工作负载进行隐式(implicit)表征[1,105]. 比如，OtterTune[1]通过分析并选取数据库部分内部运行时指标(internal runtime metric)来向量化工作负载. 但在索引选择问题中，索引的创建依赖数据库模式，索引的采用与查询语句内容、数据库统计信息关系紧密. 通过性能信息隐式表征工作负载的方式很难捕获到这些关系，因此显式地将查询语句的文本特征(比如查询所包含的关系、属性等信息)或/和数据库统计信息等特征与索引收益预期关联的方式可能更适合索引选择问题.

比如，在基于窗口的查询处理方案中，Hammer 等人[38]以当前窗口内的各类统计信息作为工作负载的表征，包括数据更新相关的统计信息(比如当前时间窗口内的数据记录的增加、删除和更新量)、属性选择率的统计信息(和索引的采用相关)以及查询类型相关的统计信息. Schnaitter 等人[34]则将当前窗口内的查询分布作为工作负载的表征. 而对于基于查询流的处理方案，Bruno 等人[37]提出用查询逻辑计划的访问路径请求信息表征工作负载. 而 QB5000[48]将工作负载中的查询抽象为查询模板，然后根据查询模板的历史到达率信息对模板所代表的查询进行表征. Jain 等人[53]则提出用词嵌入(word embedding)和基于长短期记忆(long short-term memory, LSTM)网络的自编码器技术实现查询语句的向量化，并将该技术用于索引调优[106]. Paul 等人[107]通过设计和训练 2 种不同的编码器模型，从查询计划中分别捕获查询的结构与性能信息，综合地对查询语句进行向量化表征.

5.4 索引配置的诊断与调优

通过监控 DBMS 获取工作负载等索引调优所需的信息后，在线索引选择方案需要进一步地分析和判断，解决是否需要对当前索引配置进行调整以及如何进行调整的问题.

5.4.1 反应式与主动式调优

调优阶段的算法与离线索引选择方案类似，输出当前时刻的索引调优建议，该过程需要耗费时间和计算资源. 如果当前 DMBS 和工作负载比较稳定或新索引配置的预期提升不大，则应尽量避免调整配置. 在线索引调优有 2 种处理方式：一种是反应式的(reactive)，另一种则是主动式的(proactive). 反应式调优分析历史信息，检测系统状态变化，并根据这些信息输出索引配置的调整建议；而主动式调优则基于对未来状态的预测结果进行调优，比如根据预测得到的未来工作负载计算索引配置的调整策略.

对于反应式调优方案，重要的是触发条件的确定及对应的处理操作. 触发条件可以是特定长度的时间间隔或特定系统状况的出现，比如工作负载出现了偏移、数据库系统性能出现了明显下降、系统指标超过了预设阈值等. 比如，PDAlerter[37]即是一个轻量级的在线警告器，利用收集到的历史查询执行信息估计对当前工作负载进行调优的收益上下界，帮助判断当前时刻是否需要进行调优. 只有调优收益可观，该工具才会发出调优提示. 这种方式一定程度上可看成离线索引方案在在线 ISP 上的直接扩展，更适合在工作负载较少发生偏移、偶尔需要调优的场景下使用. OnlinePT[71]也是一种反应式调优方案，在查询优化过程中收集候选索引，在查询执行过程中记录和更新每个候选索引的累积收益情况，即文献 [71] 中候选索引的累积收益被定义为过去一时间段内如果索引配置中存在该索引所带来的累积代价缩减. 如果观测到一个候选索引的剩余收益(累积收益和索引创建代价之差)小于或等于 0，则删除这个索引；如果观测到一个候选索引的创建满足存储空间约束，且能带来最大剩余收益，则创建这个索引. 文献 [71] 方案利用索引历史时间段的累积收益情况作为当前时刻的决策依据，这也意味着该方案只有在工作负载较少发生偏移或偏移幅度较小情况下能保证效果和可用性. 此外，Elnaffar 等人[108]提出了一种工作负载偏移检测机制，能判断工作负载类型是否在 DSS 和 OLTP 间发生转变. Holze 等人[109]则提出了一种更通用的工作负载变化检测机制，将工作负载建模为 n-gram 模型[110-111]，根据工作负载的困惑度(perplexity)变化情况来判断工作负载是否发生了偏移. 如果发生了偏移，则进行后续的索引调优操作. 半自动化的索引调优方案 WFIT[98]也属于反应式的调优方案，文献 [98] 定义了索引配置的工作函数(work function)概念，并将其作为调优目标. 然后通过分析流入的查询持续更新各索引配置的工作函数值，并基于这些信息生成调优建议.

而对于主动式调优方案,其重点在于对未来工作负载(表征)的预测. 在得到未来工作负载的预测信息后,即可和不同的索引调优策略结合进行调优. 比如, Hammer 等人[38] 以统计信息表征工作负载,并在每个时间段末尾基于历史工作负载的统计信息用指数平滑法来预测下一时间段工作负载的统计信息,然后利用启发式规则推荐索引. 而 COLT[34] 通过历史窗口的各索引收益情况预测那些索引在未来窗口的收益情况,且随着工作负载中查询的执行不断更新这些索引的收益估计. 最后利用这些索引收益信息,采用背包问题解法进行索引推荐. QB5000[48] 则用查询的到达率信息对查询进行表征,集成线性回归、循环神经网络和核回归模型,对未来工作负载中各查询的到达率进行预测以用于索引推荐.

5.4.2 索引配置的调整代价

索引配置的调整是需要代价的. 离线索引选择方案的运行频率较低,而且索引配置调整代价和工作负载的实际执行代价相比较小,因此通常会忽略索引配置调整代价. 但在线索引选择方案需要更频繁地对索引配置进行调整,不能再忽略索引配置调整代价. 而且只有真实索引(非虚拟索引)才能给查询执行带来实际收益,所以如果一个索引的创建比较耗时,或创建后查询性能提升小于索引创建代价,则没必要推荐这个索引. 比如,在 COLT[34], OnlinePT[71], WFIT[98], SOFT[112] 方案中,索引的收益计算过程便会扣除索引的创建代价. 同时,索引调优算法应避免短时间内频繁创建和删除同一个索引,因为这可能对数据库系统带来负担. 此外,考虑到方案的鲁棒性,避免性能抖动,通常只有当调优建议的预期性能提升与代价间的差值超过自定义阈值[37,98,113] 时,该建议才会被采纳.

5.4.3 调优算法代价

在线索引调优是持续进行的,所以对调优方案的时效性和代价控制的要求更高. 这些方案的代价来源主要分为 3 个部分:监控代价(见 5.3 节)、调优算法代价和索引配置调整的代价(见 5.4.2 节). 本节将对如何降低调优算法代价的问题进行讨论.

