软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(12):5512−5536 [doi: 10.13328/j.cnki.jos.007405] [CSTR: 32375.14.jos.007405]

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

# 多目标深度强化学习驱动的数据库系统参数优化技术*

荣垂田[1], 田浩辉[1], 杜 方[2]

[1](天津工业大学 计算机科学与技术学院, 天津 300387)

[2](宁夏大学 信息工程学院, 宁夏 银川 750021)

通信作者: 荣垂田, E-mail: chuitian@tiangong.edu.cn

**摘 要**: 数据库系统的参数配置直接影响其性能和系统资源的利用率. 主流的关系数据库管理系统有数百个参数可供调整以获得最佳的性能和服务能力. 数据库系统性能的优化通常由经验丰富的数据库管理员 (DBA) 手动进行, 但是由于数据库系统配置参数众多、异构且参数之间存在复杂的相关性, 传统的人工进行参数调优的工作方法效率低、成本高、可复用性差. 为了提高数据库系统性能优化的工作效率, 数据库系统的自动化参数调优技术成为数据库领域的研究热点. 由于强化学习具有与系统运行环境交互、反馈并逐步优化的能力, 被广泛应用于复杂系统的优化过程. 相关的研究工作将强化学习及其改进方法应用于数据库系统的参数优化, 但是都采用单目标优化的方法.实际上, 数据库系统的参数优化属于多目标优化任务, 且调优工作常在资源受限的情况下进行, 因此现有的工作存在一些缺陷: (1) 将数据库系统优化任务的多个目标通过简单线性转换为单目标优化问题具有一定的盲目性, 需要反复迭代尝试优化, 实现成本高; (2) 无法应对数据库系统需求的动态变化, 适用性差; (3) 相关工作使用的强化学习方法本身是属于单目标优化算法, 将其应用于多目标任务时, 导致难以有效对齐偏好 (当前的各个目标的权重系数) 和相应的最优策略, 可能产生次优解; (4) 现有数据库系统参数优化的目标通常仅考虑吞吐量和延迟, 未考虑内存等资源的利用率. 针对以上问题, 设计一种基于多目标深度确定性策略梯度的强化学习算法 (MODDPG). 该方法是原生多目标的强化学习方法, 不需要将数据库系统优化的多目标任务转换为单目标任务, 可以高效适应数据库系统需求的动态变化. 通过改进强化学习算法的奖励机制可以快速实现偏好与最优策略的对齐, 有效避免次优解的产生, 提高数据库系统参数优化的效率. 为了更进一步验证所提方法的普遍适用性, 将提出的多目标优化的方法进行扩展, 实现了提升数据库的性能和资源利用率的多目标协同优化. 实验部分在主流关系数据库系统上使用 TPC-C 和 SYSBench 测试基准对所提参数优化方法的有效性和实用性进行了验证. 实验结果表明, 所提方法在模型的训练效率和数据库参数优化的作用方面具有明显优势, 并且易于根据优化需求扩展到更多目标.

**关键词**: 参数优化; 数据库系统; 深度强化学习; 多目标优化; 性能优化

**中图法分类号**: TP311

中文引用格式: 荣垂田, 田浩辉, 杜方. 多目标深度强化学习驱动的数据库系统参数优化技术. 软件学报, 2025, 36(12): 5512–5536. http://www.jos.org.cn/1000-9825/7405.htm

英文引用格式: Rong CT, Tian HH, Du F. Technique for Database System Parameter Optimization Using Multi-objective Deep Reinforcement Learning. Ruan Jian Xue Bao/Journal of Software, 2025, 36(12): 5512–5536 (in Chinese). http://www.jos.org.cn/1000-9825/7405.htm

## Technique for Database System Parameter Optimization Using Multi-objective Deep Reinforcement Learning

RONG Chui-Tian[1], TIAN Hao-Hui[1], DU Fang[2]

* 基金项目: 国家自然科学基金 (62062058)
收稿时间: 2024-06-07; 修改时间: 2024-09-22, 2025-01-02; 采用时间: 2025-02-11; jos 在线出版时间: 2025-07-17
CNKI 网络首发时间: 2025-07-18

[1](School of Computer Science and Technology, Tiangong University, Tianjin 300387, China)
[2](School of Information Engineering, Ningxia University, Yinchuan 750021, China)

**Abstract**: The tuning of database system parameters directly impacts its performance and the utilization of system resources. Relational database management systems typically offer hundreds of parameters that can be adjusted to achieve optimal performance and service capabilities. Database system performance optimization is traditionally carried out manually by experienced database administrators (DBAs). However, due to the characteristics of parameter tuning, such as the large number of parameters, their heterogeneity, and the complex correlations among them, traditional manual methods are inefficient, costly, and lack reusability. To enhance the efficiency of database system performance optimization, automated parameter tuning techniques have become a key focus in the database field. Reinforcement learning, with its ability to interact with the system environment and gradually improve through feedback, has been widely applied in the optimization of complex systems. Some related studies have applied reinforcement learning or its variants to database parameter tuning, but they have relied on single-objective optimization methods. Database system parameter tuning is a multi-objective optimization task, usually performed under resource constraints. Therefore, existing methods have several limitations: (1) transforming the multi-objective optimization problem into a single-objective optimization problem through simple linear transformations requires iterative attempts, making optimizations costly; (2) existing methods cannot adapt to the dynamic changes in database system requirements, limiting their adaptability; (3) reinforcement learning methods used in existing studies are designed for single-objective optimization, and their applications to multi-objective tasks make it difficult to effectively align preferences (the weight coefficients of current objectives) with corresponding optimal strategies, potentially leading to suboptimal solutions; (4) existing research primarily focuses on optimizing throughput and latency, while ignoring resource utilization such as memory. To address these issues, this study proposes a multi-objective deep deterministic policy gradient-based reinforcement learning algorithm (MODDPG). This method is a native multi-objective reinforcement learning approach that does not require transforming the multi-objective task of database system parameters tuning into a single-objective task, enabling it to efficiently adapt to dynamic changes in database system requirements. By improving the reward mechanism of the reinforcement learning algorithm, the alignment between preferences and optimal strategies can be quickly achieved, effectively avoiding suboptimal solutions. Consequently, the training process of the reinforcement learning model can be accelerated, and the efficiency of database system parameter tuning can be improved. To further validate the generality of the proposed method, the multi-objective optimization approach is extended to achieve a collaborative optimization goal of improving both database performance and resource utilization. Experiments using TPC-C and SYSBench benchmarks demonstrate the effectiveness and practicality of the proposed parameter tuning method. The results show significant advantages in terms of model training efficiency and the effectiveness of database parameter tuning.

