软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007680] [CSTR: 32375.14.jos.007680]

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

AI 赋能的多查询优化技术综述*

刘 全[1,2], 李翠平[1,2], 陈 红[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)

通信作者: 李翠平, E-mail: licuiping@ruc.edu.cn

摘 要: 随着数据规模与查询复杂度的持续增长, 传统数据库系统在面对高并发、多样化查询负载时, 暴露出显著的冗余计算与资源竞争问题. 多查询优化技术是提升数据库系统性能的重要途径, 其核心思想是通过识别和共享不同查询间的重叠计算, 减少重复执行并优化资源利用率. 根据重用时机与粒度的不同, 现有研究主要分为两条技术路线: 基于计算重用的多查询优化与基于存储重用的多查询优化. 前者在查询执行阶段实现算子级或计划级共享; 后者在存储层面实现持久化重用, 也被称为物化视图技术. 随着人工智能技术的发展, 强化学习和深度学习方法被广泛引入, 用于收益估计、计划生成和共享策略决策, 使计算重用过程具备自感知、自优化与自演化能力. 系统梳理传统方法到 AI 赋能方法的演化脉络, 对不同模型的优化目标、决策机制与系统架构进行比较与总结, 并分析现有方法在通用性、可扩展性与协同优化方面的不足. 最后, 展望未来智能化、多层级协同的数据库优化研究方向.

关键词: 计算重用; 多查询优化; 物化视图; 强化学习; 深度学习; 自适应处理

中图法分类号: TP311

中文引用格式: 刘全, 李翠平, 陈红. AI赋能的多查询优化技术综述. 软件学报. http://www.jos.org.cn/1000-9825/7680.htm

英文引用格式: Liu Q, Li CP, Chen H. Survey on AI-enabled Techniques for Multi-query Optimization. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7680.htm

Survey on AI-enabled Techniques for Multi-query Optimization

LIU Quan[1,2], LI Cui-Ping[1,2], CHEN Hong[1,2]

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

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

Abstract: With the continuous growth of data scale and query complexity, traditional database systems exhibit significant redundant computation and resource contention when handling highly concurrent and diverse query workloads. Multi-query optimization (MQO) is a crucial approach for enhancing database performance. Its core principle is to identify and share overlapping computations among different queries to minimize repetitive execution and optimize resource utilization. Based on the timing and granularity of reuse, existing research is primarily divided into two technical routes: computation-reuse-based MQO and storage-reuse-based MQO. The former implements operator-level or plan-level sharing during query execution, while the latter achieves persistent reuse at the storage layer, also known as materialized view technology. With the advancement of artificial intelligence, reinforcement learning and deep learning methods are widely used for benefit estimation, plan generation, and sharing strategy decision-making, enabling the computation reuse process to achieve self-awareness, self-optimization, and self-evolution. This study systematically reviews the evolution from traditional methods to AI-enabled approaches, compares and summarizes the optimization objectives, decision mechanisms, and system architectures of various models, and analyzes the limitations of existing methods in terms of generality, scalability, and collaborative optimization. Finally, future research

* 基金项目: 国家重点研发计划 (2023YFB4503600); 国家自然科学基金 (U23A20299, U24B20144, 62172424, 62276270, 62322214)
收稿时间: 2025-11-05; 修改时间: 2026-01-26, 2026-03-04; 采用时间: 2026-03-26; jos 在线出版时间: 2026-07-15

directions for intelligent and multi-level collaborative database optimization are envisioned.

Key words: computation reuse; multi-query optimization (MQO); materialized view; reinforcement learning; deep learning; adaptive processing

1 引 言

随着数据规模与业务复杂度的持续攀升, 现代数据库系统在面对高并发、多样化查询负载时暴露出显著的冗余计算与资源争用问题. 传统数据库管理系统多以 query-at-a-time 的方式独立编译和执行单条查询: 从逻辑优化、物理计划生成、算子执行到结果返回均相互隔离. 该范式在处理单条复杂 OLAP (on-line analytical processing) 或 OLTP (on-line transaction processing) 查询时已相对成熟, 但当多条复杂查询同时运行且其访问数据、子表达式或算子存在重叠时, 不同执行计划之间会竞争 I/O、CPU、内存与锁资源, 系统整体吞吐量与尾延迟显著恶化. 如何在并发条件下降低冗余、缓解冲突并提升资源利用率, 成为数据库系统设计与优化的重要课题.

围绕这一目标, 形成了两条各有侧重的技术路线: 其一是基于计算重用的多查询优化 (multi-query optimization, MQO), 通过重用公共子计划、共享中间结果或共享算子, 减少重复计算与数据访问; 其二是基于存储重用的多查询优化, 即物化视图 (materialized view, MV), 通过离线/在线选择关键子表达式进行预计算与持久化, 在查询重写阶段直接复用以缩短执行路径. 两类方法在优化目标上趋同 (提升吞吐、降低延迟与成本), 但在时机 (运行时 vs. 预先)、粒度 (算子/子计划 vs. 视图) 等方面各具侧重. 传统方法虽已取得诸多进展, 但仍普遍面临许多挑战: (1) 共享机会识别依赖规则与专家经验, 泛化性与可迁移性不足; (2) 工作负载动态变化导致既有共享结构与缓存策略失效; (3) 收益评估困难, 尤其在并发下锁与资源竞争、计划重排等因素难以被静态代价模型准确捕捉; (4) 搜索空间爆炸, 全局共享计划或候选视图选择在大规模负载下代价高昂.

近年来, 人工智能 (AI) 技术的快速发展为数据库系统带来了新的优化范式. 通过学习驱动的模式识别、强化学习调度与自适应代价建模, 强化学习辅助视图价值判断, AI 能够在复杂、高维的优化空间中自动发现潜在重用机会与高效执行策略, 从而显著提升传统多查询优化与物化视图方法的自动化与智能化水平.

本文聚焦“AI 赋能的多查询优化技术”, 系统梳理从执行期共享到视图级重用的核心思路与代表性工作. 图 1 给出了本文内容的逻辑关系图, 展示了上述两条技术路线, 在基于计算的多查询优化路线中, 技术演进经历了从传统启发式方法到自适应处理, 再到强化学习 (如 RouLette) 与深度学习 (如 Lemo) 驱动的范式转变; 在基于存储重用的多查询优化 (自动物化视图) 路线中, 研究脉络从传统方法演进至 ILP 全局建模, 并进一步发展为基于学习型收益估计与策略决策的自治管理体系, 如 RLView、DQM 及 AutoView 等代表性成果.

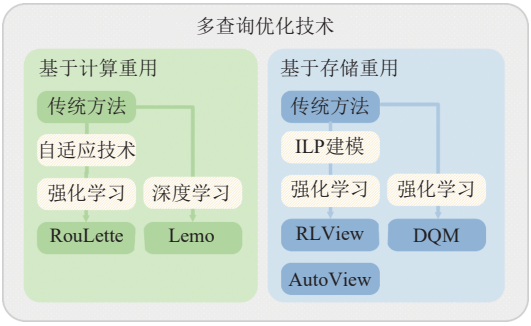

图 1 本文内容结构图

与已有研究相比, 文献 [1] 对智能数据管理技术进行了综述, 其中部分内容涉及并发查询代价估计与物化视图, 但仅限于早期的机器学习方法, 且具体研究任务与本文有所不同. 文献 [2] 则系统总结了在查询执行不同阶段采用的多查询共享技术, 对各类传统启发式方法进行了较为全面的归纳与分析, 但未涵盖与机器学习或深度学习相结合的最新研究进展, 也未涉及物化视图相关方法. 综合来看, 现有综述多侧重于传统优化框架或早期的智能化尝试, 而缺乏对多查询优化与自动物化视图在人工智能赋能下的系统总结与比较, 本文正是针对这一空白展开研究.

2 基于计算重用的多查询优化技术

2.1 传统的计算重用技术

基于计算重用的多查询优化 (MQO) 是指针对一组相关查询生成统一且高效的执行计划的过程, 而不是对每条查询分别进行独立优化与执行. 其主要目标是通过识别并共享查询之间的公共子表达式、数据表扫描以及中间结果, 以降低整体执行代价. 通过重用这些共享计算, MQO 能够在系统性能和资源利用率方面显著优于传统的逐查询处理模式.

如图 2 所示, 基于计算重用的多查询优化按优化方式可分为实时查询优化和批量查询优化. 前者是在线运行时进行共享机会的检查, 主要针对单用户的实时查询, 优点是检测开销低, 例如基于子表达式匹配的方法, 但只能发现工作负载中部分共享机会, 共享覆盖范围有限; 后者是一种离线预执行重构的方式, 在执行前评估所有共享机会并重构查询执行方式, 适用于分析型任务, 能够全面发现共享潜力, 但搜索空间随查询数呈双指数增长, 优化代价高昂.

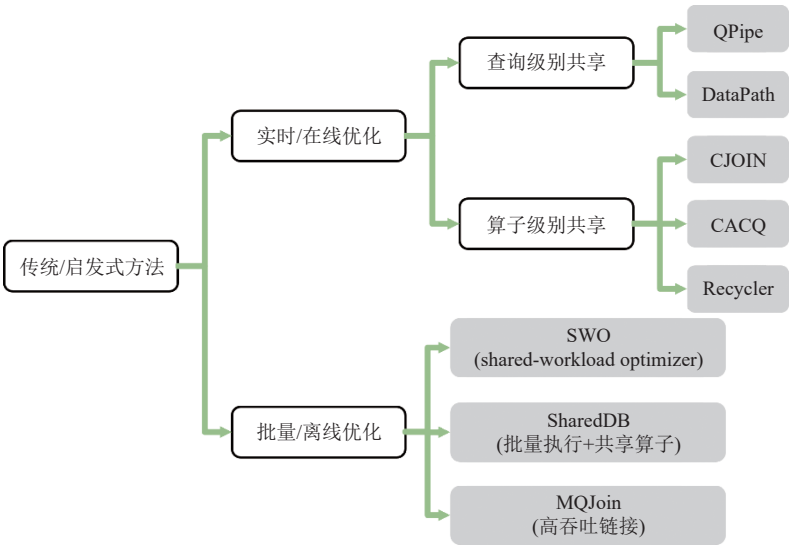

图 2 传统的基于计算重用的多查询优化方法分类

在实时的优化方向, 一类方法通过传统 DBMS 生成查询计划, 并缓存先前查询的子查询的中间结果, 然后利用这些结果来加速后续查询的执行. 例如 QPipe[3]、Recycler[4]都是这种思路; 另一类方法采用手动设计的规则来指导查询计划的生成, 如 DataPath[5]、CJOIN[6]和 CACQ[7], 以 DataPath 为例, 它为并发查询维护一个全局计划, 提交新查询时, 通过重用公共子计划并最小化计划中每个节点的基数总和, 将新查询集成到全局计划中. 其中 QPipe 和 DataPath 方法都是在查询级别检测共享机会, 前者仅共享公共子计划, 后者则维护了一个全局最优查询计划, CJOIN、CACQ 和 Recycler 都是在算子级别检测共享机会, 基于选择率或重用价值在运行时重排算子.

在离线的优化方向, 早期系统如 SWO (shared-workload optimizer)[8]通过生成可共享算子组成的全局计划实现共享, 但代价极高. SharedDB[9]在此基础上提出批处理执行模型 (batched execution model), 通过数据-查询模型与全局计划实现算子级共享; 其代表性思想是以查询批次 (batch) 为单位进行全局优化和共享算子执行, 在高并发场景下显著减少冗余计算. 后续的 MQJoin[10]则在 SharedDB 的基础上进一步增强了连接操作的吞吐能力.

2.2 基于强化学习和自适应处理的方法

随着机器学习和深度学习的不断发展, 人们将 AI 与多查询共享技术相结合, 增强传统的多查询共享技术, 实现自动地感知查询结构、预测共享收益、选择并维护共享数据. 并依此解决传统计算重用方法高度依赖人工规则、无法适应动态负载等问题.