无论是离线 ISP 还是在线 ISP,其调优算法代价的主要来源都是 what-if 优化操作. 离线索引选择方案中用到的减少工作负载大小(比如工作负载压缩)、原子配置概念等方法均可被用于在线索引选择方案. 而针对在线 ISP 的动态特点和持续、重复的调优需求,也有一些在线索引选择方案设计了减少 what-if 优化过程代价的新机制. 比如, COLT[34] 将候选索引集合分为 3 部分:第 1 部分被称为物化索引集,包含所有已在数据库系统中被物化的索引;第 2 部分被称为热索引集,包含那些虽还未被物化但有潜力的索引;第 3 部分被称为普通索引集,包含所有不属于物化索引集和热索引集的索引. 考虑到用 what-if 优化获取所有候选索引收益的昂贵代价, COLT 设计了一种 2 阶段策略来维护更新索引的收益值,其中采用不同的方法对不同种类索引的收益进行更新. 在第 1 阶段, COLT 利用外部代价模型生成粗粒度的索引收益统计信息,并根据这些信息对候选索引进行排序. 一部分排名在前但还未被物化的候选索引会被加入热索引集. 在第 2 阶段,因为物化索引集和热索引集中的索引最终被推荐的概率更大,为了更准确有效地进行调优, COLT 会用更准确但更昂贵的 what-if 优化操作来收集和更新这 2 个集合中候选索引的收益统计信息,避免对所有候选索引配置进行 what-if 优化. OnlinePT[71] 则利用文献 [36–37] 中查询计划的局部变换操作,在不进行 what-if 调用情况下收集候选索引的收益相关信息,用于在线索引调优.

5.4.4 基于强化学习的在线索引选择

在线 ISP 处理的对象是窗口化的工作负载或查询流,所以其强化学习建模与离线情况强化学习建模的主要区别在于状态和奖励函数的设计,而重点在于模型对动态工作负载的适应能力. 为了让智能体更好地适应动态环境、降低重新训练的需要,状态建模中应包含所有能帮助进行索引选择的重要信息,使智能体在面对不同工作负载时,仍能输出合理的调优建议. 比如利用基于学习的工作负载表征[49,51] 让智能体能捕捉查询中影响索引选择策略的特征,使智能体训练后能更容易适应不同的工作负载/查询.

比如, DBA bandits[49] 将在线 ISP 建模为多臂老虎机(multi-arm bandits, MAB)问题[90]. 其中每个手臂相当于一个索引配置,而每个手臂的评分用来评价该索引配置的好坏. 面对每个查询模板,该方法选取每个手臂包含的索引键前缀及其统计信息(比如是否是覆盖索引、当前索引配置与数据库的存储空间比值、动作臂的历史选取频率)作为当前手臂的上下文(context)信息,并假设一个手臂的评分和其上下文信息存在线性关系,然后通过强化学习的训练量化该线性关系. 在实际决策过程中,智能体选择当前状态下评分最大的手臂所代表的索引配置用于推荐. 文献 [49] 从实验上验证了该方法对动态工作负载的适应能力,但要求未来查询和现有查询具有一定的相似性,不能完全不同.

而 SWIRL[50] 是一个基于 PPO 的在线索引选择方案. 其智能体的状态表示包含 3 种信息, 即源信息(比如存储空间限制值这样的约束描述信息)、当前状态的索引配置信息以及工作负载信息(比如每个代表性查询语句的频率、每个代表性查询语句在当前配置下的执行时间估计等). SWIRL 将代表性查询语句的计划以操作符为中心进行向量化, 同时利用潜在语义索引(latent semantic indexing, LSI)技术[114] 降低特征向量的维度, 形成最终的工作负载表征. SWIRL 可以帮助捕捉工作负载的语义信息, 让智能体更容易适应动态的工作负载环境.

针对非共享数据库集群上的在线索引选择问题, Sadri 等人[15] 提出了一种基于 DQN 的方案 DRLinda, 通过同时考虑查询代价和集群负载均衡, 让智能体学习在面对一个查询时如何选择一个特定数据库副本并建立特定的索引配置的策略.

此外, Basu 等人[84] 提出了一种基于强化学习的数据库调优框架, 并以索引调优为例进行讨论. 该框架以查询为基本单位, 不需要提供代价模型, 同时对代价模型和索引选择策略的值函数进行学习. 但该方案假设查询特征和代价估计、值函数之间都是线性关系, 且没有考虑索引间的相互影响, 同时其采用的传统强化学习算法也很难处理巨大的候选索引数量.

5.4.5 调优算法对比

传统在线索引选择算法中, OnlinePT[71] 和 COLT[34] 分别是反应式调优方法和主动式调优方法的代表. Schnaitter 等人[95] 也针对在线 ISP 提出了一个基准测试方法, 用工作负载处理时间和索引配置变更时间之和以及工作负载处理的时钟时间(wall-clock time) 2 个指标来评价方案的优劣, 并对 COLT 和 OnlinePT 的表现进行了对比. 总的来说, OnlinePT 在工作负载变化快和变化幅度大的情况下效果比 COLT 更稳定和更好, 因为在这种情况下 COLT 这种主动式调优方法对未来工作负载情况的预测可能出现偏差. 基于强化学习的在线索引选择方案[15,49-50,84] 属于反应式调优, 通过不断试错、学习和改进, 学习面对各种查询/工作负载的灵活策略. 这些方法理论上是可行且有潜力的, 但仍属于探索性工作, 也没有和传统在线索引选择方案(比如 COLT 和 OnlinePT)相互之间进行实验的对比分析.

5.5 索引配置的调整部署

索引配置的调整只有真实反映到数据库系统中才能真正对查询执行造成影响, 而合理的索引配置调整操作顺序能提高调优建议部署的效率. 比如, 如果索引配置的调整包含创建索引 $i_1(a)$ 和删除索引 $i_2(a,b)$ 这 2 个操作, 我们可能倾向于先创建索引 i_1, 后删除索引 i_2, 因为索引 i_2 中已包含按照属性 a 排序的索引数据, 因此可加速索引 i_1 的创建.

虽然这个问题在离线 ISP 中也存在, 但由于离线情况下调优频率相对较低、影响不大, 故离线索引选择方案中很少考虑这个问题. 而现有在线索引选择方案还主要集中在解决获取合理的索引配置调整建议问题上, 因此这方面研究也较少. 为了降低索引配置调整的代价, 避免性能抖动, Luhring 等人[112] 和 Sattler 等人[33] 在进行索引配置调整时都使用了一种延时索引(deferred index)机制, 即先在数据库注册索引信息但不实际创建索引, 实际创建操作会选择在有查询对该索引所在关系表进行全表扫描时进行, 通过共享扫描过程, 减少显式创建索引的代价.