**Key words**: parameter optimization; database system; deep reinforcement learning; multi-objective optimization; performance optimization

数据库系统是数据管理的核心, 能够对大规模的数据进行高效的组织、存储, 提供高效的数据访问服务, 确保数据的完整性和一致性, 已经成为企业信息和数据资产管理的核心, 也是业务运营、决策制定和创新发展的关键基础. 数据库不仅是存储和管理数据的核心技术, 更是支撑复杂业务和系统高效、稳定运行的基础软件. 因此, 数据库系统性能的优化不仅有利于提高数据管理的效率, 还有助于数据价值的挖掘和释放, 进一步提升业务处理效率, 助力企业实现提质、降本、增效.

数据库系统的参数优化是提高性能和提供可靠性服务的关键, 主流的关系数据库系统通常包含数百个参数可供调整和优化[1]. 例如, MySQL 数据库中的 innodb_buffer_pool_size 用于调整内存缓存池大小; table_open_cache 用于调整会话中可打开表的数量[2]; innodb_buffer_pool_instances 用于指定将 InnoDB 缓冲池划分为独立实例的数量; skip_name_resolve 设置为 ON 时, MySQL 不进行 DNS 解析, 可加快网络连接的速度. 根据实际的业务需求、工作负载和硬件环境正确地对数据库系统的调优参数进行设置, 不仅可以直接改善数据库系统的响应速度和吞吐量、提高系统资源的利用率, 还有利于提升信息系统整体的性能和可靠性.

数据库系统参数优化的难点在于如何选择合适的参数并设置合理的取值使系统的性能与负载的需求相匹配. 数据库系统产品通常会提供一些默认参数, 这些参数是数据库厂商根据历史运行经验提供的具有广泛的应用场景和硬件适配能力的参数. 但是, 对于特定的应用需求或工作负载数据库系统使用默认的参数难以提供高性能、高可靠的服务. 为了充分发挥数据库系统的性能优势、提高资源的利用率, DBA 通常需要根据业务需求、负载特点、硬件环境等对数据库系统的参数进行优化. 数据库系统的参数调优是数据库领域公认的复杂工程问题, 数据库中

一般都有上百个可供调整的参数, 参数之间、参数与数据库性能之间存在着复杂的关系. 数据库系统参数调优的挑战主要来自以下几个方面.

(1) 参数的变化对数据库系统性能的影响关系复杂. 如图 1 所示. 其中, 99%th-tile 表示 99% 的请求完成时间小于等于图 1 中点对应的值. 随着缓冲池大小 (buffer pool size) 的增加数据库系统的延迟先提升后减少, 随后因为存储资源有限, 资源耗尽而延迟增加. 当同时调整缓冲池大小和日志文件大小 (log file size) 的情况下, 数据库系统的延迟变化趋势更加复杂, 如图 2 所示.

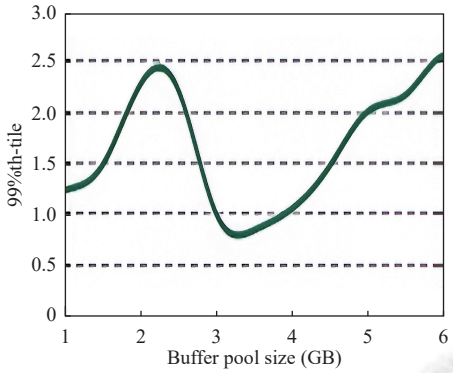



图 1 单个参数的变化对延迟的影响

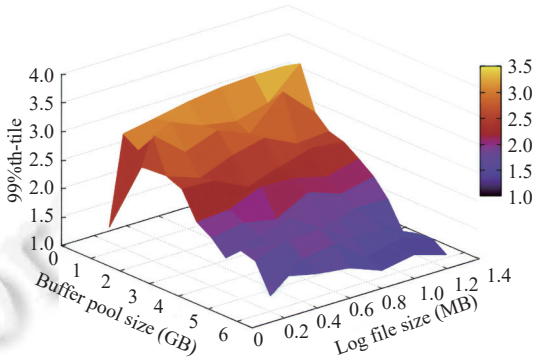



图 2 多个参数的变化对延迟的影响

(2) 参数空间高维且异构. 数据库系统有数百个配置参数, 有的是连续的 (innodb_buffer_pool_size 和 tmp_table_size), 有的是离散的 (innodb_stats_method 和 innodb_flush_neighbors). 这两种不同类型的参数在可微性和连续性上有根本的不同[3], 造成参数调整的难度和调整的空间不同.

(3) 参数的最优设置与负载特征、运行环境高度相关, 具有不可重用性. 面向一种负载特征进行的参数优化结果很难适用于另一种负载情况.

(4) 参数数量不断增长. 随着数据库系统的不断发展, 用于调整数据库系统性能的参数数量不断增长.

传统的数据库系统参数调优的工作由 DBA 根据工作经验通过人工试错法完成, 调优的过程包括: 根据经验调整数据库系统参数, 然后对当前选择的配置进行性能测试, 根据性能反馈来进一步调整参数. 这种工作方式的时间成本和人力成本较高, 很难在有限时间内达到较好效果[4].