如图 3 所示, 文献 [11] 提出了一个专门用于多查询执行的执行引擎 RouLette, 通过运行时自适应地检测共享机会解决过去工作存在的不足之处, 并创新性地运用强化学习来选择要探索和利用的共享机会. RouLette 的优化思路来源于 SharedDB[9]提出的数据-查询模型, 并在此基础上融入了多种自适应处理技术和强化学习方法. 早期的采用手动设计的规则来指导查询计划的生成的方法, 如 DataPath、CJOIN 和 CACQ 等方法也都有类似数据-查询模型的设计, 利用位图 (bitmap) 对数据进行标记, 以便于后续的优化处理.

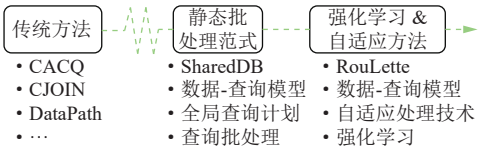

图 3 从传统方法到强化学习驱动的自适应方法

2.2.1 多查询优化中批处理共享范式回顾

文献 [9] 提出了基于计算重用多查询优化的批处理共享执行范式, 并据此构建了 SharedDB 系统, 能高效执行大规模并发查询负载. SharedDB 的核心思想是通过批量化执行查询, 在多条并发查询之间实现计算共享, 完成在高吞吐条件下保证相应时间的目标. 数据-查询模型 (data-query model)、全局查询计划 (global query plan)、批量查询执行 (batched query execution) 和共享算子 (shared operator) 共同帮助 SharedDB 实现了在大量并发查询负载下的优异性能.

文献 [9] 引入了一种新颖的数据-查询模型来表示查询的中间结果. 该模型在传统的关系数据模型基础上进行了扩展, 新增了一列查询标识符, 用来记录对当前元组感兴趣的查询. 具体来说, 在该数据-查询模型中, 一个关系 R 的模式可表示为: $\{R_a, R_b, \ldots, R_n, \text{query}_{\text{id}}\}$, 其中, R_i表示 R 的常规属性, 而 query_{id} 属性则用于唯一标识使用该元组的查询. 在传统的“第一范式 (first normal form, 1NF)”关系数据库系统中, 实现这种数据-查询模型会导致大量冗余. 如果一个元组与多个查询相关, 就必须为每个相关查询生成并处理一份拷贝. 这样一来, 查询处理的复杂度以及主内存的占用都会随着查询数量线性增长. 因此, SharedDB 在内部将该列实现为集合值属性 (set-valued attribute), 采用 NF^2 (non-first normal form) 模型. 一个算子只需对一个元组应用一次, 而无需重复应用于所有同时订阅该元组的并发查询或更新操作. 这不仅显著减少了冗余计算, 还有效降低了内存占用, SharedDB 中使用了空间效率和时间效率优秀的列表来实现该集合值属性.

SharedDB 将系统中的全部工作负载编译为一个全局查询计划. SharedDB 的算子共享机制与 QPipe、CJOIN、DataPath 等支持算子级共享的系统不同. 支持算子级共享的方法在查询一到达时便立即开始执行, 而 SharedDB 则采用批处理的方式对查询和更新, 当一批查询和更新正在执行时, 新到达的查询与更新会进入队列; 当前批次处理完成后, 队列中的查询与更新将被取出, 形成下一批任务. 每一批任务中都会进行共享算子的计算, 在某种意义上, 全局查询计划中的所有算子都可以被查询编译器视为可用的物化视图, 以此加速临时查询的处理, 类似的观点也出现在 QPipe 方法中.

SharedDB 中对算子的共享针对连接、排序和分组这 3 类. 以连接算子的共享为例, 针对涉及相同表但谓词条件不同的查询. 每个查询都会单独解析与编译, 并执行谓词下推等逻辑查询优化, 得到每个查询对应的查询计划. 再将查询计划进行合并, 得到全局计划, 系统不会分别执行 3 个小规模的连接, 而是执行一个同时满足多个查询的较大规模的连接 (多个查询所需元组的并集作为连接输入, query_{id} 帮助进行元组的选择). 直觉上看, 这种处理方式似乎并不理想: 通常同时执行几个小连接要比执行一个大连接更高效. 然而, 当系统中存在大量并发查询, 且这些查询在待处理元组上存在重叠时, 这种全局共享连接的方式就会展现出明显优势. 其他算子也是类似的做法, 将多次小规模的计算合并为一个满足多个查询的计算, 来减少高并发场景下的计算冗余.

综上, SharedDB 通过批量化执行与全局共享计划的方式, 在高并发场景下显著提升了系统吞吐量与资源利用率. 其关键思想在于将多条查询统一为一个可共享的全局计划, 通过数据-查询模型与共享算子机制, 在批次内最

大化计算重用.

2.2.2 强化学习驱动的自适应方法

SharedDB 提出的是一种“先优化后执行 (optimize-then-execute)”的静态批处理范式, 在大规模查询负载下的优化代价非常高昂. 为突破静态全局计划的限制, 研究者逐步将注意力转向自适应处理范式, 使查询执行过程能够根据实时反馈信息动态调整优化策略, 实现“边执行、边优化”的自调整机制. 自适应处理技术通过运行时监控操作符性能与数据流特征, 动态改变算子顺序、执行策略及共享决策, 从而在不可预测的环境中保持高效执行.

在这一思路基础上, 研究者进一步引入强化学习, 将操作符调度与共享机会选择建模为决策过程. 与 SharedDB 的离线批量共享不同, 强化学习方法能在运行时自适应地探索和利用共享机会, 通过学习动态环境下最优的执行与共享策略, 实现了多查询优化从静态批处理到智能自学习优化的关键跨越.

以 RouLette 方法为代表, 其创新之处在于自适应的动态优化, 它不使用大多数数据库采用的“先优化再执行 (optimize-then-execute)”的范式, 而是在查询的执行过程中根据运行时收集的信息动态调整执行计划, 使得“计划生成”与“计划执行”能够并行进行. 这种在线优化机制能够在动态环境中捕获新的共享机会, 从而显著提升系统吞吐率与响应效率.

例如, 假设历史查询中出现了连接序列 $R\bowtie S\bowtie T\bowtie U$, 而当前系统收到的新查询计划为 $R\bowtie S\bowtie V\bowtie U$. 传统实时共享方法只能识别 $R\bowtie S$ 的共享机会, 而无法检测到更复杂的 $R\bowtie S\bowtie U$ 共享潜力, 因为先前的执行计划及中间结果缓存已固定. RouLette 则能够在运行时重新规划连接顺序, 通过动态决策发现此类跨查询的潜在共享机会.

2.2.2.1 自适应处理技术与 RouLette 执行流程

RouLette 中使用了 3 种自适应处理 (adaptive processing) 技术, 以应对统计信息不足、数据高度相关或执行环境不可预测的复杂场景.

● 对称哈希连接 (symmetric hash join): 传统哈希连接依赖固定的构建端 (build side) 和探测端 (probe side), 因此灵活性有限. 而对称哈希连接在两个输入关系上同时建立哈希表, 从而实现无序数据流的处理能力. 每当新的元组到达时, 系统既向对应哈希表插入元组, 又立即在另一侧哈希表中探测匹配项, 从而实现真正的流式、对称执行. 这一机制为动态重构执行路径奠定了基础.

● 涡流操作符 (eddy): eddy 是文献 [12] 提出的一种在执行期间动态调整算子执行顺序的机制. 它通过控制元组在算子之间的流动路径来观察输入与输出行为, 从而不断学习最优的执行序列. eddy 可视为一种“路由层”算子, 它根据运行时反馈决定下一步的操作符方向, 实现执行顺序的自我优化.

● 状态模块 (state module): 状态模块则是文献 [13] 提出的一种增强涡流操作符适应性的技术, 状态模块为每个基表存储元组, 定义了插入和探测两个操作, 可以用来实现多元的对称哈希连接.

RouLette 独立于数据库管理系统运行. 数据库管理系统在解析与优化阶段生成多个 SPJ (select-project-join) 子查询计划, 并将这些子计划交由 RouLette 并发处理. RouLette 在执行过程中周期性地调度涡流算子, 对计划执行路径进行重新规划, 并根据执行日志实时更新强化学习策略. 通过这种循环迭代方式, RouLette 能够在保证执行效率的同时, 持续逼近全局最优的查询共享策略.

在每个执行周期中, 涡流算子依据运行时采样的中间结果基数估计当前操作符的响应时间, 以最小化全局执行代价为目标函数, 对操作符选择和连接顺序进行自适应调整. 总代价被定义为所有操作符代价之和, 通过强化学习策略逐步逼近响应时间最小化的全局最优策略.

2.2.2.2 马尔可夫决策过程建模

RouLette 将涡流操作过程抽象为一个马尔可夫决策过程 (Markov decision process, MDP), 具体建模如下.

● 智能体: 涡流操作符 (eddy).

● 状态: 由操作符集合、查询集合和输入规模 (由操作符选择率决定) 构成一个三元组, 并以栈的形式存储三元组, 栈顶为当前步骤, 状态即为该栈.

● 动作: 从候选操作符集合中选择一个操作符推入栈顶, 表示执行该操作.

● 状态转移: 动作的执行会触发递归步骤, 导致查询计划的重构与状态更新, 进而生成新的系统状态.

● 奖励: 动作的即时奖励对应所选操作符的执行代价. RouLette 通过最小化累积代价 (即最大化负代价奖励) 来逼近最优的全局调度策略.

在强化学习框架下, RouLette 通过持续采样执行日志、估计延迟反馈并更新策略网络, 实现了查询执行的在线自适应优化与共享机会发现. 这一设计有效填补了“离线批量优化”与“在线单查询执行”存在的方法缺陷, 为多查询优化系统提供了可扩展的智能优化路径.

2.3 基于深度学习和子计划重用的方法

与 RouLette 所代表的“在线自适应优化”路线不同, 有另一类探索多查询优化问题的技术路线, 即通过系统化的计划重用与缓存共享机制提升查询执行效率. 与 RouLette 依赖自适应处理技术实现运行时的动态调度不同, 这一思路更强调从历史执行中提取可复用结构, 以减少后续查询的优化与执行开销.

代表性工作是 Recycler[4]. 该方法提出了一种高效的查询计划重用框架, 其核心思想是在系统中维护一个可索引的共享计划库, 通过识别历史查询中的公共子计划, 实现新查询的快速匹配与复用. Recycler 在逻辑计划层面建立了“重用索引”, 并结合基于代价估计的策略选择最优重用路径, 从而避免重复计算、降低优化成本. 与 RouLette 不同, Recycler 并不依赖在线学习或运行时决策, 而是以结构化的计划缓存与启发式重用机制为核心, 实现跨查询的子计划共享.

如图 4 所示, 过去也有若干篇类似 Recycler 回收架构的研究, 早在 20 世纪 80 年代, 文献 [14,15] 就已经研究了如何在连续的查询中自动识别并利用公共子表达式的问题. 随后也有多项研究提出了不同的回收体系结构, 例如文献 [16] 提出了在逐操作符 (operator-at-a-time) 执行范式下的回收机制, 该体系结构在列存储系统中通过一种轻量级机制对中间结果进行回收与重用; 文献 [17] 提出了 DynaMat 系统, 能够动态地在多个粒度层次上物化信息, 来适应不断变化的动态负载; 文献 [18] 提出了 Exchequer 系统, 是一种与查询优化器高度耦合的自动查询缓存系统, 确保缓存管理与优化决策相互一致, 并设计了一种名为 Incremental 的缓存管理与替换算法, 用于动态维护缓存内容; 文献 [19] 提出了一个为数据仓库环境设计的智能缓存管理 WATCHMAN, 设计了缓存替换与缓存准入策略来最小化查询响应时间, 但缓存考虑的只有最终查询的结果; 文献 [20] 设计了一种面向决策支持系统 (decision support system, DSS) 的查询缓存管理器, 通过两种方式在缓存中查找查询结果: 一是基于精确匹配, 直接返回先前完全相同查询的结果, 二是通过查询分解算法, 高效地查找包含 (subsumes) 当前查询的已缓存结果, 从而实现部分结果的复用; 文献 [21] 提出了 COD (cache-on-demand) 按需缓存系统, 核心思想是将正在执行的查询的中间结果或最终结果视为可被后续查询利用的“虚拟缓存”, 当有新查询到来时, 系统会检测现有查询过程中产生的中间或最终结果, 如果这些结果对新查询有帮助, 则可立即将其物化为实际缓存.