Agrawal 等人[115] 首先意识到查询顺序对索引调优的重要性, 将工作负载看成一个查询序列, 并将索引的创建和删除操作合理地插入工作负载查询处理的序列中, 最小化工作负载的执行时间. 该方法将索引推荐和索引配置的调整融合成一个过程, 但假定工作负载是已知的, 更适合离线索引选择场景. Bruno 等人[44] 则首次提出物理设计调度(physical design scheduling)问题的概念和形式化定义, 并以索引配置的调整为例进行讨论, 但没有形成具体的方法. Kimura 等人[116] 针对索引配置调整问题进行了系统的研究, 对索引间相互影响进行分类和描述, 并基于分类信息和基于约束规划(constraint programming)的数学模型对这个问题进行形式化定义和求解.

5.6 小 结

面对动态的工作负载, ISP 也呈现出新的挑战. 本节综合现有代表性在线索引选择文献中的描述, 对在线 ISP 进行了形式化定义. 然后基于在线反馈控制回路框架归纳和总结了相关的研究内容和方法. 其中, 数据库系统的监控模块为调优模块提供输入的查询/工作负载或预处理后的监控数据(比如通过手动设计或表示学习得到的特征). 调优模块解决是否需要对当前索引配置进行调整以及如何进行调整的问题. 从形式上看, 可以选择反应式或主动式的调优方式, 反应式调优的效果取决于历史和未来的工作负载间、DBMS 状态间的相似性, 而主动式调优的效果则取决于对未来的工作负载和 DBMS 状态的预测准确性. 在工作负载变化快和变化幅度大的情况下, 由于工作负载预测难度的提升, 反应式调优方案效果通常相对更好. 而从细节上看, 为了提高调优模

块的效率效果，需要综合考虑索引配置的调整代价和调优算法的代价. 合理的索引配置调整操作顺序能提高索引配置的变更效率，该问题也是 NP 难的，但现有研究较少. 我们认为，未来对该问题进行系统性研究的潜力较大.

6 索引调优工具的发展与现状

6.1 索引调优工具

在工业界，商业数据库厂商为了提升产品竞争力，通常会在产品中集成相关调优工具(GUI 或命令行工具)辅助 DBA 进行调优. 商业数据库厂商很早就开始研究数据库的物理设计调优(包括索引调优)，并将研究成果以调优工具的形式集成在其产品中. 比如，微软在 1997 年便开始布局和推进该方向[3,25,117]，并在 2001 年正式成立 AutoAdmin 项目[118]，并延续至今，专注研究数据库的管理和调优，很多研究成果也被集成在 SQL Server 数据库调优顾问中. 同时期，IBM 也在 DB2 上集成了 db2advis 命令和可视化的 DB2 设计顾问[119]，帮助 DBA 进行索引调优，并逐步形成对物化查询表、多维聚簇表转换等其他物理设计的调优支持；而 Oracle 则在其数据库中实现了 SQL 访问路径顾问，支持对索引、物化视图、分区等物理设计的调优.

随着开源数据库的兴起和流行，也有开发者和公司为开源数据库实现索引调优工具，扩展开源数据库的功能，提升开源数据库的可用性. 比如，Percona 工具箱中的 pt-index-usage 命令[120]就可被用来分析 MySQL 的索引使用情况，输出不常用索引的删除建议. 开源插件 dexter[121]，PoWA[122] 和商业的 pganalyze[123]，EDB Postgres[124] 在 PostgreSQL 上实现了索引推荐功能. 还有第三方工具支持对多种数据库的调优，比如 EverSQL[125] 支持对 Oracle，SQL Server，PostgreSQL，MySQL 等主流数据库的索引调优.

虽然支持开源数据库的索引调优工具越来越多，但总的来说，商业数据库的调优工具起步更早，功能更加完善，且有相关论文[3,9-13,16,20,25-26,54,71,117-118] 支撑.

6.2 自动化的调优方案

随着企业数据管理分析的场景日趋复杂多变，数据库调优工具也逐渐向更自动化、智能化、自治的(autonomous)方向发展. Oracle 在现有数据库产品[126]中同时集成了主动式调优和反应式调优的能力，通过持续的系统监控，自动捕捉和收集工作负载及相关执行环境信息等源信息，支撑 SQL 访问顾问生成调优建议及其预期收益的估计，极大降低了 DBA 的调优负担. 微软最新的 SQL Server 则集成了自动调优(automatic tuning)[127] 特性，通过分析查询性能问题的潜在原因，自动推荐解决方案进行修复，比如自动进行索引调优、自动纠正可能有问题的计划等. 且会持续对数据库性能进行监控，并对解决方案进行验证，即如果性能出现下降，则自动恢复到最近的最佳状态. 开源数据库 OpenGauss 在其 DBMind 模式中也集成了一些机器学习驱动的数据库技术，其中包括索引推荐功能[128]. OtterTune 作为一个第三方工具，集成了很多机器学习算法，支持在其面板查看索引健康度的指标[129]，并提供索引调优建议.

随着云数据库的发展，很多企业将部分数据库调优工作外包给云数据库厂商. 为了服务大量不同场景下的客户，云数据库厂商更需要自动化、智能化的索引调优服务. 比如，阿里云便为其云数据库产品提供了数据库自治服务[130]. 同时，传统数据库厂商也开始提供云数据库产品，比如微软的 Azure SQL 数据库[131]、Oracle 的自治数据库 Exadata[132]、IBM 的云数据库系列[133] 也将其在传统数据库产品中成熟的索引调优功能搬上云端，比如，Oracle 的自治数据库便基于 Oracle 数据库及其 Exadata 云数据库提出，以应对各种场景不同复杂度和不同规模工作负载下的索引调优任务.

7 总结与展望

7.1 总 结

索引是数据库系统重要的物理数据结构之一，索引调优也是数据库物理设计的重要组成部分. 一个合适的索引配置能给数据库性能带来极大的提升，这也让 ISP 成为数据库优化的经典问题之一. 本文首先对离线 ISP 进行定义，并把现有解决方案归结为一个包含索引配置搜索空间生成、索引配置评价及索引配置的枚举与搜索三大部分的调优范式，然后对每部分的相关研究进行了归纳、总结和对比.

索引配置搜索空间的生成阶段主要通过启发式规则和优化器来生成候选索引和索引配置，形成索引配置的搜索空间. 作为输入，索引配置搜索空间的大小极大影响着索引选择方案的代价和效率. 但是过于激进的空间剪枝策略反而可能错误地排除掉有益的索引，影响索引调优方案的效果，所以启发式规则的设计应该在效率和效果之间取得平衡.