为了提高数据库系统参数调优的工作效率, 工业界和学术界开始尝试采用基于搜索[5,6]和学习[2,7,8]的方法开展自动化参数调优的研究. 基于搜索的方法 BestConfig[5]可以在给定的资源限制内推荐最佳配置, 但无法利用过往经验. 基于学习的方法, 如 OtterTune[7], 它利用机器学习的高斯过程回归 (GPR) 方法来自动学习和推荐参数配置, 但此类方法需要大量的高质量训练样本. 由于高斯过程回归 (GPR) 方法是在高维连续空间中获取最优参数的过程, 基于此方法的数据库系统参数优化难以获得最优解. CDBTune+[2]采用基于 DDPG (deep deterministic policy gradient) 的深度强化学习方法实现数据库系统参数的自动化优化, 解决了传统机器学习方法依靠大量高质量训练样本、在高维连续空间中难以获取最优参数的难题. QTune[9]基于数据库系统当前的状态信息和 SQL 查询语句的信息使用两状态的深度确定性策略梯度模型 (double states DDPG, DS-DDPG), 能够在更细粒度的层面上对数据库系统配置参数进行性能优化. Gur 等人[10]提出了一种基于 DDPG 的多模型数据库系统参数优化方法, 解决生产环境中可能出现的负载和软件、硬件环境发生变化的情况.

以上基于 DDPG 模型的优化方法虽然都取得了一定的效果, 但是都把本质上是多目标优化任务的数据库参数调优, 通过线性组合将多个目标转换成了单个目标, 来使用传统的单目标强化学习模型, 降低了强化学习模型使用的难度, 但也存在以下几个问题需要改进.

(1) 提前确定偏好是一项繁琐且具有盲目性的任务. 在训练 DDPG 模型之前需要根据预先设置的偏好定制标

量奖励函数[11], 例如在 CDBTune+中, 通过线性加权 (设定吞吐量奖励加权系数 0.6, 延迟奖励加权系数 0.4) 来同时表示吞吐量和延迟的奖励. 然而, 对具体的数据库系统应用来说其吞吐量和延迟的偏好通常是难以提前预知, 并且随着业务的发展需求也是动态变化的.

(2) 设定固定偏好难以应对不同特征的任务需求. 预先确定某种偏好只能针对特定偏好寻找最优解, 当实际应用任务对吞吐量和延迟的要求变化时, 就需要重新修改偏好设置并重新训练模型, 难以灵活适应和应对不同需求的任务. 尽管可以找到可接受的解决方案, 但与系统的最佳效用可能相去甚远.

(3) 基于 DDPG 的强化学习方法其内部使用的奖励机制是基于标量形式的并通过贝尔曼方程 (Bellman equation) 对其迭代更新[12], 存在难以有效对齐偏好和相应的最优策略的问题[13], 需要大量训练样本去实现对齐, 并且可能导致次优解的产生. 然而, 在数据库系统参数优化的应用中缺乏现成的训练样本, 需要通过模型与数据库系统的交互过程中获取训练样本. 每次交互需要重启数据库系统来更新数据库调优参数并使之生效. 因此, 模型训练的时间成本很高 (MySQL 在模型训练的过程中重启时间从几分钟到十几分钟不等, PostgreSQL 重启需要 2 min 以上).

(4) 已有的相关工作通常只考虑数据库系统性能的提升, 没有将资源利用率作为优化目标.

针对以上问题, 本文提出了一个原生多目标的深度强化学习算法, 可以对数据库系统的参数进行优化以提升其性能. 本文主要贡献如下.

(1) 将多目标强化学习方法用于数据库系统参数优化任务, 适配数据库系统参数优化这一多目标优化场景.

(2) 针对数据库系统参数优化这一典型的多目标优化任务, 为了避免预先设置偏好带来的繁琐迭代过程和不利因素, 改进了传统强化学习方法的奖励机制, 提出了原生多目标的深度强化学习算法 N-MODDPG (native multi-objective DDPG).

(3) 针对多目标优化过程中强化学习的价值网络更新存在的难以对齐偏好和相应的最优策略的问题, 对 N-MODDPG 进行了优化和改进, 实现了 ON-MODDPG (optimized native multi-objective DDPG), 提出基于广义的 Bellman 方程的方法来学习所有可能偏好空间上最优策略的单参数表示, 利用解边界的凸包络更新价值网络的参数, 可以实现偏好和相应的最优策略的快速对齐, 避免次优解的产生. 该方法可以基于少量训练样本使模型训练达到较好的性能, 有效加速训练过程.

(4) 针对内存等系统资源受限的场景, 对多目标优化方法进行扩展, 实现了数据库性能与内存资源利用率协同优化算法 Co-MODDPG (collaborative multi-objective DDPG).

(5) 本文基于 TPC-C 和 SYSBench 测试基准在主流的关系数据库系统上, 对本文提出的方法的性能和有效性进行了验证, 并与当前最新的相关工作进行了对比分析.

## 1 数据库系统参数优化的相关工作

近年来, 随着人工智能技术的发展对数据库管理系统的配置参数 (Knobs) 进行自动化调优的工作成为数据库领域研究的热点. 现有的关于 DBMS 参数优化方面的工作有以下 3 种方法: 基于搜索的方法、基于贝叶斯优化 (Bayesian optimization, BO) 的方法和基于强化学习 (reinforcement learning, RL) 的方法, 详情见后文表 1. 在本节中将对这 4 种方法分别进行讨论.

### 1.1 基于搜索的方法

基于搜索的方法着重于在参数空间中使用一些启发式方法寻找最佳性能配置, 如 BestConfig 这是一个在给定应用程序工作负载下的已部署系统的资源限制内自动查找最佳配置设置的系统, 通过可扩展的采样方法 DDS (divide and diverge sampling) 和性能优化算法 RBS (recursive bound and search), 能够在给定应用程序工作负载下的已部署系统的资源限制内自动查找最佳配置[5].

然而基于搜索的自动调参方法有两个限制. 首先, 搜索最优配置本身需要花费大量时间; 其次, 每当新的调优请求到达时, 都需要重新启动搜索过程, 因此无法利用从以前的调优工作中获得的知识.