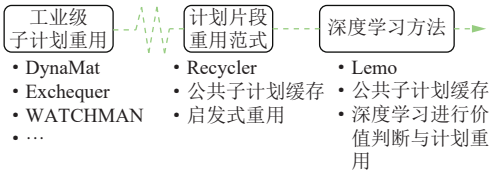

图 4 从工业级子计划重用到深度学习方法

Recycler 则代表着计划片段重用 (plan fragment reuse) 的典型范式, 从离线规则迈向了部分自动化的共享管理, 为后续基于学习或自适应机制的共享优化提供了基石.

在此的基础上, 研究者进一步提出了 Lemo[22]. Lemo 延续了 Recycler 关于“计划缓存与重用机制”的核心理念, 但引入了深度学习模型来自动捕捉查询间的相似性与共享模式, 从而实现数据驱动的多查询优化. 换言之, Lemo 将 Recycler 的工程化“重用”思想发展为智能化的“可学习”机制, 代表了多查询优化从规则驱动向学习驱动演进的重要方向.

2.3.1 计划片段重用范式回顾

Recycler 提出了一种自动化的查询中间结果重用框架, 其核心思想是通过全局计划图 (global plan graph) 与有

限缓存池 (cache pool) 的结合, 实现查询间的中间结果识别与复用. 系统维护一个历史查询负载的全局查询计划图, 用以记录以往查询的逻辑结构和子计划片段, 同时配备一个有限容量的内存缓存, 用于存储执行过程中产生的中间结果. 新查询到来时, Recycler 能够利用该计划图识别可复用的部分, 从缓存池中直接重用已有中间结果, 从而减少重复计算、降低优化与执行开销.

在 MonetDB[16]中也实现了中间结果复用的功能, MonetDB 的设计基于逐操作符范式, 中间结果总是作为查询执行的副产品进行物化, 中间结果已经在主内存中, 系统可依据执行代价与结果大小决定保留或丢弃. 而更为广泛使用的逐元组 (tuple-at-a-time) 或逐向量 (vector-at-a-time) 查询执行范式通过流水线式处理尽量避免中间结果物化, 这导致在流水线模型中实现结果复用变得困难. Recycler 的提出正是为了解决这一挑战, 它在流水线式查询执行框架下引入可控的中间结果物化与重用机制, 实现了高效的共享计算.

考虑到物化一个中间结果必然会导致计算资源的消耗和存储成本的增加, 必须慎重选择哪些中间结果需要缓存, 以及需要缓存多长时间. 因此, 系统必须在缓存前对每个潜在的中间结果进行成本收益分析, 指引有效的中间结果物化与重用. Recycler 因此在体系结构中引入了两个关键设计.

● Recycler graph (中间结果重用图): Recycler 以有向无环图 (directed acyclic graph, DAG) 的形式维护关系操作符及其参数, 每个节点对应一个关系代数算子. 新查询的逻辑计划会被映射到该图中, 与已有节点进行模式匹配, 完全匹配的子树会被合并, 从而识别潜在的复用机会. 该结构不仅支持对新查询中可重用结果的查找, 也用于标识哪些中间结果值得物化.

● Recycler cache (缓存管理机制): Recycler 设计了带有准入策略 (admission policy) 与替换策略 (replacement policy) 的有限缓存池. 当缓存未满时, 系统根据准入策略选择当前正在执行的查询中的中间结果进行物化; 当缓存已满时, 则依据收益指标 (benefit metric) 选择性地淘汰低收益结果, 并替换为收益指标更高的中间结果.

Recycler 方法的一个关键点是收益指标的设计, Recycler 设计了一个收益度量 $B(R)$ 来评估每个结果 R 的收益, $B(R)=\frac{cost(R)\cdot h_R}{size(R)}$, 其中 $cost(R)$ 代表真实计算所需的时间, 反映每次复用可节约的计算量, h_R 代表中间结果 R 的重要性, 反映它未来被复用的可能性, $size(R)$ 代表中间结果 R 在缓存中占用的内存空间. h_R 是对中间结果使用频率的度量, 被定义为 Recycler graph 中假设所有当前缓存的已物化结果都被首次计算时物化的情况下, 可以使用 R 来回答的最大查询数量. 节点首次插入 Recycler 图时, h_R 被设置为 0, 后续根据缓存的操作而动态调整, 比如节点 v 的结果被物化添加到缓存后, 需要递归地对其所有后代节点更新, 减去节点 v 对应的 h_R 值, 即节点 v 不需要再计算.

2.3.2 Transformer 模型驱动的方法

尽管 Recycler 在计划片段重用方面实现了自动化的框架设计, 但其核心依然依赖人工设定的启发式规则与静态收益度量. 在实际应用中, Recycler 的中间结果识别高度依赖专家经验, 其共享检测主要通过语法树模式匹配或启发式规则实现. 由于系统难以准确评估共享收益, 当中间结果的存储与维护开销超过其带来的复用收益时, 整体性能甚至可能下降.

为突破这些限制, 后续研究引入学习驱动的决策机制, 使系统能够在运行时自行感知负载特征并自适应地调整重用策略. Lemo[22]提出了首个基于 Transformer 的多查询延迟预测模型——MQFormer, 其中设计了两个针对多查询优化任务的注意力机制, 可以捕捉并发查询之间的关联, 同时考虑多查询间的冗余计算和并行冲突. 借助该模型, 系统能够在实时多查询优化场景中预测并发执行代价, 从而为新到查询生成高效的计划. 这是首个在实时多查询优化中利用价值网络 (value network) 预测并行查询代价的研究工作. Lemo 考虑的是一个实时的多查询优化场景, 数据库当前正在执行查询 Q_1, 此时一个新的查询 Q_2 到达, 为查询 Q_2 生成一个合适的执行计划.

2.3.2.1 计划片段重用范式与 Lemo 架构

如图 5 所示, 左侧是计划片段重用范式的优化流程, 查询输入经过查询解析器、优化器, 对子计划片段进行价值判断, 并在缓存中预存下预测高重用价值的片段, 交予查询执行器进行并行执行; 右侧是 Lemo 的架构概览, 核心变动包含价值网络、计划搜索、缓存管理器这 3 部分. Lemo 实现了一个共享缓存管理器来缓存子查询的中间

结果, 利用负载驱动的缓存管理策略减少冗余计算, 管理策略同时考虑子查询的重用概率和重用价值; 同时, 设计了一个两阶段的计划搜索算法, 在 MQFormer 的指引下广度优先搜索找到一个最优的查询计划.

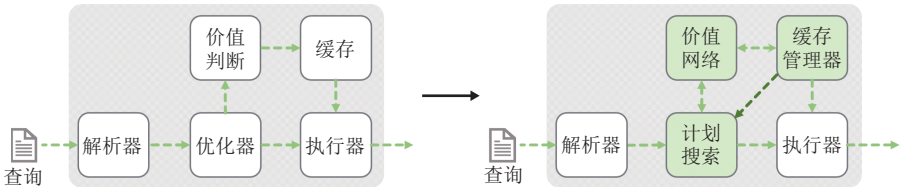

图 5 从计划片段重用范式到 Lemo 架构

Lemo 价值网络的设计受到了 QueryFormer[23]中查询表示方法的启发, 设计了一个基于 Transformer 的价值网络, 称为 MQFormer, 用于预测查询片段的计算时间. 标准的 Transformer 模型主要由两个模块构成: ① Encoder: 负责理解输入文本, 为每个输入构造对应的语义表示 (语义特征); ② Decoder: 负责生成输出, 使用 Encoder 输出的语义表示结合其他输入来生成目标序列. Lemo 用到的是纯 Encoder 模型, 要求编码器能够准确地理解查询及其计划的语义.

2.3.2.2 跨查询的语义特征表征

MQFormer 实现了跨查询的语义特征表征, 其输入由两部分组成: 一组并发 (或历史) 查询及其完整计划, 以及一个新到查询及其部分计划. 模型同时编码查询级特征和计划级特征.

● 查询级编码: 提取查询的连接图和谓词信息, 然后通过多层感知器 (MLP) 将这些特征转换为一个 32 维向量进行查询信息嵌入.

● 计划级编码: 对每个计划节点提取 3 类信息.

● 操作类型: 聚合为 7 种典型操作 (哈希连接、归并连接、循环连接、索引扫描、顺序扫描、聚合、共享), 编码为一个 7 维独热向量.

● 关系信息: 以独热编码方式表示节点涉及的表.

● 位置特征: 用二维向量编码节点在计划中的序列位置及深度层级.

最终, 编码器输出经 3 层 MLP 成本估计器预测各计划的最优可能延迟, 经过全连接层整合形成固定维度的延迟预测输出.

2.3.2.3 基于注意力机制的关联度建模

多查询优化的核心难点在于识别海量查询中哪些子计划具有重用价值. Lemo 引入了注意力机制 (attention mechanism) 来建模查询间的相关性. 这种设计改变了传统方法只能依靠文本匹配或硬性规则识别重用的局限, 使得系统能够动态察觉到查询间的关联与冲突, 从而在复杂的并发负载中自适应地提取共享策略.

Lemo 为捕捉查询间依赖与冲突模式, 设计了两种新颖的注意力机制.

● 数据流注意力 (data flow attention, DA), 对于查询计划树中的节点对 n_i 和 n_j, 由于 DAG 中的边代表数据流方向, 仅在 DAG 中 n_i 和 n_j 间存在边时, 计算注意力得分.

● 相关性注意力 (correlation attention, CA), 类似于数据流注意力, 由于相关性模式 (如资源竞争和锁冲突) 仅存在于不同查询之间, 我们仅计算一个节点与其他不在同一查询计划中的节点之间的注意力得分.

与全自注意力机制相比, 这两种机制有效减少了 Transformer 的计算复杂度, 使模型能够在保持表达能力的同时高效感知并发查询间的语义与资源相关性.

2.3.2.4 基于深度增强学习的执行策略寻优

在决策层面, 如何排布多个查询的执行顺序以最大化重用收益是一个复杂的组合优化问题. Lemo 的解决方案是利用价值网络 (value network) 结合强化学习来替代传统的动态规划求解器. Lemo 的优化目标是最小化整体的执行时间, 因此损失函数定义为 $Loss = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{l}_i - l_i\right)^2$, 其中 l_i 是第 i 个计划的真实执行时间, 而 $\hat{l}_i$ 是由 MQFormer

预测的 m 个并发执行的情况下, 第 i 个计划的预测执行时间.

在 MQFormer 的指导下, 计划搜索模块采用两阶段算法生成最优查询计划.

● 候选节点选择阶段: 从共享缓存中选择重用收益最高的子计划节点.

● 计划生成阶段: 基于 MQFormer 的预测结果执行 beam best-first search, 逐步扩展并生成完整查询计划.

共享缓存管理器作用是维护一个共享缓存区, 由写入策略和替换策略两部分组成, 替换策略同时考虑节点被重用的概率和重用收益 (重用收益使用价值网络 MQFormer 进行预测), 写入策略在缓存未满时直接写入候选子查询, 缓存区空间不足时则使用替换策略. 查询执行器当中集成了自定义的共享操作符, 在执行期间写入和读取共享缓存器.

与传统的利用查询优化器生成计划或采用人工规则加速计划生成的方法不同, Lemo 提出了一个基于 Transformer 的价值网络作为查询计划生成的基础. 该网络的预测目标是预测部分计划 (partial plan) 在并发执行场景下可能达到的最佳运行时间, 并在网络中设计了两种新颖的注意力机制来降低模型的时间复杂度, 能够捕获并发查询之间的相关性, 例如全局缓冲区共享、锁冲突和资源冲突, 这些相关性会影响并发查询的查询延迟.

2.4 小　结

本节对基于计算重用的多查询优化领域的技术演进进行了系统的梳理, 通过对比分析揭示从底层结构化共享向高层智能化驱动转变的技术演进. 如表 1 所示, 现有的技术可以根据其核心方法划分为传统方法与深度学习方法两大类. 在传统方法路线中, SharedDB 与 Recycler 分别代表了算子级物理共享与子计划结果缓存的典型范式. 此类方法主要通过修改数据库执行引擎的算子逻辑或采用启发式缓存策略来减少冗余计算, 其优势在于执行开销极低且在模式固定的高并发短查询任务中表现出卓越的吞吐量; 然而, 其共同的局限在于高度依赖静态代价模型或固定的规则匹配, 导致其在面对异构工作负载漂移时缺乏足够的灵活性.