为了评价索引配置，一种常用思路是通过工作

负载在各索引配置下代价的数值来评价，相关研究大多通过优化器的 what-if 优化技术进行估计. 另一种思路则是通过构建索引配置间的偏序关系来比较各索引配置. 比如，训练一个机器学习分类器模型对索引间的偏序关系进行预测，然后在搜索过程中不断寻找更好的索引配置.

在索引配置的枚举与搜索策略中，自底向上的枚举策略简单直观，效率相对较高，效果和具体的枚举策略有关，在存储空间限制较小时更适用. 自顶向下的枚举策略从不考虑存储约束的初始配置，开始迭代地对索引配置进行松弛，通常在索引配置存储限额较大时效果较好，但运行时间也更长. 而基于整数规划的隐枚举策略侧重在索引推荐的最优性保证上，更难考虑到索引间的相互影响. 且在面对大规模数据库系统和工作负载时，这种策略可用性下降. 基于强化学习的搜索策略通过训练一个智能体，学习一个灵活的智能策略函数以生成索引调优建议是一个有前景的方向.

面对动态工作负载下的 ISP，我们依照在线反馈控制回路[24]框架，分别对数据库系统的监控、索引配置的诊断与调优以及索引配置的部署调整 3 方面的相关方法进行归纳和总结. 面对动态工作负载，数据库系统的监控是整个在线反馈控制回路框架的入口，监控过程收集到的各种数据被处理后发送给诊断与调优模块进行分析，形成调优建议. 该模块中的调优算法即对应离线索引选择方案从工作负载输入到调优建议输出的过程，反应式地或主动式地生成索引调优建议. 最后根据该建议规划配置调整的计划进行部署，形成在线反馈控制回路的闭环.

7.2 展　望

在近几年的数据库领域顶级会议上，研究者们开始探索使用机器学习技术来增强或替换数据库系统的优化器，比如，Kraska 等人[134]提出一种学习型的数据库系统框架 SageDB，Marcus 等人[135]则提出了一个学习型优化器 Bao. 在索引调优问题上：1）越来越多的研究工作开始使用机器学习技术来改进索引选择方案. 比如，DingIndexAdvis[29]通过学习一个分类器来对索引配置进行比较和评价，替代 AutoAdmin[3]中基于 what-if 优化的索引配置评价方式. 2）索引调优方案所考虑的要素也越来越精细和实际. 比如，Siddiqui 等人[32]提出一种基于学习的新型工作负载压缩算法 ISUM 以提升索引选择方案的效率. Siddiqui 等人[67]还提出一个新型索引调优原型系统 DISTILL，设计了一个过滤冗余候选索引的过滤器和一个高效的学习型代价估计模型以提升索引选择方案的效率和可伸缩性. 而 Wu 等人[88]针对 what-if 优化的代价问题提出一种能在 what-if 调用次数预算内进行索引选择的方案. 3）越来越多变的应用场景也让学术界更加关注更实际的在线 ISP，特别是在索引选择过程中对动态工作负载的处理. 比如，AutoIndex[51]针对索引配置的管理问题，提出利用蒙特卡洛树搜索增量更新索引配置的方法，保证未来新工作负载的高效执行效率. 而 SWIRL[50]通过抽取查询或查询计划的特征并经过潜在语义模型进行向量化，利用机器学习的泛化能力实现对动态工作负载的适应.

相比传统索引选择方案，现有基于机器学习的索引选择方案仍不多，有较大的研究空间. 我们认为，ISP 在 6 个方向有较大的研究潜力，为研究者提供思路和参考：

1）集成基于学习的代价评估方法的索引选择方案. 现有研究中基于优化器的代价估计方法是不准确的，可能导致调优性能的退化[29]. 而当前面向 ISP 的基于学习的代价评估方法较少，具有较大研究潜力. 其挑战包括如何提高机器学习模型的训练效率以及如何保证代价预测模型在不同情形下的稳定性（面对不同工作负载和索引配置的组合时）和泛化能力（比如对动态工作负载的适应能力）等.

2）集成基于学习的索引配置比较方法的索引选择方案. 和利用工作负载代价值找到最优索引配置的方式不同，基于学习的索引配置比较方法关注索引配置间的偏序关系，由于现有研究较少，具有较大的研究潜力. 其挑战在于对索引配置和工作负载的有效表征，使模型能准确地对索引配置进行比较.

3）基于强化学习的索引选择方案. 基于强化学习的方案能从数据和与 DBMS 的交互过程中学习出灵活的索引配置搜索策略. 基于强化学习的索引选择方案越来越多，但仍存在很多挑战. 比如如何合理地将索引选择问题的挑战内化到强化学习建模中，以及如何让智能体适应动态的工作负载等.

4）索引调优建议的最优调整部署策略. 该问题是 NP 难的，现有相关研究较少，但却影响索引调优方案的效率（特别是在线场景下）. 该问题可被看成是一个调度问题，且在数据库索引间存在相互影响的上下文环境下变得更具挑战性，值得进行更深入的研究.

5）考虑索引类型的索引调优. 经典的 ISP 通常不考虑具体的索引类型，仅从逻辑上选择特定的属性列组成合适的索引并构建索引配置，通常也默认使

用B+树进行研究和实验.但在实际调优工具落地过程中,索引类型也是需要考虑的因素,比如,哈希索引在处理点查询的时候通常是存在优势的.如果考虑索引类型,那么ISP的候选索引配置空间将成倍增长,对索引选择方案的伸缩性也提出了更高的要求.作为另一种选择,新的学习型索引(learned index)用机器学习模型模拟索引从搜索键到实际记录位置的映射,弱化了传统索引结构的差别.但这些研究通常集中在内存数据库领域,如果能够将学习型索引技术迁移到磁盘数据库中,那么将有利于消除索引调优方案中对索引类型这个因素的考虑,同时也能提升学习型索引的应用面.这个方向已有少量研究[136-137],仍具有较大潜力.

6)现有索引选择研究主要集中在单机数据库上,但分布式数据库和云数据库的迅猛发展为索引调优引入了新的挑战.比如:在分布式数据库中,如何选择在哪个数据库副本上建立怎样的索引;在云数据库中,如何利用云数据库灵活的可扩展性和计费方式,经济地进行索引选择,以及是否应该提高索引的存储空间限额以提升数据库的查询处理性能等.

作者贡献声明: 赖思超负责调研、论文撰写和修改;吴小莹指导论文写作、审阅和修改;彭煜玮、彭智勇提出论文指导意见.