表 1 现有数据库配置参数调优方法的分类和简要描述

| 分类 | 配置优化系统 | 特点 | 应用 |
|---|---|---|---|
| 基于搜索 | BestConfig[5] | 通过启发式方法在配置空间中搜索 | DBMS的性能优化 |
| 基于BO | iTuned[6] | 首次采用BO | DBMS的性能优化 |
| | OtterTune[7] | 增量式旋钮、工作负载映射、GPR | DBMS的性能优化 |
| | ResTune[14] | 采用RGPE传递历史知识 | 面向资源的DBMS调优 |
| | LlamaTune[15] | 低维投影、偏抽样 | DBMS的性能优化 |
| 基于RL | CDBTune+[2] | 首次采用DDPG | DBMS的性能优化 |
| | QTune[9] | 支持3种调优粒度 | DBMS的性能优化 |
| | DB-BERT[16] | 使用NLP | DBMS的性能优化 |
| | WATuning[17] | 引入注意力机制 | DBMS的性能优化 |
| | HUNTER[18] | 遗传算法生成样本预热 | DBMS的性能优化 |
| | UDO[19] | 基于RL | DBMS的性能优化 |

### 1.2 基于 BO 的方法

基于 BO 的方法, 将调优建模为一个黑盒优化问题, 对配置和数据库性能之间的关系进行建模. 他们遵循 BO 框架来搜索最优的 DBMS 配置: (1) 拟合概率代理模型; (2) 通过最大化目标函数来选择下一个配置进行评估.

iTuned 首先采用 BO 模型来寻找性能良好的配置. 它使用随机抽样技术——Latin hypercube sampling 进行初始化, 并且不使用从以前的调优会话中收集的观察结果来加速目标调优任务, 通过计划的实验主动引入适当的数据, 探索解空间找到高影响参数和高性能参数设置, 通过周期偷窃范式降低在生产工作负载上的开销.

OtterTune[7]通过工作负载映射来选择重用历史数据, 通过 FA 因子分析和 K-means 聚类剪裁冗余的数据库内部指标 (Metrics), 用 Lasso 暴露与系统整体性能相关性最强的旋钮, 在回归中包含多项式特征来检测旋钮之间的非线性相关性和依赖性, 训练高斯过程回归 (GPR) 来为当前的工作负载推荐配置参数.

ResTune 定义了一个面向资源的调优问题, 并采用了约束贝叶斯优化求解器. 它使用一个集成框架 (即 RGPE) 来跨调优任务组合工作负载编码器, 以传输历史知识.

LlamaTune 是一种利用领域知识来提高现有优化器样本效率的调谐器设计. LlamaTune 采用了一种基于随机投影的自动降维技术, 一种偏采样方法来处理某些旋钮的特殊值, 以及旋钮值桶化, 以减少搜索空间的大小, LlamaTune 集成了 3 个优化器: 两个基于 BO 的优化器 (SMAC, GP-BO) 和一个基于 RL 的优化器 (DDPG).

然而, 基于 BO 的方法存在的不足如下: 首先, 有些基于 BO 的方法 (如 OtterTune) 采用流水线式学习模型, 前一阶段的最优解不能保证后一阶段的最优解, 模型的不同阶段之间可能不能很好地协同工作. 其次, 依赖于难以获得的大规模高质量训练样本. 最后, 实际操作中存在大量参数, 大多数基于 BO 的方法 (如 OtterTune 使用的 GPR) 是难以在高维连续空间中优化参数设置的, 并且参数位于连续空间中, 具有潜在的依赖关系, 参数的组合数量规模大, 仅靠人为地在回归中包含多项式特征难以找到所有的、潜在的依赖关系.

目前在高维配置空间中表现最好的基于 BO 的方法是 SMAC, 而基于 SMAC 的模型训练时只能选择单一的优化目标 (吞吐量或延迟等), 无法实现多目标的协同优化任务.

### 1.3 基于 RL 的方法

基于强化学习的方法基本采用深度确定性策略梯度 (DDPG) 模型, 这是一种基于策略的无模型强化学习方法. 它将参数调优视为在探索未知空间和利用现有知识之间进行权衡的试错过程.

Zhang 等人[2]在 2019 年提出了一种端到端的数据库自动调参工具 CDBTune+, 该系统使用深度确定性策略梯度 (DDPG) 算法. 该深度强化学习方法是基于 Q-learning 算法演化而来, 基于贝尔曼最优方程进行迭代更新, 借助神经网络不仅可以在高维连续的参数空间中优化最优配置并且可以自动考虑参数间潜在的依赖关系, 奖励函数的设计同时考虑了当前参数配置的性能与初始的性能变化以及当前参数配置的性能同前一次的性能变化. 为了降低

模型对大量样本的依赖性并打乱马尔可夫决策过程 (MDP) 带来的相邻数据的相关性, CDBTune+使用了优先经验回放 (PER) 方法[20], 从经验池中抽取将 TD-error 作为优先级存放的过往经验样本. CDBTune+的实验结果表明相比于当时已有的其他数据库自动调参工具, 它针对不同负载环境产生的数据库参数配置使数据库系统的吞吐量更高、时延更低.

CDBTune+在与数据库系统环境交互以优化数据库参数时, 只将数据库当前的运行指标 (Metrics) 作为强化学习从环境中获取的状态信息, 而忽略了 SQL 语句本身包含的信息. 该方法只支持粗粒度调优 (例如, 工作负载级调优), 而不能提供细粒度调优 (例如, 查询级调优). Li 等人[9]在 2019 年提出了一个基于深度强化学习 (DRL) 模型的查询感知数据库调优系统 QTune, 除了数据库状态之外, 该系统将查询特征也输入到 Actor 网络中, 以选择合适的配置. 即使用双状态深度确定性策略梯度 (DS-DDPG) 模型来实现查询感知的数据库配置调优, 该模型利用 Actor-Critic 网络来基于查询向量和数据库状态调优数据库配置. 虽然 QTune 模型能够针对具体的 SQL 提供更细粒度的性能优化, 但是由于 SQL 语句有时会包含用户个人的隐私信息, 所以该工具存在隐私泄漏的安全隐患.