表 1　基于计算重用的多查询优化方法对比

类别	方法	解决问题	核心技术	适用场景	优点	缺点
传统方法	SharedDB	高并发下的系统吞吐量瓶颈	全局共享查询计划、批处理执行引擎	模式固定、高并发的短查询任务	算子级共享极高, 能处理大规模并发	静态批处理模式, 缺乏对异构负载的灵活性
	Recycler	细粒度中间结果的缓存与替换	启发式缓存管理、子计划匹配	数据重复度高、具有时间局部性的负载	实现了算子结果的灵活复用	依赖静态代价模型, 面对负载漂移时适应性差
启发性研究 (学习型单查询优化)	Neo、Bao、Balsa等	传统代价模型不准及搜索空间爆炸	深度神经网络、强化学习、Hint-based思路	具有复杂关联逻辑或传统优化器难以精确建模的数据库优化场景	有效捕捉查询计划中的非线性特征, 显著提升了复杂环境下的计划生成质量	仅针对单查询, 未考虑并发查询间的重用机会
深度学习方法	RouLette	算子共享决策的实时自适应	强化学习、自适应调度	负载频繁波动、查询模式动态变化的环境	无需精确代价模型, 实时平衡重用收益与开销	状态空间表征较简单, 难以处理极复杂的长查询
	Lemo	复杂并发查询下的全局执行计划优化	Transformer、价值网络	逻辑复杂、重用机会隐含的分析型长查询	语义表征能力强, 能从全局视角优化执行序列	训练成本高, 推理延迟对实时性有一定影响

值得注意的是, 尽管在传统方法到深度学习方法期间, 专门针对学习型多查询优化的研究产出相对平缓, 但这一时期学习型查询优化 (learned query optimization) 技术的突破为后续智能化多查询优化的产出奠定了核心技术基石. 以 Neo[24]为代表的工作探索了利用 TreeCNN 捕捉查询计划特征以进行代价预测; Balsa[25]则通过构建价值网络 (value network) 引导强化学习搜索, 实现了高效的计划寻优; 而 Bao[26]则开创了基于 Hint 的辅助优化模式, 通过引导数据库原生优化器进行计划选择, 有效平衡了模型预测潜力与系统稳定性. 这些启发性研究成功证明了深度学习在处理非线性代价建模与超大规模搜索空间中的卓越能力, 直接启发了后续如 RouLette 和 Lemo 等方法将

研究对象从“单计划代价拟合”升级为“并发查询间的全局重用价值预测”.

进入深度学习驱动阶段后, 研究重心从“如何物理共享”转向了“如何识别隐含重用机会”. RouLette 引入强化学习机制, 通过实时感知负载反馈实现了算子共享决策的自适应调度, 有效解决了传统方法难以应对动态负载波动的问题. 而 Lemo 作为最新的代表性成果, 利用 Transformer 架构捕捉并发查询间的深层语义关联, 实现了从局部缓存替换向全局执行序列优化的跨越, 显著提升了复杂分析型长查询的重用效率.

从综述视野出发, 本文认为基于计算重用的多查询优化技术的发展已进入“价值预测驱动”的新阶段, AI 的引入并非改变了降低冗余的本质, 而是赋予了系统在海量执行计划空间中精准捕获重用价值的能力. 尽管如此, 深度学习模型带来的推理延迟与训练成本仍是制约其进入工业级内核的主要障碍. 未来的研究应致力于开发与数据库执行器深度耦合的轻量化神经算子, 以实现快速决策, 并探索将执行期的计算重用与存储期的结果持久化进行跨层协同, 从而在多维资源约束下寻求全局性能的最优权衡.

3 基于存储重用的多查询优化技术

基于计算重用的多查询优化策略可以大大提高并行负载下的数据库整体性能, 通过共享公共子表达式, 以及智能调度减少查询间的并发冲突, 系统可以有效地减小冗余计算和系统负载. 在机器学习和深度学习技术的加持下, 实现了自动识别查询间的共享机会、自主预测查询间冲突, 并寻找最优的执行计划.

此外, 还有另一种基于存储重用的更为结构化和持久化的共享策略——物化视图 (materialized view). 物化视图是一种存储预先计算结果的数据库对象, 可以是表或连接结果的行和列的子集, 也可以是聚集函数的汇总结果, 从而在后续查询中直接复用, 显著减少重复计算并降低响应延迟. 物化视图广泛应用于联机分析处理 (OLAP) 系统中, 用于提升复杂查询的执行效率, 也可在混合负载环境下改善系统吞吐率与资源利用率.

在传统数据库系统中, 物化视图的定义和管理通常由数据库管理员 (database administrator, DBA) 人工完成. DBA 需要根据经验、历史工作负载和系统约束, 选择一组候选视图进行物化, 以最大化查询性能. 然而, 这种方法存在明显局限: 其一, 视图选择依赖专家经验, 难以自动化; 其二, 候选空间随数据规模与查询复杂度呈指数级增长; 其三, 维护与刷新视图的开销高昂, 尤其在数据频繁更新的场景下, 易造成性能瓶颈. 这些挑战推动了自动化物化视图生成与管理技术的研究.

关于自动物化视图的研究大量借鉴了多查询优化与公共子表达式选择[24,27–33]等方向的思想, 这些研究在策略上具有相似性, 但目标并不完全相同. 相较而言, 自动物化视图问题比子表达式选择问题更加广泛, 因为前者可以考虑那些并未直接出现在工作负载查询中的计算, 这使得可能的解空间大幅扩展, 并增加了查询包含性判定与物化视图改写的复杂度.

图 6 展示了基于存储重用的多查询优化方法的发展脉络. 从商业数据库中实现的物化视图设计, 到近年来的优化建模方法与学习驱动方法, 相关研究逐步实现了从人工经验到智能决策的转变.

在早期阶段, 代表性系统如 Oracle ECSE[34]、Microsoft Tuning Adivisor[35]、IBM DB2 Advisor[36,37]. Oracle ECSE 通过内建的启发式方法和基于成本的选择策略, 来推荐高质量的物化视图, 并实现了自动化物化视图生成与管理. 微软的 DTA, 发布自 SQL Server 2005, 是一款能够为索引、物化视图以及水平分区提供完整一体化推荐的工具, 推荐机制基于 SQL Server 的 what-if 分析框架[38]. IDB 的 DB2 Advisor 能够推荐物化视图和索引, 结合数据查询优化器和文献 [30] 提出的多查询优化技术来生成候选物化视图, 并给予储存成本和查询收益进行后续评估. 这类方法依赖人工设定的代价模型和阈值, 设计繁琐且难以应对复杂负载下的动态变化, 例如 Oracle ECSE 输入利用 SQL Tunning Set[39]获取的 SQL 语句的文本、SQL-ID、执行计划、优化器估计成本、执行统计信息 (如 CPU 时间、执行时间、缓冲区读取次数、处理的行数) 以及执行上下文, 设计了复杂的启发式算法, 有效提升了查询性能, 但算法中涉及的众多参数需要根据工作负载特征手动调整, 且无法处理复杂的多查询语句.

随后, 研究者开始尝试将自动物化视图选择问题形式化为整数线性规划 (integer linear programming, ILP), 以实现全局最优的候选视图选择, BIGSUBS 方法将共享子表达式选择与重用建模为二分图标记问题 (bipartite graph

labeling), 通过优化求解器在空间约束下选择最大化整体收益的视图集合. 这一形式化建模框架为物化视图选择提供了较为统一的全局优化视角, 为后续研究奠定了坚实基础.

RLView[40]与 AutoView[41]在 BIGSUBS 思路的基础上引入强化学习, 实现了自动物化视图管理. 这类基于强化学习的方法将视图选择与维护过程建模为整数线性规划决策问题, 利用强化学习代理在不断变化的查询负载中自我探索最优策略. RLView 采用 wide-deep 网络预测视图收益, 采用 DQN (deep Q-network) 方法进行视图选择, 而 AutoView 采用 Encoder-Reducer 网络预测视图收益, 采用 DDQN (double deep Q-network) 方法进行视图选择.

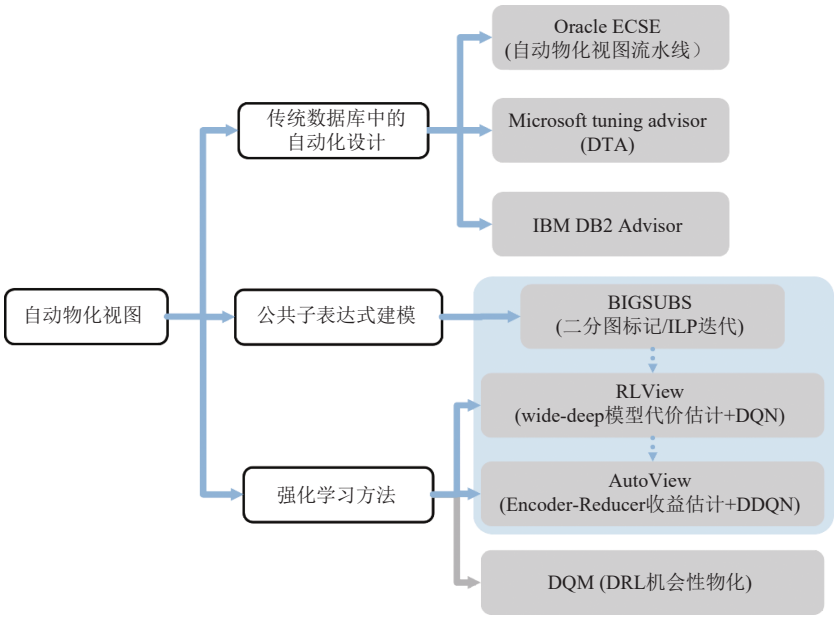

图 6　基于存储重用的多查询优化方法

还有一类基于强化学习方法 DQM[42], 并不是延续 BIGSUBS 的思路, 通过学习历史运行时统计信息来评估物化视图的收益, 采用 DDQN 方法进行视图选择.

总体来看, 自动化物化视图方法的发展经历了从启发式规则到全局优化建模, 再到强化学习驱动的动态决策的演进过程, 为构建具备自适应与自演化能力的智能数据库系统奠定了重要基础.

3.1　基于强化学习和子表达式的方法

基于存储重用的多查询优化方法设计思路通常聚焦于大规模工作负载中的公共子表达式 (common sub-expression), 通过对其进行物化以加速后续查询执行. 这类方法可视为基于公共子表达式的自动物化视图生成, 代表性工作是 BIGSUBS[43].

子表达式选择问题可看作视图选择 (materialized view selection) 问题的一种特化形式——其候选集合由查询逻辑计划中的子树组成. 相比之下, 视图选择的搜索空间更为广泛, 因为视图可以包含未出现在查询计划中的计算逻辑, 这虽然拓展了优化空间, 但也显著增加了查询包含性判断与视图重写的复杂度. 因此, 自动化的子表达式选择成为一种更具工程可行性的简化任务.

如图 7 所示, BIGSUBS 标志着从启发式物化策略向形式化可优化建模范式的过渡. 它通过整数线性规划框架实现了对子表达式选择的全局最优近似, 在传统静态方法与后续学习型优化方法之间起到了关键的承上启下作用. 在此基础上, 后续的 RLView 与 AutoView 等工作进一步将强化学习引入自动物化视图决策, 沿着 BIGSUBS 提出的“子表达式优化-代价约束-收益最大化”范式继续演化, 探索以智能代理替代手工建模与启发式决策的新路径.

图 7　从工业级物化视图设计到强化学习方法

3.1.1　整数线性规划建模范式回顾

在早期研究中, 常见的自动物化方法主要依赖启发式策略来识别高价值子表达式, 例如按照子表达式的出现频次、优化收益总和或“收益与存储开销比”进行排序[44]. 在这些方法下, 系统依序选取排名靠前的子表达式, 直至超出存储预算.