参 考 文 献

[1] Van Aken D, Pavlo A, Gordon G J, et al. Automatic database management system tuning through large-scale machine learning [C] //Proc of the 2017 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2017: 1009–1024

[2] Cui Yuesheng, Zhang Yong, Zeng Chun, et al. Database physical structure optimization technology[J]. Journal of Software, 2013, 24(4): 761−780 (in Chinese)
(崔跃生,张勇,曾春,等.数据库物理结构优化技术[J].软件学报,2013,24(4):761−780)

[3] Chaudhuri S, Narasayya V R. An efficient cost-driven index selection tool for Microsoft SQL Server [C] //Proc of the 23rd Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 1997: 146–155

[4] Lum V Y, Ling H. An optimization problem on the selection of secondary keys [C] //Proc of the 26th Annual Conf. New York: ACM, 1971: 349–356

[5] Bayer R, McCreight E. Organization and maintenance of large ordered indices [C] //Proc of the 1970 ACM SIGFIDET Workshop on Data Description, Access and Control. New York: ACM, 1970: 107–141

[6] Lum V Y. Multi-attribute retrieval with combined indexes[J]. Communications of the ACM, 1970, 13(11): 660−665

[7] Stonebraker M. The choice of partial inversions and combined indices[J]. International Journal of Computer & Information Sciences, 1974, 3(2): 167−188

[8] Comer D. The difficulty of optimum index selection[J]. ACM Transactions on Database Systems, 1978, 3(4): 440−445

[9] Chaudhuri S, Datar M, Narasayya V. Index selection for databases: A hardness study and a principled heuristic solution[J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(11): 1313−1323

[10] Agrawal S, Chaudhuri S, Kollar L, et al. Database tuning advisor for Microsoft SQL Server 2005 [C] //Proc of the 30th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2004: 1110–1121

[11] Valentin G, Zuliani M, Zilio D C, et al. DB2Advisor: An optimizer smart enough to recommend its own indexes [C] //Proc of the 16th Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2000: 101–110

[12] Zilio D C, Rao Jun, Lightstone S, et al. DB2 design advisor: Integrated automatic physical database design [C] //Proc of the 30th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2004: 1087–1097

[13] Dageville B, Das D, Dias K, et al. Automatic SQL tuning in Oracle 10g [C] //Proc of the 30th Int Conf on Very Large Data bases. San Francisco, CA: Morgan Kaufmann, 2004: 1098–1109

[14] Li Guoliang, Zhou Xuanhe, Sun Ji, et. al. A survey of machine learning based database techniques[J]. Chinese Journal of Computers, 2020, 43(11): 2019−2049 (in Chinese)
(李国良,周煊赫,孙佶,等.基于机器学习的数据库技术综述[J].计算机学报,2020,43(11):2019−2049)

[15] Sadri Z, Gruenwald L, Lead E. DRLindex: Deep reinforcement learning index advisor for a cluster database [C/OL] //Proc of the 24th Symp on Int Database Engineering & Applications. New York: ACM, 2020[2022-09-01].https://dl.acm.org/doi/10.1145/3410566.3410603

[16] Das S, Grbic M, Ilic I, et al. Automatically indexing millions of databases in Microsoft Azure SQL database [C] //Proc of the 2019 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2019: 666–679

[17] Idreos S, Kersten M L, Manegold S. Database cracking [C/OL] //Proc of the 3rd Biennial Conf on Innovative Data Systems Research. 2007[2020-09-01].https://www.cidrdb.org/cidr2007/

[18] Graefe G, Kuno H. Adaptive indexing for relational keys [C] //Proc of the 26th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2010: 69–74

[19] Graefe G, Idreos S, Kuno H, et al. Benchmarking adaptive indexing [G]//LNPSE 6417: Proc of the 2nd Technology Conf on Performance Evaluation and Benchmarking. Berlin: Springer, 2011: 169–184

[20] Bruno N. Automated Physical Database Design and Tuning [M]. 1st ed. Boca Raton, FL: CRC Press, 2011

[21] Kossmann J, Halfpap S, Jankrift M, et al. Magic mirror in my hand,

which is the best in the land? An experimental evaluation of index selection algorithms[J]. Proceedings of the VLDB Endowment, 2020, 13(11): 2382−2395

[22] Faerber F, Kemper A, Larson P Å, et al. Main memory database systems[J]. Foundations and Trends in Databases, 2017, 8(1/2): 1−130

[23] Schlosser R, Kossmann J, Boissier M. Efficient scalable multi-attribute index selection using recursive strategies [C] //Proc of the 35th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2019: 1238–1249

[24] Weikum G, Moenkeberg A, Hasse C, et al. Self-tuning database technology and information services: From wishful thinking to viable engineering [C] //Proc of the 28th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2002: 20–31

[25] Chaudhuri S, Narasayya V. AutoAdmin "what-if" index analysis utility [C] //Proc of the 1998 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 1998: 367–378

[26] Dias K, Ramacher M, Shaft U, et al. Automatic performance diagnosis and tuning in Oracle [C/OL] //Proc of the 2nd Biennial Conf on Innovative Data Systems Research. 2005[2022-09-01].https://www.cidrdb.org/cidr2005/

[27] Chaudhuri S, Narasayya V, Weikum G. Database tuning using combinatorial search [M]//Encyclopedia of Database Systems. New York: Springer Press, 2018: 985–989

[28] Bruno N, Nehme R V. Configuration-parametric query optimization for physical design tuning [C] //Proc of the 2008 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2008: 941−952

[29] Ding Bailu, Das S, Marcus R, et al. AI meets AI: Leveraging query executions to improve index recommendations [C] //Proc of the 2019 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2019: 1241–1258

[30] Schnaitter K, Polyzotis N, Getoor L. Index interactions in physical design tuning: Modeling, analysis, and applications[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 1234−1245

[31] Finkelstein S, Schkolnick M, Tiberio P. Physical database design for relational databases[J]. ACM Transactions on Database Systems, 1988, 13(1): 91−128

[32] Siddiqui T, Jo S, Wu Wentao, et al. ISUM: Efficiently compressing large and complex workloads for scalable index tuning [C] //Proc of the 2022 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2022: 660–673

[33] Sattler K-U, Schallehn E, Geist I. Autonomous query-driven index tuning [C] //Proc of the 8th Int Database Engineering and Applications Symp. Los Alamitos, CA: IEEE Computer Society, 2004: 439–448

[34] Schnaitter K, Abiteboul S, Milo T, et al. On-line index selection for shifting workloads [C] //Proc of the 23rd Int Conf on Data Engineering Workshop. Piscataway, NJ: IEEE, 2007: 459–468