由于云上环境中系统资源、工作负载和数据库大小的不断的动态变化, 因此, 用于自动调优的系统需要灵活并适应这些变化, 以便为给定的环境状态提供最佳性能. 2021 年 Gur 等人[10]提出了一种对工作负载变化敏感的多模型在线调优算法, 该算法将多个 DDPG 强化学习模型在不同负载下进行训练, 为不断变化的工作负载选择最优模型.

除此之外, HUNTER[18]和 UDO[19] 也是基于强化学习的系统, 它们使用强化学习技术在系统的内部指标和参数之间建立神经网络来调整系统的性能. DB-BERT[16,21]结合 NLP 和强化学习来调整数据库参数, 通过分析文本以确定了数据库调参的热点, 并应用奖惩措施来优化数据库性能. WATuning[17]在深度强化学习的基础上引入了注意力机制, 将不同的工作负载也作为模型的输入以更好地适应实际调优环境中工作负载的变化.

上述的基于单目标优化模型 (DDPG) 的数据库系统参数自动优化方法, 在面对数据库性能优化这种多目标优化任务时都存在一定缺陷. 在设置奖励机制时, 需要提前确定调优偏好, 这个过程依赖于专家经验而且具有一定的盲目性; 单目标强化学习的值函数更新方式难以应对多目标优化场景; 从发展的角度来看, 用户对数据库系统的需求是变化的, 调优的偏好也要根据需求进行重新设计, 模型需要重新训练, 造成此类方法的适用性差; 并且没有考虑到在系统资源受限的场景下进行参数优化的问题, 即难以根据需求扩展到更多优化目标.

## 2 强化学习与数据库系统自动参数优化的相关知识

### 2.1 强化学习

强化学习是机器学习的一个分支, 是一种模仿人类或动物在面对未知环境时的学习方式, 是一种从试错过程中发现最优行为策略的技术, 已经成为解决环境交互问题的通用方法[22]. 强化学习主要由智能体 (Agent)、环境 (Environment)、状态 (State)、动作 (Action)、奖励 (Reward)、策略 (Policy) 组成. 智能体执行某个动作后, 环境将会转换到一个新的状态. 对于该新的状态, 环境会给出奖励信号 (正奖励或者负奖励). 智能体会根据新的状态和环境反馈的奖励, 按照一定的策略执行新的动作, 如图 3 所示. 强化学习的过程是让智能体与环境的交互中, 通过尝试不同的行为并观察环境的反馈以最大化累积奖励 (回报) 的方式使智能体找到行为的最佳策略.

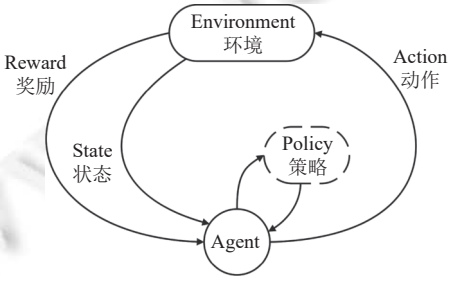



图 3 强化学习框架

如图 3 所示, 强化学习涉及以下几个要素.

(1) 环境 (Environment): 智能体进行学习和决策的场所. 环境可以是现实世界中的任何场景, 也可以是模拟环境.

(2) 状态 (State): 描述了在特定时刻智能体所处的情况或条件. 状态对于智能体的决策非常重要, 因为它决定了智能体应该采取何种行动.

(3) 动作 (Action): 智能体在特定状态下可以执行的操作或行为. 智能体通过选择不同的动作来影响环境.

(4) 奖励 (Reward): 环境根据智能体执行的动作和状态的变化给予的反馈. 奖励可以是正数、负数或 0, 用来评估智能体的行为好坏.

(5) 策略 (Policy): 智能体采取行动的规则或策略. 策略可以是确定性的 (给定状态, 选择一个特定的动作) 或是随机的 (根据概率选择动作).

将强化学习应用于数据库系统参数的优化时, 数据库系统作为环境 (Environment), 优化算法作为智能体 (Agent), 数据库系统的运行时度量指标 (Metrics) 作为状态 (State). 数据库系统的指标 (Metrics) 是记录其内部运行时组件活动的计数器. 例如, 系统自启动以来从磁盘读取的页面数, 表的行锁持有的平均时间等. 根据数据库的当前性能指标 (如: 吞吐量、延迟) 的变化设置奖励函数 (Reward), 将优化算法推荐的参数设置作为智能体的动作 (Action). 工作流程如图 4 所示.

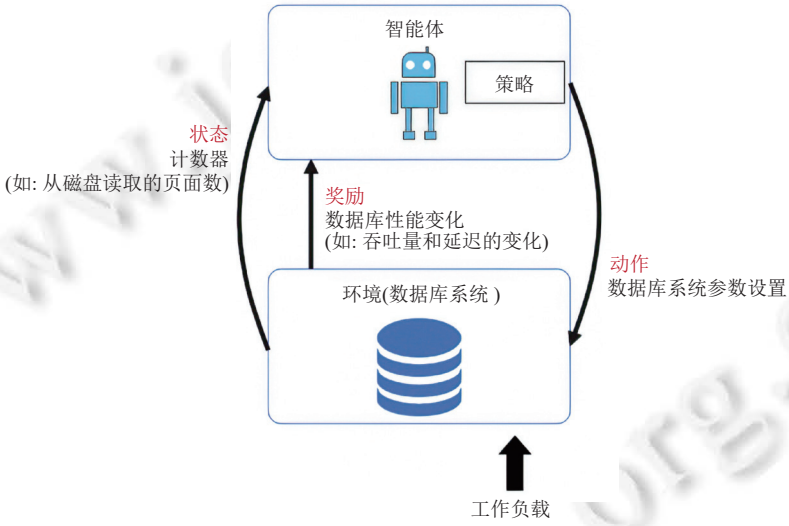



图 4 强化学习应用于数据库系统参数优化的工作框架

(1) 智能体 (优化算法) 从环境 (数据库系统) 中获取当前的环境状态, 根据策略做出动作 (参数设置), 动作会对环境产生影响, 数据库系统的性能会发生变化;

(2) 环境会根据预先设置的奖励函数对当前的动作产生的影响做出评价, 给出一个奖励反馈;

(3) 智能体会根据收到的奖励反馈和预定的策略, 做出下一个动作;

(4) 迭代上述过程, 直到累计奖励达到最大. 即智能体与环境的交互过程的累计奖励的变化趋于稳定.