然而, BIGSUBS 摒弃了纯启发式的局部优化思路, 将子表达式选择问题形式化为一个整数线性规划 (ILP) 模型, 并引入迭代求解机制以在近似条件下实现全局最优. 具体而言, BIGSUBS 将查询工作负载建模为一个二分图标记问题.

● 二分图的两个顶点集分别代表查询与候选子表达式.

● 边连接每个查询与执行时会使用的物化子表达式.

模型的求解被分解为两个子任务.

● 顶点标记 (vertex labeling): 决定哪些子表达式应被物化, 即收益估计.

● 边标记 (edge labeling): 确定每个查询在执行时将使用哪些已物化结果, 即视图选择.

通过求解这一图标记问题, 系统可在给定约束下获得最优的物化子表达式集合, 这一建模框架为后续“可学习的物化决策”提供了明确的数学基础. 两个子任务收益估计和视图选择也是这条技术路线下始终考虑的关键任务.

在对子表达式进行价值评估之前, 首先需确定候选集范围. 理论上, 一个查询逻辑计划的任意子树均可视为候选表达式, 但若不加限制, 搜索空间将呈指数级膨胀. BIGSUBS 因此采用了基于优化器输出计划的候选约束策略: 仅从优化器生成的逻辑计划中提取子表达式作为候选集. 这样做有多方面的好处, 能降低搜索空间和求解复杂度, 且候选集生成可独立于数据库内核, 在系统外部实现与集成, 提升灵活性; 同时能利用已有的查询计划签名 (plan signature) 机制, 可快速判定查询等价性, 显著提高匹配效率.

候选集确定后, 关键任务是对子表达式进行价值评估. 例如, 针对 4 个查询 Q_1–Q_4 和两个候选子表达式 S_1、S_2, 若 S_1 出现在全部查询中且包含于 S_2, 而 S_2 仅出现在 Q_3、Q_4 中, 则在覆盖率上 S_1 更广, 但若 S_2 所涵盖的计算量更大, 其潜在收益可能更高. 与此同时, S_2 的存储开销也更大, 因此需要在收益与成本之间进行权衡.

BIGSUBS 从整体收益最大化的角度出发, 定义了单查询收益函数, 即在最优重写条件下, 计算原始子表达式与扫描已物化结果的代价差. 优化目标是在全局存储预算约束下, 选择一组子表达式, 使所有查询的收益总和最大化, 并确保每个查询的所选子表达式之间不存在包含关系.

该优化问题被建模为 ILP, 但当工作负载规模庞大时, 直接求解的计算开销极高. 为此, BIGSUBS 方法提出了迭代近似求解策略.

● 在全局约束条件下, 通过概率方式为顶点打标, 决定子表达式是否物化 (顶点标记).

● 在固定顶点标签的情况下, 求解局部 ILP 子问题, 确定每个查询应使用的物化子表达式 (边标记).

这一过程反复执行, 直到顶点与边标签收敛或达到预设迭代次数.

顶点标签翻转的概率 P_{flip} 由两个因素共同决定.

● P_{capacity}: 当前存储使用量距预算越远, 被翻转概率越高.

● P_{utility}: 子表达式收益越高, 被保留或选中的概率越大.

在边标记阶段, BIGSUBS 方法采用分支定界 (branch-and-bound) 技术加速局部 ILP 的求解. 其核心思想是: 一旦发现某集合中存在包含关系的子表达式, 则无需再考虑其任何超集, 因为该超集必然包含冗余子表达式. 该方法有效裁剪了搜索空间, 提高了求解效率.

3.1.2 ILP 建模思路下基于强化学习的方法

在 BIGSUBS 中, 研究者首次将自动物化视图选择问题从启发式规则转化为可优化求解的形式化模型: 通过整数线性规划统一刻画“候选生成-收益评估-视图选择”流程, 实现对子表达式物化的全局近似最优. 然而, 该范式仍然存在两个限制: 一是收益估计依赖传统代价模型的准确性; 二是视图选择全局搜索开销随候选空间指数增长, ILP/贪心算法易陷入局部最优或代价过高.

基于强化学习的方法正是在此基础上进一步发展. 其核心思想是以学习型方法替代人工代价建模与启发式搜索, 通过与环境交互在试错过程中学习最优物化策略. 总体流程如图 8 所示. 系统从查询输入开始, 首先进行预处理以生成候选视图集合; 随后依次完成两个关键子任务: 收益估计与视图选择. 前者对候选视图的潜在收益进行预测, 决定哪些视图值得物化; 后者将视图选择过程建模为马尔可夫决策过程, 利用强化学习方法做出决策. 最终, 系统将选择结果用于查询重写, 并将执行反馈数据回传, 用于离线模型更新与持续优化.

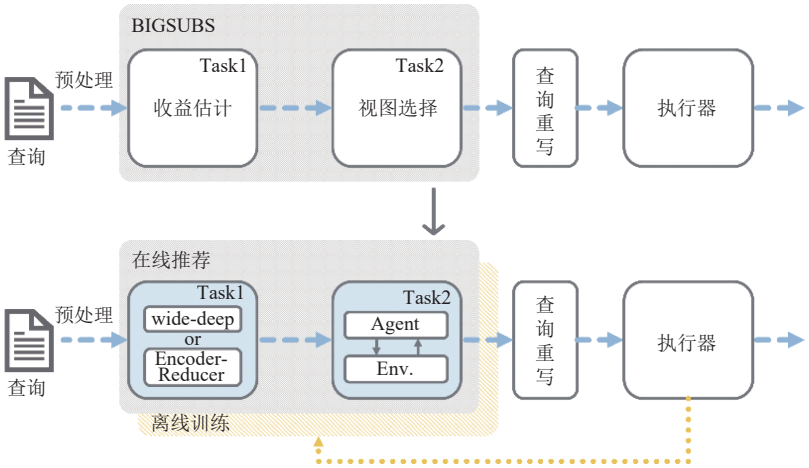

图 8 从 BIGSUBS 到基于强化学习的方法

在这一类方法中, 不同模型对收益估计与策略决策的实现方式各不相同.

● RLView 使用 wide-deep 网络进行收益估计, 并以深度 Q 网络 (DQN)[45,46]实现视图选择.

● AutoView 则进一步采用 Encoder-Reducer 框架提升语义表达能力, 并以双深度 Q 网络 (DDQN)[47]提升策略稳定性与泛化性.

这两种方法共同构成了从“优化建模”向“学习决策”演进的关键阶段, 标志着自动物化视图研究正式进入学习驱动自治优化的新范式.

3.1.2.1 基于 wide-deep 网络 DQN 的强化学习方法

在 BIGSUBS 方法中, 查询代价被定义为执行该查询所需的计算代价, 包括 CPU 使用量与内存使用量. 物化视图的收益可通过使用前后查询代价的差值进行评估; 物化一个视图的开销则由执行该视图所需的计算代价和结果存储成本共同构成. 由此, 可以在一个查询负载下定义一组物化视图的总体收益: 即将每个查询在应用相应物化视图后获得的收益求和, 再减去所有物化视图的总开销. 与 BIGSUBS 的设定一致, RLView 假设视图之间不存在重叠, 即同一查询所使用的视图间没有共享子树.

因此, 任务可描述为: 在给定负载下选择最优的物化视图集合, 使总收益最大化. 朴素做法是针对每个查询遍历全部候选视图并枚举所有组合, 但在大规模负载下该方案计算代价极高, 几乎无法求解. 为此, RLView 引入强化学习模型来近似求解最优解.

如图 8 所示, 输入一个查询工作负载, 首先经过预处理提取候选子查询, 预处理会进行子查询提取, 筛选出所有以 Aggregate、Join 或 Project 运算符为根的子计划, 使用等价检测器[48]识别其中逻辑等价的子计划, 并将等价

子计划聚合成同一簇 (cluster). 对于每个子计划簇, 选择计算代价最低的子计划作为代表候选. 在线推荐阶段中, 基于模型从候选子查询中选择高收益的子查询来生成物化视图, 使用到的模型在离线训练阶段中完成训练.

在线推荐阶段包含两个组件: 收益估计和视图选择. 在收益估计中, 目标是计算物化视图的总收益, 难点在于评估一个查询在应用特定视图集后的查询代价, RLView 方法设计了一个基于深度学习的代价估计模型 (cost estimation model) 来预测该代价. 在视图选择中, 目标是求解视图选择问题, 作者设计了一个基于强化学习的视图选择模型, 根据已计算的收益推荐最优子查询用于生成物化视图.

离线训练阶段负责训练上述两个模型. 查询计划、物化视图、表信息 (如表大小) 以及重写查询的实际执行成本均可从查询引擎中收集并存入元数据库. 收益估计模型训练使用每个查询对应的查询计划、物化视图以及相关表信息作为输入特征, 将重写查询的实际成本作为目标值进行训练. 视图选择模型训练首先利用重写查询的实际成本计算查询与视图的实际收益, 再基于该收益计算不同状态之间的中间奖励, 用于强化学习模型的微调.

3.1.2.1.1 特征提取 & 基于 wide-deep 模型的收益估计

特征提取包含查询计划特征和相关表的特征. 查询计划特征的提取包含两个层面, 首先筛选出查询计划中所有操作符 (如 Scan、Projection、Filter、Join 等), 并用前缀表示法表示所有操作符和其对应属性; 再记录该查询计划和其子查询计划分别包含那些操作符. 相关表的特征则是从源数据库中提取的信息, 包含各种表结构信息这类非数值特征, 以及统计信息这类数值特征.

代价估计模型则采用 wide-deep 模型[49]架构, 这是一种兼顾模型记忆能力和泛化能力的方法. 该模型由两部分组成, 宽线性模型部分用来捕捉特征与结果之间的线性关系, 深度模型用来捕捉输入特征与结果之间的非线性关系, 前者保证了模型良好的泛化性与稳定性, 后者则提升了模型对复杂查询计划结构、表结构的建模能力. 模型输入分为两部分, 数值特征和非数值特征. 因为非数值特征是离散的, 其取值变化与结果之间的关系往往是非线性的, 所以在宽线性模型中, 只考虑来自元数据库的数值特征, 先对所有数值特征进行标准化, 并拼接成一个固定的长度, 然后用仿射变换模型对标准化向量进行线性变换. 深度模型的输入则由来自元数据库的表结构编码和提取自查询计划的计划序列编码拼接构成, 接着使用两个残差网络实现模型建模. 最后, 模型通过一个回归器将两个模型的输出结合起来, 输出最终的预测结果.

3.1.2.1.2 基于 DQN 的视图选择

延续 BIGSUBS 的思路, RLView 将物化视图选择问题建模为一个整数线性规划 (ILP) 问题, 并进一步转化为马尔可夫决策过程 (MDP), 利用深度强化学习进行求解. 该任务包括两个子问题.

- 选择最优子查询 (物化视图) 进行物化.
- 为每个查询选择最优的视图组合.

BIGSUBS 采用的迭代优化方式在不同目标间交替求解, 但缺乏“记忆能力”: 每次迭代仅能获得局部最优结果, 且各轮迭代间无法共享反馈信息, 易导致结果震荡. RLView 通过强化学习机制解决此问题, 使系统能够从历史决策中学习并持续优化策略.

如图 9 所示该方法中设计的强化学习定义如下.

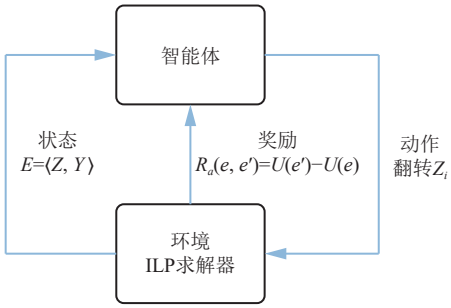

图 9 RLView 方法中的强化学习

● 状态 $E=\langle Z,Y\rangle$: 定义 Z 和 Y 的元组, Z 表示最优的物化视图集合, Z_j 为 1 表示第 j 个视图被物化, 为 0 则不被物化, Y 表示每个查询的视图选择情况, Y_{ij} 为 1 表示第 i 个查询选择使用第 j 个被物化的视图.