[35] Jimenez I, Sanchez H, Tran Q T, et al. Kaizen: A semi-automatic index advisor [C] //Proc of the 2012 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2012: 685−688

[36] Bruno N, Chaudhuri S. Automatic physical database tuning: A relaxation-based approach [C] //Proc of the 2005 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2005: 227–238

[37] Bruno N, Chaudhuri S. To tune or not to tune? A lightweight physical design alerter [C] //Proc of the 32nd Int Conf on Very Large Data Bases. New York: ACM, 2006: 499–510

[38] Hammer M, Chan A. Index selection in a self-adaptive data base management system [C] //Proc of the 1976 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 1976: 1–8

[39] Whang K Y. Index selection in relational databases [C] //Proc of the 2nd Int Conf on Foundations of Data Organization. New York: Plenum, 1985: 487–500

[40] Schkolnick M. The optimal selection of secondary indices for files[J]. Information Systems, 1975, 1(4): 141−146

[41] Ip M Y L, Saxton L V, Raghavan V V. On the selection of an optimal set of indexes[J]. IEEE Transactions on Software Engineering, 1983, SE-9(2): 135−143

[42] Chaudhuri S, Narasayya V. Index merging [C] //Proc of the 15th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 1999: 296–303

[43] Nehme R, Bruno N. Automated partitioning design in parallel database systems [C] //Proc of the 2011 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2011: 1137–1148

[44] Bruno N, Chaudhuri S. Physical design refinement: The 'merge-reduce' approach[J]. ACM Transactions on Database Systems, 2007, 32(4): 28−71

[45] Deep S, Gruenheid A, Koutris P, et al. Comprehensive and efficient workload compression[J]. Proceedings of the VLDB Endowment, 2020, 14(3): 418−430

[46] Chaudhuri S, Gupta A K, Narasayya V. Compressing SQL workloads [C] //Proc of the 2002 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2002: 488–499

[47] Kołaczkowski P. Compressing very large database workloads for continuous online index selection [G]//LNISA 5181: Proc of the 19th Int Conf on Database and Expert Systems Applications. Berlin: Springer, 2008: 791–799

[48] Ma Lin, Van Aken D, Hefny A, et al. Query-based workload forecasting for self-driving database management systems [C] //Proc of the 2018 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2018: 631–645

[49] Perera R M, Oetomo B, Rubinstein B I P, et al. DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees [C] //Proc of the 37th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2021: 600–611

[50] Kossmann J, Kastius A, Schlosser R. SWIRL: Selection of workload-aware indexes using reinforcement learning [C/OL] //Proc of the 25th Int Conf on Extending Database Technology. 2022[2022-12-01].https://openproceedings.org/2022/conf/edbt/

[51] Zhou Xuanhe, Liu Luyang, Li Wenbo, et al. AutoIndex: An incremental index management system for dynamic workloads [C] //Proc of the 38th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2022: 2196–2208

[52] Dash D, Polyzotis N, Ailamaki A. CoPhy: A scalable, portable, and interactive index advisor for large workloads[J]. Proceedings of the VLDB Endowment, 2011, 4(6): 362−372

[53] Jain S, Howe B, Yan Jiaqi, et al. Query2Vec: An evaluation of NLP

techniques for generalized workload analytics [J]. arXiv preprint, arXiv: 1801.05613, 2018

[54] Chaudhuri S, Ganesan P, Narasayya V R. Primitives for workload summarization and implications for SQL [C] //Proc of the 29th Int Conf on Very Large Data Bases. San Diego, CA: Morgan Kaufmann, 2003: 730–741

[55] Kul G, Luong D, Xie T, et al. Ettu: Analyzing query intents in corporate databases [C] //Proc of the 25th Int Conf Companion on World Wide Web. Geneva: Int World Wide Web Conf Steering Committee, 2016: 463−466

[56] Whang K Y, Wiederhold, Sagalowicz. Separability—An approach to physical database design[J]. IEEE Transactions on Computers, 1984, 33(3): 209−222

[57] Choenni S, Blanken H M, Chang T. On the selection of secondary indices in relational databases[J]. Data & Knowledge Engineering, 1993, 11(3): 207−233

[58] Papadomanolakis S, Dash D, Ailamaki A. Efficient use of the query optimizer for automated physical design [C] //Proc of the 33rd Int Conf on Very Large Data Bases. New York: ACM, 2007: 1093–1104

[59] Papadomanolakis S, Ailamaki A. An integer linear programming approach to database design [C] //Proc of the 23rd Int Conf on Data Engineering Workshop. Piscataway, NJ: IEEE, 2007: 442–449

[60] Chaudhuri S, Narasayya V. Anytime algorithm of database tuning advisor for Microsoft SQL Server [EB/OL]. [2020–11–11]. https://www.microsoft.com/en-us/research/publication/anytime-algorithm-of-database-tuning-advisor-for-microsoft-sql-server/

[61] Konig A C, Nabar S U. Scalable exploration of physical database design [C] //Proc of the 22nd Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2006: 37–37

[62] Meng Xiaofeng, Ma Chaohong, Yang Chen. Survey on machine learning for database systems[J]. Journal of Computer Research and Development, 2019, 56(9): 1803−1820 (in Chinese)
(孟小峰, 马超红, 杨晨. 机器学习化数据库系统研究综述 [J]. 计算机研究与发展, 2019, 56(9): 1803−1820)

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

[64] Wang Xiaoying, Qu Changbo, Wu Weiyuan, et al. Are we ready for learned cardinality estimation?[J]. Proceedings of the VLDB Endowment, 2021, 14(9): 1640−1654

[65] Kipf A, Kipf T, Radke B, et al. Learned cardinalities: Estimating correlated joins with deep learning [C/OL] //Proc of the 9th Biennial Conf on Innovative Data Systems Research. 2019[2022-09-01]. https://www.cidrdb.org/cidr2019/

[66] Sun Ji, Li Guoliang. An end-to-end learning-based cost estimator[J]. Proceedings of the VLDB Endowment, 2019, 13(3): 307−319

[67] Siddiqui T, Wu Wentao, Narasayya V, et al. DISTILL: Low-overhead data-driven techniques for filtering and costing indexes for scalable index tuning[J]. Proceedings of the VLDB Endowment, 2022, 15(10): 2019−2031

[68] Gao Jianling, Zhao Nan, Wang Ning, et al. Automatic index selection with learned cost estimator[J]. Information Sciences, 2022, 612: 706−723

[69] Yuan Haitao, Li Guoliang, Feng Ling, et al. Automatic view generation with deep learning and reinforcement learning [C] //Proc of the 36th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2020: 1501–1512

[70] Kimura H, Huo G, Rasin A, et al. CORADD: Correlation aware database designer for materialized views and indexes[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 1103−1113