强化学习的方法有很多种类别, 包括基于价值的、基于策略的、基于策略和价值相结合的方法, 每种方法都有各自的特点和应用场景. 其中, 基于价值和策略相结合的强化学习方法能够充分利用价值函数和策略函数的优势, 并通过相互协作来提高学习的效率和稳定性, 在解决复杂问题时通常具有稳定性、高效性、广泛适用性. 深度确定性策略梯度 (DDPG) 方法属于基于策略和价值相结合的方法, 适用于连续动作空间的强化学习问题. DDPG 学习的是确定性策略, 即给定状态直接输出动作, 而不是学习一个概率分布, 适用于数据库参数调优这种应用

场景.

### 2.2 基于 DDPG 的数据库参数优化模型的训练过程

现有的基于强化学习的数据库系统参数优化的工具基本都采用基于 DDPG 的算法. DDPG 算法通过构造一个确定性策略能够从连续的动作空间中获取一个 (一组) 确定的动作. 该算法的训练数据为<$s_t$, $a_t$, $r_t$, $s_{t+1}$>储存于经验回放池中, 其中 $s_t$ 表示当前数据库系统的状态, $a_t$ 表示当前智能体的动作 (即推荐的数据库参数), $r_t$ 表示采用当前动作获取的奖励, $s_{t+1}$ 表示下一次调优时数据库系统的状态. 如图 5 所示, 在模型训练过程中, $s_t$ 与 $s_{t+1}$ 均从数据库系统中获取的数据库系统的运行时度量参数; Actor 网络获取 $s_t$ 用于生成动作 $a_t$; 然后借助性能测试工具如: TPC-C、SYSBench 等测量当前数据库配置参数下的性能 (吞吐量、延迟), 并用事先设计好的奖励函数由吞吐量与延迟计算出奖励 $r_t$.

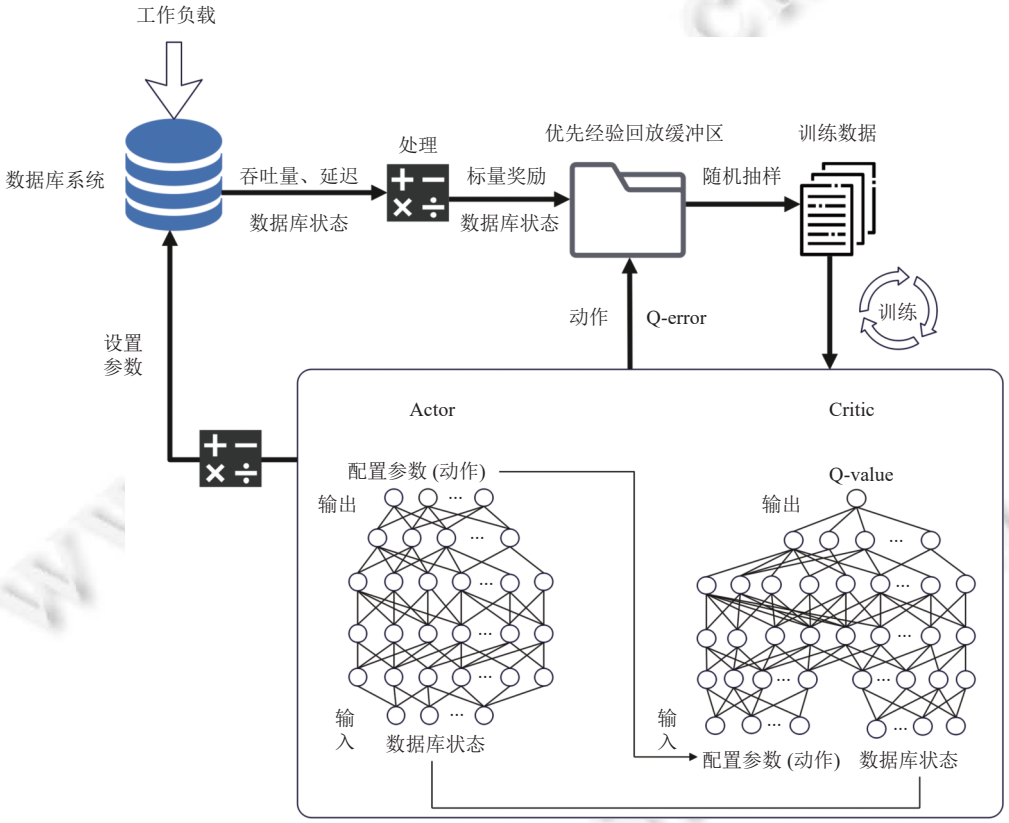



图 5 基于 DDPG 的数据库参数优化模型的训练过程

DDPG 模型训练过程的关键要素包括以下几个方面.

(1) Actor-Critic 结构: DDPG 采用了 Actor-Critic 结构. 其中, Actor 学习一个确定性策略, 即直接输出连续动作的值. Critic 则评估 Actor 输出的策略, 并提供相应的奖励反馈.

(2) 经验回放池 (experience replay): 经验回放, 即存储和重用经验样本的方法. 这些经验中包含了过去决策的信息, 有助于更好地训练模型, 提高样本利用效率; 通过随机抽样的方式获取经验, 可以减少数据之间的相关性, 避免连续样本之间的相关性导致模型训练不稳定.

(3) 目标网络 (target network): 为了提高算法的稳定性和收敛性, DDPG 使用了目标网络. 这包括目标策略网络和目标值函数网络. 通过延迟更新目标网络的参数, 可以减少训练中的相关性问题.

(4) 奖励函数: DDPG 的奖励函数通常用于衡量动作的优劣, 以指导策略的更新. 这些奖励可以根据具体问题

进行设计, 以鼓励或抑制特定行为 (动作).