● 动作 A: 翻转所选变量 z_i 的标签, 即改变视图的物化状态.

● 环境: 整数线性规划求解器. 执行动作后得到 Z 的新标签, 然后用求解器求解 Y 的新标签, 从而获得新状态 e_i.

● 奖励 $R_a(e,e')=U(e')-U(e)$: 即两个状态收益的变化, 其中 $U(e)$ 定义为状态 e 对应的收益.

● 智能体: 强化学习模型 RLView.

● 策略函数 $P_a(e,e')=P_r(e_{t+1}=e'\mid e_t=e,a_t=a)$ 是在 t 时刻, 状态 e 采取动作 a 转移到 $t+1$ 时刻 e' 的概率.

因此, 整数线性规划问题转化为一个强化学习问题, 目标是学习一个最优策略 $\pi^*=\arg\max_{\pi}E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}\mid e_t=e\right\}$, $\forall e\in E$, $\forall t\geqslant 0$, 其中 π: $E\times A\to[0,1]$ 表示一个策略函数, E_{π} 表示在策略 π 下的期望值, $\gamma\in[0,1]$ 是折扣率 (discount rate), k 是时间步, r_{t+k} 是 $t+k$ 时刻的中间奖励.

RLView 方法采用经典的 Q-learning 算法, 但视图选择问题的状态空间和动作空间都非常庞大, 想建立并维护一个对应的 Q-table 几乎是无法实现的. 为此, RLView 使用 DQN 以神经网络近似 Q 函数: $Q(e,a)=\mu(e,a\mid\theta)$.

由此就可以得到状态 e 的 Q 值向量 $Q(e)=[Q(e,a_1),\ldots,Q(e,a_n)]\in R^n$, 通过比较 $Q(e)$ 中所有维度的值, 选取最大值维度对应的动作为状态 e 对应的动作. 为提高收敛速度与稳定性, RLView 方法利用 BIGSUBS 的迭代优化结果作为初始解对 DQN 模型进行预训练, 从而缩短学习周期并减少探索成本.

3.1.2.2 基于 Encoder-Reducer 网络与 DDQN 的强化学习方法

文献 [41] 在 RLView 的基础上提出了一个自治物化视图管理系统, 通过分析查询工作负载, 估计将查询物化为视图的成本和收益, 并在空间预算内选择物化视图以实现收益最大化. 该方法使用了 DDQN 模型来选择高质量的物化视图, 并通过 Encoder-Reducer 架构模型的输出丰富状态表示.

如图 8 所示, 整体流程与 RLView 基本一致, 其中收益估计部分 AutoView 设计了一个基于 Encoder-Reducer 架构的循环神经网络 (RNN) 模型, 用于将查询和视图嵌入为向量并预测视图的收益. 编码器 (encoder) 将查询编码为一个语义向量, 而规约器 (Reducer) 则从查询中归约 (reduce) 视图并预测收益. 视图的收益为查询执行减少时间减去视图物化成本, 这二者都由 Encoder-Reducer 模型进行估计. 视图选择则基于 DDQN 方法, 设计了一个强化学习模型 ERDDQN (Encoder-Reducer double deep Q-learning network) 来选择物化视图集, 并使用 ERDDQN 模型选择每个查询对应物化视图进行查询重写.

AutoView 中的视图候选生成策略思路与 RLView 相似, 但处理方式有所差异. 系统首先从数据库优化器输出的树形查询计划中检测查询间的公共子树, 并定义: 若两个子树在连接、选择和投影条件上完全一致, 且子节点相同, 则判定这两个子树等价. 通过合并所有等价子树, 可获得若干由等价性合并而成的查询计划集合. 随后, 计算每个子树的收益, 其定义为包含该子树的查询数量与估计收益的乘积. 最终, 选取收益最高的一组子树作为物化视图候选集, 从而在保证代表性的同时显著减少候选空间.

3.1.2.2.1 特征提取&基于 Encoder-Reducer 模型的收益估计

AutoView 中设计了一个 Encoder-Reducer 模型来对候选的物化视图进行收益估计. Encoder-Reducer 不同于传统的 Encoder-Decoder 模型 (用于语言翻译), Encoder 将 q 编码为一个语义向量, 而 Reducer (归约器) 则用语义向量和视图特征进行归约并预测收益.

由于深度网络仅能接受张量输入, AutoView 采用序列化编码方法, 将 SQL 查询计划树转换为张量表示, 同时保留结构与语义信息. 系统通过后序遍历对查询计划树进行序列化, 其顺序与查询引擎执行顺序一致. 序列化后, 模型将每个节点 (操作符) 编码为定长张量.

Encoder 单元包括嵌入层 (embedding layer)、门控循环单元 (GRU) 及两个输出层, 最后一次循环的隐藏状态被视为整个查询的编码表示, 其输出为查询执行时间与基数估计.

Reducer 单元包括嵌入层、GRU、线性层、ReLU 层及输出层, 最后一次循环输出为使用相应视图集合回答

查询时的执行时间估计.

为了进一步提高模型精度, AutoView 在 Reducer 阶段引入多头注意力机制, 捕获查询计划树与视图计划树节点间的三类关键关系: 节点相似性 (node similarity): 表示物化视图节点与查询节点输出相同结果, 有助于估计可复用计算比例; 节点冲突 (node conflict): 表示查询与视图谓词冲突, 用于识别物化带来的负面影响; 节点重排序 (node reordering): 表示物化可能导致连接顺序变化, 从而影响最优计划.

在 Reducer 单元中, 输入的嵌入和隐藏状态将与编码器中节点的隐藏状态进行比较, 以找到相关的子查询. 然后, 相关子查询的隐藏状态将用于直接增强重写查询的估计, 而无需长距离信息传递. 为捕捉节点间的不同关系, 使用多个投影矩阵 W_i^Q 变换查询节点的隐藏状态, 并用 W_i^K 变换物化视图节点的隐藏状态. 通过不同的投影矩阵, 模型可以从不同的表示子空间捕获查询节点的信息.

3.1.2.2.2 基于 DDQN 的视图选择

物化视图问题的描述与 RLView 一致, 将其建模为一个整数线性规划问题, 求解在一定空间约束前提下, 一个物化视图候选子集以最大化总收益. 使用 Encoder-Reducer 收益估计器来估计每个物化视图的执行时间和空间成本, 如果对所有物化视图候选逐一计算, 时间复杂度为 $O(n^2)$, n 为内部节点数量, 计算成本比较大. AutoView 方法将所有查询的根节点通过一个超级根节点连在一起, 再后序遍历整个查询计划图, 在每个内部节点上保存模型的中间输出, 就以 $O(n)$ 的方式估算了所有物化视图候选的执行时间和空间成本. 更进一步, AutoView 方法为每个物化视图候选计算一个分数 $w_v = \frac{f_v \times (t_v - t_{\text{scan}}^v)}{|v|}$, f_v 是子树出现频率, $t_v - t_{\text{scan}}^v$ 是估计收益, t_{scan}^v 是从磁盘扫描结果的时间, $|v|$ 是空间成本, 保留具有较高分数的物化视图候选, 直到总空间成本超过预算为止.

有了物化视图集, 需要在当前物化视图及下求解整数线性规划问题, 如图 10 所示, AutoView 方法使用深度强化学习中的 DDQN 模型来求解此问题.

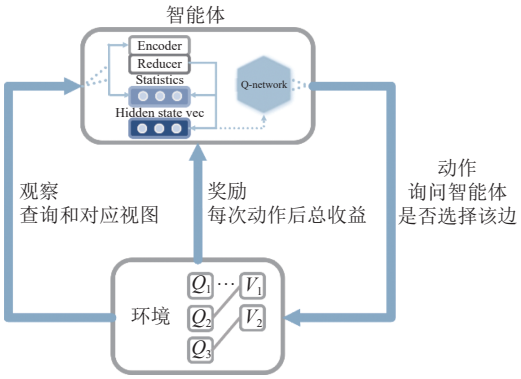

图 10 AutoView 方法中的强化学习

● 环境: MV 选择状态被建模为二分图 (bipartite graph): 左侧节点代表查询集合 Q, 右侧节点代表视图集合 V, 它们之间的边表示 $e_{i,j}$ (即查询 q_i 是否使用视图 v_j). 环境模块为智能体提供观测值, 例如选择状态和总收益.

● 状态: 包括从环境观测中提取的特征, 和 Encoder-Reducer 模型输出的语义向量.

● 智能体: 由两个神经网络和经验回放机制 (experience replay mechanism) 组成. 这两个网络 (Q-网络和目标网络) 近似于动作-值函数 $W(s,a)$, 表示在状态 s 选择动作 a 后可以获得的最大反馈.

● 奖励: 每次动作后总收益的变化.

● 动作: 在每次迭代中, 环境选择一条边, 并询问智能体是否使用这条边. 使用则将 $e_{i,j}$ 置 1, 反之置 0.

● 策略: 智能体根据获取更高反馈的策略行事, 反复反馈最高的动作.

训练过程不断迭代并更新 Q-network 参数, 使其逼近真实的动作-值函数.

在查询重写阶段, 系统需在已物化视图集合中为新查询选择最优子集以最小化执行时间. 该任务可视为视图选择问题的特例: 工作负载仅包含一个查询, 视图集为已物化集合, 空间预算无限, 目标为最大化单查询收益. 因

而, ERDDQN 模型可直接复用, 为查询提供最优重写方案.

为了加速 Encoder-Reducer 模型收益估计效率, 在进行收益估计期间缓存隐藏状态向量和注意力值, 这样在不同动作输入时, 减少动作中重复出现视图的计算量, 例如 3 个动作 $\{v_1\}$、$\{v_1,v_2\}$、$\{v_1,v_3\}$, 无缓存的情况下, 如果单独估计这 3 个动作, 查询 q 被输入 Encoder 模型 3 次, 视图 v_1 将被 Reducer 模型计算 3 次. 而有缓存的情况下, 记录下查询 q 的隐藏状态向量, 查询 q 只需要被 Encoder 计算一次, 且视图 v_1 也只被 Reducer 计算一次.

3.2 基于强化学习和自适应的机会性方法

“机会性物化 (opportunistic materialization)”一词最早用于描述 Hive、Pig 等大规模数据处理系统中的自动持久化机制[50]. 它指代由查询执行过程自然产生的物化操作, 而非由数据库管理员显式定义. DQM 借用了这一概念, 用于刻画在查询运行时动态决定物化与否的机制.

传统的基于存储的多查询优化方法中, 视图推荐系统通常依赖历史查询负载、数据库模式及代价模型来选择最优视图集合[37,39,51]. 这类方法属于回顾性 (retrospective) 策略, 假设未来查询与历史模式相似. 然而, 当负载频繁变化、出现临时查询 (ad-hoc query) 或存储约束调整时, 这种静态策略往往性能显著下降. 理想的动态系统应能通过“淘汰”与“重建”操作持续适应环境变化.

现有的基于存储的多查询优化方法多基于启发式规则, 例如 LRU、LFU[17,19,52]或基于代价模型的评分策略[51]. 这些方法缺乏普适性: LRU 容易出现顺序泛洪 (sequential flooding), 而基于代价模型的策略则受限于优化器估计精度, 可能反而降低性能. 不同负载下难以预先确定最优启发式. 相比之下, DQM 通过引入强化学习 (reinforcement learning) 实现自适应的决策更新, 摆脱了固定启发式的限制.

相较于回顾性方法, DQM 克服了其对历史负载的过度依赖, 能够灵活应对即时查询中常见的负载偏移与动态驱逐需求, 避免了定期重新运行视图推荐工具带来的高额维护成本. 而相较于传统的反应式启发式方法, 如仅关注访问热度的 LRU 或 LFU 策略, DQM 摆脱了规则的限制与代价模型估算偏差的限制. 它通过直接利用真实执行时间作为反馈信号, 在性能收益与物化成本之间建立了更深层的逻辑关联, 从而在复杂多变的 OLAP 负载下展现出更强的稳健性与适应力.

DQM 方法的核心贡献在于其精巧的决策模型及其与 SparkSQL 紧集成的系统框架设计. 该系统主要解决了两个问题: 如何精准筛选待物化视图, 以及如何动态确定视图的生存周期.