[71] Bruno N, Chaudhuri S. An online approach to physical design tuning [C] //Proc of the 23rd Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2007: 826–835

[72] Bruno N, Chaudhuri S. Constrained physical design tuning[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 4−15

[73] Caprara A, Fischetti M, Maio D. Exact and approximate algorithms for the index selection problem in physical database design[J]. IEEE Transactions on Knowledge and Data Engineering, 1995, 7(6): 955−967

[74] Frank M R, Omiecinski E, Navathe S B. Adaptive and automated index selection in RDBMS [G]//LNCS 580: Proc of the 3rd Int Conf on Extending Database Technology. Berlin: Springer, 1992: 277–292

[75] Gurobi. Gurobi optimizer [EB/OL]. [2022-09-01].https://www.gurobi.com

[76] IBM. IBM ILOG CPLEX optimizer [EB/OL]. [2022-09-01].https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[77] Li Guoliang, Zhou Xuanhe, Li Shifu, et al. QTune: A query-aware database tuning system with deep reinforcement learning[J]. Proceedings of the VLDB Endowment, 2019, 12(12): 2118−2130

[78] Zhang Ji, Zhou Ke, Li Guoliang, et al. CDBTune+: An efficient deep reinforcement learning-based automatic cloud database tuning system[J]. The VLDB Journal, 2021, 30(6): 959−987

[79] Hilprecht B, Binnig C, Roehm U. Learning a partitioning advisor with deep reinforcement learning [J]. arXiv preprint, arXiv: 1904.01279, 2019

[80] Heitz J, Stockinger K. Join query optimization with deep reinforcement learning algorithms [J]. arXiv preprint, arXiv: 1911.11689, 2019

[81] Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning [J]. arXiv preprint, arXiv: 1808.03196, 2019

[82] Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration [C/OL] //Proc of the 1st Int Workshop on Exploiting Artificial Intelligence Techniques for Data Management. New York: ACM, 2018[2020-09-01].https://dl.acm.org/doi/10.1145/3211954.3211957

[83] Liang Xi, Elmore A J, Krishnan S. Opportunistic view materialization with deep reinforcement learning [J]. arXiv preprint, arXiv: 1903.01363, 2019

[84] Basu D, Lin Qian, Chen Weidong, et al. Regularized cost-model oblivious database tuning with reinforcement learning [J]. Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII. Berlin: Springer, 2016: 96–132

[85] Sharma A, Schuhknecht F M, Dittrich J. The case for automatic database administration using deep reinforcement learning [J]. arXiv preprint, arXiv: 1801.05643, 2018

[86] Lan Hai, Bao Zhifeng, Peng Yuwei. An index advisor using deep reinforcement learning [C] //Proc of the 29th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2020: 2105–2108

[87] Lai Sichao, Wu Xiaoying, Wang Senyang, et al. Learning an index advisor with deep reinforcement learning [G] // LNISA 12859: Proc of the 5th APWeb and WAIM Joint Int Conf on Web and Big Data. Berlin: Springer, 2021: 178–185

[88] Wu Wentao, Wang Chi, Siddiqui T, et al. Budget-aware index tuning with reinforcement learning [C] //Proc of the 2022 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2022: 1528–1541

[89] Schaarschmidt M, Kuhnle A, Ellis B, et al. LIFT: Reinforcement learning in computer systems by learning from demonstrations [J]. arXiv preprint, arXiv: 1808.07903, 2018

[90] Sutton R S, Barto A G. Reinforcement Learning: An Introduction [M]. 2nd ed. Cambridge, MA: MIT Press, 2018

[91] Licks G P, Couto J C, Miehe P F, et al. SmartIX: A database indexing agent based on reinforcement learning[J]. Applied Intelligence, 2020, 50(8): 2575−2588

[92] Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with deep reinforcement learning [J]. arXiv preprint, arXiv: 1312.5602, 2013

[93] TPC. TPC-H benchmark [EB/OL]. [2022-09-01].https://www.tpc.org/tpch

[94] TPC. TPC-DS benchmark [EB/OL]. [2022-09-01].https://www.tpc.org/tpcds

[95] Schnaitter K, Polyzotis N. A benchmark for online index selection [C] //Proc of the 25th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2009: 1701–1708

[96] Stillger M, Lohman G M, Markl V, et al. LEO-DB2's learning optimizer [C] //Proc of the 27th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2001: 19–28

[97] Holze M, Ritter N. Towards workload shift detection and prediction for autonomic databases [C]//Proc of the 1st ACM PhD Workshop in CIKM. New York: ACM, 2007: 109–116

[98] Schnaitter K, Polyzotis N. Semi-automatic index tuning: Keeping DBAs in the loop[J]. Proceedings of the VLDB Endowment, 2012, 5(5): 478−489

[99] Bruno N, Chaudhuri S. Interactive physical design tuning [C] //Proc of the 26th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2010: 1161–1164

[100] Chaudhuri S, Konig A C, Narasayya V. SQLCM: A continuous monitoring framework for relational database engines [C] //Proc of the 20th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2004: 473–484

[101] Thiem A, Sattler K U. An integrated approach to performance monitoring for autonomous tuning [C] //Proc of the 25th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2009: 1671–1678

[102] Calzarossa M, Serazzi G. Workload characterization: A survey[J]. Proceedings of the IEEE, 1993, 81(8): 1136−1150

[103] Yu P S, Chen M S, Heiss H U, et al. On workload characterization of relational database environments[J]. IEEE Transactions on Software Engineering, 1992, 18(4): 347−355

[104] Elnaffar S, Martin P, Schiefer B, et al. Is it DSS or OLTP: Automatically identifying DBMS workloads[J]. Journal of Intelligent Information Systems, 2008, 30(3): 249−271

[105] Wasserman T J, Martin P, Skillicorn D B, et al. Developing a characterization of business intelligence workloads for sizing new database systems [C] //Proc of the 7th ACM Int Workshop on Data Warehousing and OLAP. New York: ACM, 2004: 7–13

[106] Jain S, Yan Jiaqi, Cruanes T, et al. Database-agnostic workload management [C/OL] //Proc of the 9th Biennial Conf on Innovative Data Systems Research. 2019[2022-09-01]. www. cidrdb. org/cidr 2019/

[107] Paul D, Cao Jie, Li Feifei, et al. Database workload characterization with query plan encoders[J]. Proceedings of the VLDB Endowment, 2021, 15(4): 923−935