### 2.3 参数优化模型的应用

模型经过离线训练之后, 将使用图 6 中描述的过程来为数据库系统推荐优化的配置参数. 首先, 需要将用户的数据库负载回放以获取数据库的状态参数, Actor 网络基于获取的数据库状态参数为数据库系统推荐相应的配置参数. 由于 DDPG 算法是确定性策略梯度方法, 存在动作空间 (参数空间) 探索性不足的缺陷. 为了获取最优的推荐参数, 增加 DDPG 算法对未知参数设置的探索能力, 可以选择在 Actor 网络向前传播的过程中加入噪声, 或者将产生的动作经过 OU (Ornstein-Uhlenbeck) 随机过程处理.

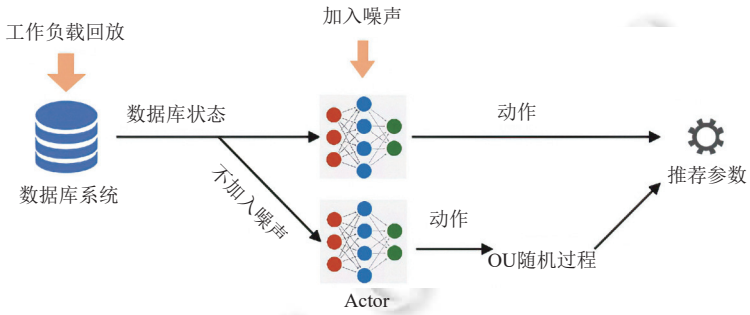



图 6 推荐参数生成的过程

## 3 基于多目标强化学习的数据库参数自动调优方法

尽管许多实际问题可以简化为单目标强化学习 (SORL) 任务, 但这种方式只能在偏好空间中学习“平均”策略, 而不能针对特定偏好定制最优策略. 实际上, 许多现实世界的任务需要考虑它们的多目标性质[23]. 多目标强化学习 (MORL) 是强化学习 (RL) 的一个子领域, 它试图为至少有两个目标的问题找到最优策略[24].

针对数据库系统参数优化这一典型的多目标优化任务, 为了解决相关工作中存在的将多目标优化任务简单线性变换为单目标优化任务带来的缺陷, 本文提出了原生多目标的强化学习算法 N-MODDPG, 该算法改进了传统强化学习方法中的奖励机制, 避免了模型的重复训练. 为了实现目标偏好和相应的最优策略的快速对齐, 基于广义的 Bellman 方程改进了 DDPG 算法中价值函数的更新方式, 使得价值函数的更新过程能够考虑整个偏好空间, 从而避免次优解的产生, 实现了优化的 ON-MODDPG (optimized native multi-objective DDPG) 算法.

### 3.1 基于多目标强化学习的数据库调优系统

本文提出的多目标的强化学习算法 MODDPG 的系统架构 MODBTune, 如图 7 所示. 该系统主要由 3 个模块组成, 分别是数据库系统模块、性能测试模块和系统调优模块.

数据库系统模块可以适配主流关系数据库系统 (MySQL, PostgreSQL 等), 该模块需要为系统调优模块提供数据库状态参数.

性能测试模块为模型训练生成工作负载并监测数据库性能, 以及在线参数调优时回放用户的工作负载. 对于数据库性能 (延迟和吞吐量) 的监测, 本文设置每 5 s 采样一次, 然后计算采样结果的平均值用于计算模型的奖励.

系统调优模块为数据库系统提供高性能的配置参数. 其中控制器用于协调数据库系统、性能测试和系统调优这 3 个模块的工作.

图 7 中 Metrics 表示 DBMS 运行过程中的一些运行时度量指标, 是记录其内部运行的组件活动的计数器. 数据库系统的运行时度量指标使 DBA 和监视工具能够观察系统的行为并诊断性能问题. 运行时度量指标有 3 种类型: 累积指标、聚合指标和状态指标.

累积指标用于计算自某个时间点以来发生的事件数量. 例如, DBMS 可以记录自启动以来从磁盘读取的页面数. 聚合指标用于记录一个时间窗口内的平均事件数. 例如, 获取表的行锁的平均时间. 状态指标用于记录活动或

选项的当前状态. 例如, 垃圾收集器的最后一次运行时间和 SSL 协议版本.

Metrics 收集器用于收集和处理 DBMS 的运行时度量指标数据, 这些数据捕获了特定时间间隔内数据库系统运行时行为的各个方面的特征. 对于每一个 Metric 根据其类别不同需要计算一定时间间隔内的均值或计算其累积值之间的差值, 最终以向量的形式提供给深度强化学习模型.

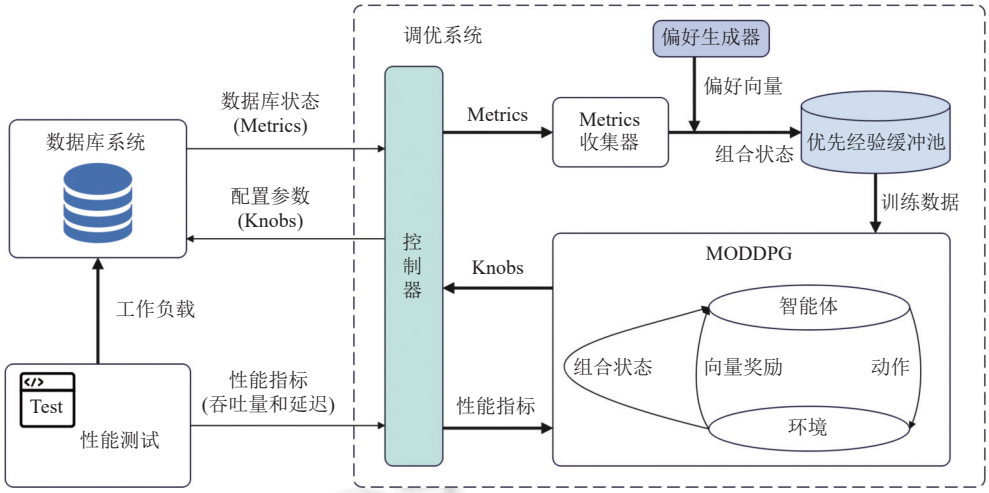



图 7 基于多目标深度强化学习的数据库参数调优系统 MODBTune

偏好生成器生成具有标准正态分布的随机权重向量用于代表对不同优化目标的偏好, 将偏好向量与数据库状态 (Metrics) 结合形成组合状态, 将组合状态作为模型多目标强化学习模型的状态输入.