在物化视图的选择层面, DQM 创新性地将该任务建模为强化学习过程. 针对数据库环境下难以在单次执行中同时获取物化前后对比数据、从而导致回报信号 (reward) 缺失的难题, DQM 引入了反事实实验 (counter-factual experiment) 机制. 该机制通过在后台空闲时段异步运行“成对实验”, 对比同一查询在有无视图支持下的真实执行差异, 从而精确量化视图的边际效用. 这种基于真实运行耗时的反馈回路, 使系统能够摆脱对传统代价模型 (cost model) 误差的依赖.

在视图驱逐策略方面, DQM 并未尝试将复杂的存储硬约束直接嵌入强化学习的动作空间, 而是采用了一种解耦的“顺从型”硬约束算法. 与忽略获取成本的传统 LRU 算法不同, DQM 充分考虑了视图的性能增益与其重构成本之间的权衡. 系统通过维护一张动态信用表, 使每个视图的信用评分由累计延迟减少量与物化成本共同决定, 并辅以衰减因子以应对负载偏移. 当存储空间达到阈值时, 系统通过逐出信用分最低的视图, 确保了高价值、高成本的中间结果能够获得更长的生存周期, 从而显著提升了复杂 OLAP 环境下的系统鲁棒性.

与 RLView 和 AutoView 不同, DQM 主要解决的并不是收益估计和视图选择两个任务, 因为是考虑自适应的实时更新, 只需要考虑当前输入查询中存在的候选视图是否需要物化. 它没有沿用 BIGSUBS 的建模思路, 而是将决定视图是否创建的决策过程建模为马尔可夫决策过程, 在给定系统状态和查询上下文的情况下, 学习最优的视图创建策略. 此外, BIGSUBS 代表的技术路线中将有限内存约束建模在 ILP 问题之中, 而 DQM 则还需要一个额外的视图淘汰机制来进行缓存管理.

3.3 小　结

基于存储的多查询优化研究的技术脉络可概括为 3 步演进: 启发式顾问 (如 Oracle ECSE、Microsoft DTA、

DB2 Advisor): 基于规则与代价模型自动推荐, 但对动态负载适应性有限. 优化建模 (如 BIGSUBS): 将候选选择形式化为 ILP/图标记问题, 在预算约束下追求全局最优近似. 学习驱动 (如 RLView、AutoView、DQM): 用深度模型做收益估计, 用 DQN/DDQN 等强化学习进行策略决策; DQM 进一步支持运行时“机会性物化”, 随负载变化自适应创建与淘汰. 自动物化视图的一个典型流程包含: 候选生成、收益估计、视图选择、查询重写与反馈. 总体目标是在性能收益与维护开销之间取得平衡.

通过对上述基于存储重用的多查询优化方法的系统梳理 (见表 2), 可以发现该领域正经历从“经验驱动的启发式推荐”到“严谨数学建模”, 再到“在线自治演化”的范式转变. 从技术演进脉络分析, 早期奠基性研究如 Oracle ECSE、Microsoft DTA 和 IBM DB2 Advisor 首次实现了工业级数据库中的自动化视图管理, 通过内建的代价模型与 what-if 分析框架, 将视图选择从纯人工经验转向了基于成本的初步自动化. 尽管此类系统在生产环境下展现了极高的稳健性, 但其核心仍依赖大量人工设定的启发式规则与调优参数, 在应对现代大规模动态负载时表现出一定的迟滞性.

表 2　基于存储重用的多查询优化方法比较

类别	方法	解决问题	核心技术	适用场景	优点	缺点
奠基性研究	Oracle ECSE、MS DTA、DB2 Advisor等	商业数据库中的自动化索引与视图推荐	基于代价的启发式算法、what-if 分析框架	生产环境下的常规事务与分析负载	工业级稳健性, 集成于数据库内核	依赖人工设定的代价模型与阈值, 参数调整繁琐
静态/周期性物化	BIGSUBS	全局存储预算下的公共子表达式选择	整数线性规划 (ILP)、二分图标记	负载稳定、查询模式可预测的离线数据	建模思路的开创性, 它通过将视图选择问题巧妙地转化为二分图标记下的整数线性规划	无法处理临时查询; 在大规模负载下 ILP 求解极慢
	RLView	传统代价模型不准及大规模搜索空间爆炸	wide-deep 模型 (收益估计)+DQN (视图选择)	具有历史负载参考的复杂 OLAP 环境	兼顾线性关系与非线性特征, 决策自动化程度高	依赖历史负载的相似性, 难以应对负载漂移
	AutoView	实现一个自治物化视图管理系统	Encoder-Reducer架构+多头注意力+DDQN	结构复杂、多表连接频繁的分析任务	语义表征能力极强, 能捕获谓词冲突与重排序影响, 系统自治化	模型结构复杂, 训练与推理开销较大
动态/机会性物化	DQM	临时查询 (ad-hoc) 的实时响应与动态存储管理	强化学习 (DDQN)+实时反馈循环	工作负载频繁波动、对响应速度要求极高的环境	具备自适应淘汰机制, 能在线感知负载变化	初期探索阶段性能不稳定; 对系统运行时监控要求高

随后, BIGSUBS 主要聚焦于搜索空间膨胀问题, 试图在给定预算下通过整数线性规划 (ILP) 寻找全局收益最优子集; 而以 RLView、AutoView 为代表的 AI 赋能方法则进一步解决了传统代价模型对复杂查询估计不准的痛点, 利用深度学习 (如 wide-deep、Transformer 等模型) 强化了对查询语义及执行开销的表征能力. 从技术演进脉络分析, 静态物化路线侧重于利用历史负载进行精细化建模, 其优点在于收益预测精度高, 适用于工作负载稳定且查询复杂度极高的企业级数仓场景, 但其局限性在于难以应对负载漂移且重新训练成本较高; 相比之下, 以 DQM 为代表的机会性物化技术则侧重于自适应决策, 利用强化学习实现在线感知与动态淘汰, 更适用于云原生数据库等具有高度不确定性的临时查询场景. 本文认为, 未来的研究重心将不再局限于单一维度的优化, 而是向“跨层协同”与“轻量化自治”方向发展: 一方面, 应打破执行期计算重用与存储期结果持久化的边界, 在统一的收益模型下实现协同调度; 另一方面, 针对内核集成需求, 急需开发轻量级、高可解释性的数据库专用神经元, 以平衡 AI 决策的智能化水平与系统运行的实时性开销.

4 总结与展望

本文综述了近年来多查询优化技术的发展脉络, 系统总结了基于计算重用的多查询优化与基于存储重用的多查询优化两类技术路线在研究与工程实践中的演进过程, 重点分析了传统启发式方法、基于自适应处理的优化框架以及 AI 赋能的学习型方法的核心思想与关键进展.

在传统阶段, 基于计算重用的多查询优化主要依赖规则与启发式策略, 通过识别并共享公共子表达式、中间结果或算子实现计算重用. 离线优化方法 (如 SWO、SharedDB、MQJoin) 能够全面捕捉共享机会但优化开销高; 在线优化方法 (如 QPipe、DataPath、Recycler) 具备较好的实时性但共享范围有限. 该阶段的研究主要关注共享检测与缓存管理, 但难以应对动态负载与复杂并发场景.

随着自适应查询处理 (adaptive query processing) 技术的发展, 研究者提出了能够在运行时动态调整执行计划的框架. RouLette 系统通过强化学习驱动的自适应调度, 将“优化-执行”范式转化为“执行-学习-优化”闭环, 实现了对共享机会的动态探索与利用. 其结合对称哈希连接、涡流算子 (eddy) 与状态模块 (state module) 等机制, 有效提升了在未知统计信息环境下的性能稳定性.

在深度学习赋能阶段, 研究进一步将学习模型引入查询优化决策. 基于 Transformer 的 MQFormer 模型通过注意力机制捕捉并发查询之间的相关性和资源冲突, 实现了对延迟与共享收益的高精度预测. 此类方法体现了由“启发式决策”向“价值学习驱动”的转变, 使系统能够自动发现并维护最优共享结构.

在基于存储重用的多查询优化方向, Oracle 的 ECSE 及 BIGSUBS 等方法在候选生成、收益估计和启发式剪枝上取得了工程化进展; 而 RLView、AutoView、DQM 等系统则利用强化学习将物化视图生成、收益预测与查询重写统一建模, 显著提高了策略的泛化性与自适应性, 形成了端到端的自治优化框架.

综合来看, AI 赋能的多查询优化技术呈现出以下趋势: 从规则驱动到学习驱动: 通过强化学习与深度神经网络实现策略自动化, 减少专家依赖; 从静态优化到动态自适应: 支持运行时反馈更新, 实现在线优化与负载自适应; 从单层优化到全局协同: 将查询优化、执行调度、缓存管理与物化视图统一在同一收益模型下协调决策.

最后, 本文对 AI 赋能的多查询优化技术做出以下展望.

● 基于计算重用方法的深化: 针对以 RouLette 和 Lemo 为代表的执行期基于计算重用的多查询优化技术, 未来的核心挑战在于决策的实时性与表征的深度融合. 一方面, 现有模型在处理海量并发查询时, 其推理延迟往往会抵消计算重用带来的增益. 因此, 开发与数据库执行引擎深度耦合的轻量化神经算子, 实现快速语义匹配与共享决策, 是提升实时性的关键. 另一方面, 现有的表征模型仍存在感知力不足或代价过高的问题. 未来的研究应探索如何将数据特征与查询语义进行多模态融合, 以在更深层次上识别隐含的计算重用机会.

● 基于存储重用方法的演进: 对于以 DQM 和 AutoView 为代表的基于存储重用的自动物化视图管理技术, 未来的演进重点在于训练的稳定性与开销权衡. 目前的强化学习模型在冷启动阶段往往收敛较慢, 且存在难以解决数据迁移的问题. 未来的方向应聚焦于适应动态负载与主动预热机制的结合, 减少对任务相关训练数据的依赖.

● 跨层级协同与通用工程架构: 从宏观视角来看, 打破基于计算重用和基于存储重用方法之间的“技术孤岛”是实现全局性能的一个思路, 未来的研究应尝试构建统一的 AI 驱动框架, 在同一个代价评估体系下动态权衡“瞬时计算重用”与“长期物理存储”的收益, 实现资源在算力与存储间的优化配置.

● 标准化与可迁移性: 学习驱动的多查询优化仍依赖于特定系统、特定负载或特定模型假设. 强化学习模型往往需要大量任务相关的训练数据才能收敛, 其策略难以在不同数据库引擎间直接复用. 缺乏具备跨系统、跨负载、跨模型可迁移能力的通用框架. 针对目前原型系统接口不一、难以横向对比的问题, 急需建立统一的基准测试平台 (benchmark suite) 与开源实验框架, 为研究的可复现性提供客观评价体系

综上所述, AI 赋能的多查询优化技术正在推动数据库查询优化从启发式与静态调优阶段迈向学习驱动与自适应自治阶段. 其核心目标是通过智能化的共享检测、收益评估与策略学习, 在保证系统稳定性与可解释性的同时, 实现高效、通用、可扩展的多查询优化与物化视图管理, 为未来智能数据库系统的发展奠定基础.