[108] Elnaffar S S, Martin P. An intelligent framework for predicting shifts in the workloads of autonomic database management systems [C/OL] //Proc of the 2004 IEEE Int Conf on Advances in Intelligent Systems–Theory and Applications. Los Alamitos, CA. IEEE Computer Society, 2004[2022-09-01].https://research.cs.queensu.ca/home/cords2/aista04.pdf

[109] Holze M, Ritter N. Autonomic databases: Detection of workload shifts with n-Gram-Models [G] // LNISA 5207: Proc of the 12th East European Conf on Advances in Databases and Information Systems. Berlin: Springer, 2008: 127–142

[110] Huang Xiangji, Peng Fuchun, An Aijun, et al. Dynamic web log session identification with statistical language models[J]. Journal of the American Society for Information Science and Technology, 2004, 55(14): 1290−1303

[111] Yao Qingsong, An Aijun, Huang Xiangqi. Finding and analyzing database user sessions [G] // LNISA 3453: Proc of the 10th Int Conf on Database Systems for Advanced Applications. Berlin: Springer, 2005: 851–862

[112] Luhring M, Sattler K U, Schmidt K, et al. Autonomous management of soft indexes [C] //Proc of the 23rd Int Conf on Data Engineering Workshop. Piscataway, NJ: IEEE, 2007: 450–458

[113] Sattler K U, Geist I, Schallehn E. QUIET: Continuous query-driven index tuning [C] //Proc of the 29th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2003: 1129–1132

[114] Deerwester S, Dumais S T, Furnas G W, et al. Indexing by latent semantic analysis[J]. Journal of the American Society for Information Science, 1990, 41(6): 391−407

[115] Agrawal S, Chu E, Narasayya V. Automatic physical design tuning: workload as a sequence [C] //Proc of the 2006 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2006: 683–694

[116] Kimura H, Coffrin C, Rasin A, et al. Optimizing index deployment order for evolving OLAP [C] //Proc of the 15th Int Conf on Extending Database Technology. New York: ACM, 2012: 276–287

[117] Agrawal S, Chaudhuri S, Narasayya V R. Automated selection of materialized views and indexes in SQL databases [C] //Proc of the 26th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2000: 496–505

[118] Microsoft. AutoAdmin project [EB/OL]. [2022-09-01].https://www.microsoft.com/research/project/autoadmin

[119] IBM. db2advis: Tools for designing indexes [EB/OL]. [2022-09-01].https://www.ibm.com/docs/en/db2/11.5?topic=indexes-tools-designing

[120] Percona. pt-index-usage: Percona toolkit documentation [EB/OL]. [2022-09-01].https://docs.percona.com/percona-toolkit/pt-index-usage.html

[121] ankane. dexter: The automatic indexer for Postgres [EB/OL]. [2022-09-01].https://github.com/ankane/dexter

[122] powa-team. PoWA: PostgreSQL workload analyzer [EB/OL]. [2022-09-01].https://github.com/powa-team/powa

[123] Duboce Labs, Inc. Index advisor (in-app in pganalyze) [EB/OL]. [2022-09-01].https://pganalyze.com/docs/index-advisor

[124] EnterpriseDB. EDB Postgres advanced server guide: Index advisor [EB/OL][2022-09-01].https://www.enterprisedb.com/docs/epas/latest/epas_guide/03_database_administration/02_index_advisor

[125] EverSQL. Online PostgreSQL/MySQL index advisor: Automatic indexing recommendations [EB/OL]. [2022-09-01].https://www.eversql.com/index-advisor-automatic-indexing-recommendations/

[126] Oracle. Oracle database performance tuning guide: Release 21 [EB/OL]. [2022-09-01]. https://docs.oracle.com/en/database/oracle/oracle-database/21/tdppt/index.html

[127] Microsoft. Automatic tuning [EB/OL]. [2022-09-01]. https://learn.microsoft.com/sql/relational-databases/automatic-tuning/automatic-tuning

[128] openGauss. Index advisor: Index recommendation [EB/OL]. [2023-01-25]https://docs.opengauss.org/en/docs/3.1.0/docs/Developerguide/index-advisor-index-recommendation.html

[129] OtterTune. Index recommendations [EB/OL]. [2023-01-25].https://docs.ottertune.com/documentation/database-instance-dashboard-and-recommendations/recommendations/index-recommendations

[130] Alibaba Cloud. Database autonomy service [EB/OL]. [2023-01-25].https://www.alibabacloud.com/help/en/database-autonomy-service

[131] Microsoft. Azure SQL — Family of SQL cloud databases [DB/OL]. [2022-09-01].https://azure.microsoft.com/en-us/products/azure-sql/

[132] Oracle. Oracle Exadata [DB/OL]. [2022-09-01].https://www.oracle.com/engineered-systems/exadata

[133] IBM. IBM Cloud database solutions [DB/OL]. [2022-09-01].https://www.ibm.com/cloud/databases

[134] Kraska T, Alizadeh M, Beutel A, et al. SageDB: A learned database system [C/OL] //Proc of the 9th Biennial Conf on Innovative Data Systems Research. 2019[2022-09-01]. http://www.cidrdb.org/cidr2019

[135] Marcus R, Negi P, Mao Hongzi, et al. Bao: Making learned query optimization practical[J]. ACM SIGMOD Record, 2022, 51(1): 6−13

[136] Chockchowwat S. Tuning hierarchical learned indexes on disk and beyond [C] //Proc of the 2022 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2022: 2515–2517

[137] Abu-Libdeh H, Altınbüken D, Beutel A, et al. Learned indexes for a Google-scale disk-based database [J]. arXiv preprint, arXiv: 2012.12501, 2020



Lai Sichao, born in 1993. PhD candidate. Student member of CCF. His main research interests include database tuning, database indexes, and AI4DB.

赖思超, 1993 年生. 博士研究生. CCF 学生会员. 主要研究方向为数据库调优、数据库索引、AI4DB.



Wu Xiaoying, born in 1973. PhD, associate professor. Member of CCF. Her main research interests include data management, data processing and optimization, keyword search, pattern mining, web net, and data integration.

吴小莹, 1973 年生. 博士, 副教授. CCF 会员. 主要研究方向为数据管理、数据查询处理和优化、关键字查询、模式挖掘、语义网、数据集成.



Peng Yuwei, born in 1980. PhD, associate professor. Member of CCF. His main research interests include database systems and digital watermarks.

彭煜玮, 1980 年生. 博士, 副教授. CCF 会员. 主要研究方向为数据库系统、数字水印.



Peng Zhiyong, born in 1963. PhD, professor. Fellow of CCF. His main research interests include databases, big data management and analysis, trusted data management, and complex data management.

彭智勇, 1963 年生. 博士, 教授. CCF 会士. 主要研究方向为数据库、大数据管理与分析、可信数据管理、复杂数据管理.