References

[1] Sun LM, Zhang SM, Ji T, Li CP, Chen H. Survey of data management techniques powered by artificial intelligence. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 600–619 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5909.htm [doi: 10.13328/j.cnki.jos.005909]

[2] Wei JH, Xia YF, Gong XQ. Review of research on multi-query sharing technology. Ruan Jian Xue Bao/Journal of Software, 2021, 32(10): 3176–3202 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6203.htm [doi: 10.13328/j.cnki.jos.006203]

[3] Harizopoulos S, Shkapenyuk V, Ailamaki A. QPipe: A simultaneously pipelined relational query engine. In: Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM, 2005. 383–394. [doi: 10.1145/1066157.1066201]

[4] Nagel F, Boncz P, Viglas SD. Recycling in pipelined query evaluation. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering. Brisbane: IEEE, 2013. 338–349. [doi: 10.1109/ICDE.2013.6544837]

[5] Arumugam S, Dobra A, Jermaine CM, Pansare N, Perez L. The DataPath system: A data-centric analytic processing engine for large data warehouses. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. Indianapolis: ACM, 2010. 519–530. [doi: 10.1145/1807167.1807224]

[6] Candea G, Polyzotis N, Vingralek R. A scalable, predictable join operator for highly concurrent data warehouses. Proc. of the VLDB Endowment, 2009, 2(1): 277–288. [doi: 10.14778/1687627.1687659]

[7] Madden S, Shah M, Hellerstein JM, Raman V. Continuously adaptive continuous queries over streams. In: Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 49–60. [doi: 10.1145/564691.564698]

[8] Giannikis G, Makreshanski D, Alonso G, Kossmann D. Shared workload optimization. Proc. of the VLDB Endowment, 2014, 7(6): 429–440. [doi: 10.14778/2732279.2732280]

[9] Giannikis G, Alonso G, Kossmann D. SharedDB: Killing one thousand queries with one stone. Proc. of the VLDB Endowment, 2012, 5(6): 526–537. [doi: 10.14778/2168651.2168654]

[10] Makreshanski D, Giannikis G, Alonso G, Kossmann D. MQJoin: Efficient shared execution of main-memory joins. Proc. of the VLDB Endowment, 2016, 9(6): 480–491. [doi: 10.14778/2904121.2904124]

[11] Sioulas P, Ailamaki A. Scalable multi-query execution using reinforcement learning. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 1651–1663. [doi: 10.1145/3448016.3452799]

[12] Avnur R, Hellerstein JM. Eddies: Continuously adaptive query processing. In: Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. Dallas: ACM, 2000. 261-272. [doi: 10.1145/342009.335420]

[13] Raman V, Deshpande A, Hellerstein JM. Using state modules for adaptive query processing. In: Proc. of the 19th Int'l Conf. on Data Engineering. Bangalore: IEEE, 2003. 353–364. [doi: 10.1109/ICDE.2003.1260805]

[14] Finkelstein S. Common expression analysis in database applications. In: Proc. of the 1982 ACM SIGMOD Int'l Conf. on Management of Data. Orlando: ACM, 1982. 235–245. [doi: 10.1145/582353.582400]

[15] Sellis TK. Intelligent caching and indexing techniques for relational database systems. Information Systems, 1988, 13(2): 175–185. [doi: 10.1016/0306-4379(88)90014-2]

[16] Ivanova MG, Kersten ML, Nes NJ, Gonçalves RAP. An architecture for recycling intermediates in a column-store. In: Proc. of the 2009 ACM SIGMOD Int'l Conf. on Management of Data. Providence Rhode: ACM, 2009. 309–320. [doi: 10.1145/1559845.1559879]

[17] Kotidis Y, Roussopoulos N. DynaMat: A dynamic view management system for data warehouses. In: Proc. of the 1999 ACM SIGMOD Int'l Conf. on Management of Data. Philadelphia: ACM, 1999. 371–382. [doi: 10.1145/304182.304215]

[18] Roy P, Ramamritham K, Seshadri S, Shenoy P, Sudarshan S. Don't trash your intermediate results, cache 'em. arXiv:cs/0003005, 2000.

[19] Scheuermann P, Shim J, Vingralek R. WATCHMAN: A data warehouse intelligent cache manager. In: Proc. of the 22nd Int'l Conf. on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 1996. 51–62.

[20] Shim J, Scheuermann P, Vingralek R. Dynamic caching of query results for decision support systems. In: Proc. of the 11th Int'l Conf. on Scientific and Statistical Database Management. Cleveland: IEEE, 1999. 254–263. [doi: 10.1109/SSDM.1999.787641]

[21] Tan KL, Goh ST, Ooi BC. Cache-on-demand: Recycling with certainty. In: Proc. of the 17th Int'l Conf. on Data Engineering. Heidelberg: IEEE, 2001. 633–640. [doi: 10.1109/ICDE.2001.914878]

[22] Mo SS, Chen YL, Wang H, Cong G, Bao ZF. Lemo: A cache-enhanced learned optimizer for concurrent queries. Proc. of the ACM on Management of Data, 2023, 1(4): 247. [doi: 10.1145/3626734]

[23] Zhao Y, Cong G, Shi JC, Miao CY. QueryFormer: A tree transformer model for query plan representation. Proc. of the VLDB Endowment, 2022, 15(8): 1658–1670. [doi: 10.14778/3529337.3529349]

[24] Marcus R, Negi P, Mao HZ, Zhang C, Alizadeh M, Kraska T, Papaemmanouil O, Tatbul N. Neo: A learned query optimizer. Proc. of the VLDB Endowment, 2019, 12(11): 1705–1718. [doi: 10.14778/3342263.3342644]

[25] Yang ZH, Chiang WL, Luan SF, Mittal G, Luo M, Stoica I. Balsa: Learning a query optimizer without expert demonstrations. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 931–944. [doi: 10.1145/3514221.3517885]

[26] Marcus R, Negi P, Mao HZ, Tatbul N, Alizadeh M, Kraska T. Bao: Making learned query optimization practical. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 1275–1288. [doi: 10.1145/3448016.3452838]

[27] Goldstein J, Larson PÅ. Optimizing queries using materialized views: A practical, scalable solution. ACM SIGMOD Record, 2001, 30(2): 331–342. [doi: 10.1145/376284.375706]

[28] Gupta H, Mumick IS. Selection of views to materialize under a maintenance cost constraint. In: Proc. of the 7th Int'l Conf. on Database Theory. Jerusalem: Springer, 1999. 453–470. [doi: 10.1007/3-540-49257-7_28]

[29] Kathuria T, Sudarshan S. Efficient and provable multi-query optimization. In: Proc. of the 36th ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems. Chicago: ACM, 2017. 53–67. [doi: 10.1145/3034786.3034792]

[30] Lehner W, Cochrane B, Pirahesh H, Zaharioudakis M. fAST refresh using mass query optimization. In: Proc. of the 17th Int'l Conf. on Data Engineering. Heidelberg: IEEE, 2001. 391–398. [doi: 10.1109/ICDE.2001.914852]

[31] Roy P, Seshadri S, Sudarshan S, Bhobe S. Efficient and extensible algorithms for multi query optimization. ACM SIGMOD Record, 2000, 29(2): 249–260. [doi: 10.1145/335191.335419]

[32] Silva YN, Larson PA, Zhou JR. Exploiting common subexpressions for cloud query processing. In: Proc. of the 28th IEEE Int'l Conf. on Data Engineering. Arlington: IEEE, 2012. 1337–1348. [doi: 10.1109/ICDE.2012.106]

[33] Zhou JR, Larson PA, Freytag JC, Lehner W. Efficient exploitation of similar subexpressions for query processing. In: Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of Data. Beijing: ACM, 2007. 533–544. [doi: 10.1145/1247480.1247540]

[34] Ahmed R, Bello R, Witkowski A, Kumar P. Automated generation of materialized views in Oracle. Proc. of the VLDB Endowment, 2020, 13(12): 3046–3058. [doi: 10.14778/3415478.3415533]

[35] Agarwal S, Chaudhuri S, Kollar L, Marathe A, Narasayya V, Symala M. Database tuning advisor for Microsoft SQL Server 2005: Demo. In: Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM, 2005. 930–932. [doi: 10.1145/1066157.1066292]

[36] Zilio DC, Rao J, Lightstone S, Lohman G, Storm A, Garcia-Arellano C, Fadden S. DB2 design advisor: Integrated automatic physical database design. In: Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: VLDB Endowment, 2004. 1087–1097.

[37] Zilio DC, Zuzarte C, Lightstone S, Ma WB, Lohman GM, Cochrane RJ, Pirahesh H, Colby L, Gryz J, Alton E, Valentin G. Recommending materialized views and indexes with the IBM DB2 design advisor. In: Proc. of the 2004 Int'l Conf. on Autonomic Computing. New York: IEEE, 2004. 180–187. [doi: 10.1109/ICAC.2004.1301362]

[38] Chaudhuri S, Narasayya V. AutoAdmin "what-if" index analysis utility. In: Proc. of the 1998 ACM SIGMOD Int'l Conf. on Management of Data. Seattle: ACM, 1998. 367–378. [doi: 10.1145/276304.276337]

[39] Dageville B, Das D, Dias K, Yagoub K, Zait M, Ziauddin M. Automatic SQL tuning in Oracle 10g. In: Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: VLDB Endowment, 2004. 1098–1109.

[40] Yuan HT, Li GL, Feng L, Sun J, Han Y. Automatic view generation with deep learning and reinforcement learning. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering. Dallas: IEEE, 2020. 1501–1512. [doi: 10.1109/ICDE48307.2020.00133]

[41] Han Y, Li GL, Yuan HT, Sun J. AutoView: An autonomous materialized view management system with encoder-reducer. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(6): 5626–5639. [doi: 10.1109/TKDE.2022.3163195]

[42] Liang X, Elmore AJ, Krishnan S. Opportunistic view materialization with deep reinforcement learning. arXiv:1903.01363, 2019.

[43] Pei J, Amer-Yahia S, Jindal A, Karanasos K, Rao S, Patel H. Selecting subexpressions to materialize at datacenter scale. Proc. of the VLDB Endowment, 2018, 11(7): 800–812. [doi: 10.14778/3192965.3192971]

[44] Gunda PK, Ravindranath L, Thekkath CA, Yu Y, Zhuang L. Nectar: Automatic management of data and computation in datacenters. In: Proc. of the 9th USENIX Conf. on Operating Systems Design and Implementation. Vancouver: USENIX Association, 2010. 75–88.

[45] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. arXiv:1312.5602, 2013.

[46] Wang ZY, Schaul T, Hessel M, van Hasselt H, Lanctot M, de Freitas N. Dueling network architectures for deep reinforcement learning. In: Proc. of the 33rd Int'l Conf. on Machine Learning. New York: JMLR.org, 2016. 1995–2003.

[47] van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double Q-learning. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. Phoenix: AAAI Press, 2016. [doi: 10.1609/aaai.v30i1.10295]

[48] Zhou Q, Arulra J, Navathe S, Harris W, Xu D. Automated verification of query equivalence using satisfiability modulo theories. Proc. of the VLDB Endowment, 2019, 12(11): 1276–1288. [doi: 10.14778/3342263.3342267]

[49] Cheng HT, Koc L, Harmsen J, Shaked T, Chandra T, Aradhye H, Anderson G, Corrado G, Chai W, Ispir M, Anil R, Haque Z, Hong LC, Jain V, Liu XB, Shah H. Wide & deep learning for recommender systems. In: Proc. of the 1st Workshop on Deep Learning for Recommender Systems. Boston: ACM, 2016. 7–10. [doi: 10.1145/2988450.2988454]

[50] LeFevre J, Sankaranarayanan J, Hacigumus H, Tatemura J, Polyzotis N, Carey MJ. Opportunistic physical design for big data analytics. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. Snowbird: ACM, 2014. 851–862. [doi: 10.1145/2588555.2610512]

[51] Perez LL, Jermaine CM. History-aware query optimization with materialized intermediate views. In: Proc. of the 30th IEEE Int'l Conf. on

Data Engineering. Chicago: IEEE, 2014. 520–531. [doi: 10.1109/ICDE.2014.6816678]

[52] Deshpande PM, Ramasamy K, Shukla A, Naughton JF. Caching multidimensional queries using chunks. ACM SIGMOD Record, 1998, 27(2): 259–270. [doi: 10.1145/276305.276328]

附中文参考文献

[1] 孙路明, 张少敏, 姬涛, 李翠平, 陈红. 人工智能赋能的数据管理技术研究. 软件学报, 2020, 31(3): 600–619. http://www.jos.org.cn/1000-9825/5909.htm [doi: 10.13328/j.cnki.jos.005909]

[2] 危剑豪, 夏烨峰, 宫学庆. 多查询共享技术研究综述. 软件学报, 2021, 32(10): 3176–3202. http://www.jos.org.cn/1000-9825/6203.htm [doi: 10.13328/j.cnki.jos.006203]

作者简介

刘全, 博士生, 主要研究领域为数据库系统, 机器学习.

李翠平, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为社交网络分析, 社会推荐, 大数据分析及挖掘.

陈红, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库技术, 新硬件平台下的高性能计算.