### 3.2 基于 DDPG 的原生多目标强化学习方法实现

传统的强化学习方法采用标量奖励, 适用于单目标优化场景. 现实世界的许多控制问题通常需要平衡多个相互冲突的目标 (如: 吞吐量和延迟), 当一个目标的最大化是以牺牲另一个目标为代价时, 决策者必须在它们之间找到一个妥协方案[25]. 针对这种多目标优化的问题, 需要对传统强化学习进行改进, 实现奖励的向量化. 采用向量奖励的强化学习方法称为多目标强化学习 (multi-objective reinforce learning).

为了使强化学习方法能够实现多目标优化任务, 要对传统的标量奖励机制进行改进, 实现的具体细节如下. 为了更好地描述相关的理论, 对相关符号进行如下的解释.

$T$ 和 $L$ 分别表示数据库的性能指标吞吐量和延迟.

episode: 模型训练的轮次.

$T_t$ 与 $T_0$ 分别表示一条 episode 在 $t$ 时刻和初始时刻的吞吐量.

$L_t$ 和 $L_0$ 分别表示一条 episode 在 $t$ 时刻和初始时刻的延迟.

$\Delta T_{t\to 0}$: 从初始时刻 (调优前) 到 $t$ 时刻吞吐量的变化率.

$\Delta T_{t\to t-1}$: 从 $t$–1 时刻到 $t$ 时刻吞吐量的变化率.

$\Delta L_{t\to 0}$: 从初始时刻 (调优前) 到 $t$ 时刻延迟的变化率.

$\Delta L_{t\to t-1}$: 从 $t$–1 时刻到 $t$ 时刻延迟的变化率.

$\Delta_{t\to 0}$: 从初始时刻 (调优前) 到 $t$ 时刻性能 (吞吐量或延迟) 的变化率.

$\Delta_{t\to t-1}$: 从 $t$–1 时刻到 $t$ 时刻性能 (吞吐量或延迟) 的变化率.

$$\Delta T=\begin{cases}\Delta T_{t\to 0}=\dfrac{T_t-T_0}{T_0}\\ \Delta T_{t\to t-1}=\dfrac{T_t-T_{t-1}}{T_{t-1}}\end{cases} \tag{1}$$

$$\Delta L=\begin{cases}\Delta L_{t\to 0}=\dfrac{-L_t+L_0}{L_0}\\ \Delta L_{t\to t-1}=\dfrac{-L_t+L_{t-1}}{L_{t-1}}\end{cases} \tag{2}$$

公式 (1) 和公式 (2) 分别表示吞吐量和延迟的变化量, 根据公式 (3) 计算吞吐量和延迟的奖励. 奖励函数计算公式如下:

$$r=\begin{cases}((1+\Delta_{t\to 0})^2-1)|1+\Delta_{t\to t-1}|, & \Delta_{t\to 0}>0\\ -((1-\Delta_{t\to 0})^2-1)|1-\Delta_{t\to t-1}|, & \Delta_{t\to 0}\leqslant 0\end{cases} \tag{3}$$

在传统的单目标优化的强化学习方法中将奖励函数设置为标量形式, 如 CDBTune+[2]使用的 DDPG 方法将奖励 $r$ 表示为单次调优后吞吐量和延迟的奖励的线性组合, 如公式 (4) 所示:

$$r=\alpha\times r_T+\beta\times r_L \tag{4}$$

为了避免预先设置偏好带来的繁琐迭代过程和不利因素, 以及将多目标优化任务简单线性变换为单目标优化任务带来的缺陷, 本文将奖励形式向量化表示为:

$$r=<r_T,r_L> \tag{5}$$

基于以上符号描述和定义, 本文原生多目标强化学习算法 N-MODDPG 的实现代码如算法 1 所示. N-MODDPG 采用 4 个神经网络, 其中 Actor 和 Critic 分别模拟策略和价值函数, 且各自用一个目标网络来解决 $Q$ 值高估的问题. 算法中第 1–3 行初始化各个网络参数及优先经验回放池 $R$; 算法中第 4 行设置模型训练的轮次 episode 的个数; 第 5 行使用 OU 随机过程初始化随机噪声 $N$; 算法的第 6 行从数据库系统中获取运行时度量参数 Metrics, 并处理为每个 episode 的初始状态 $s_0$; 从第 7 行开始, 每个 episode 持续 $T$ 轮, 每轮将初始状态输入给 Actor 网络 $\pi(s_t|\theta^\pi)$, 将其输出加上随机噪声 $N$ (增加对环境的探索性) 作为动作 $a_t$ (第 8 行), 执行当前动作 $a_t$ 获取向量化的奖励 $<r_T,r_L>$ 状态 $s_{t+1}$, 第 8 行操作具体含义为: 把动作处理为适当的数据库系统参数写入相应配置文件中, 重启数据库系统使配置参数生效, 执行基准测试获得数据库系统当前性能, 由提前设置的奖励函数处理为向量奖励, 数据库系统重启后状态发生变化: $s_t\to s_{t+1}$ ($t\in[0,T]$), 将获取到的数据 $<s_t,a_t,r_t,s_{t+1}>$ 放入优先经验回放池 $R$ 中; 算法中第 10–15 行用于更新 Critic 网络和 Actor 网络的参数, 包括通过数据采样获取更新操作所用的样本、用目标网络[26]计算 TD Target 来指导 Critic 网络的更新、计算 TD Target 和当前 $Q$ 值的差值 TD-error 来构造损失函数用于更新 Critic 网络、使用随机策略梯度更新 Actor 网络, 最后更新目标网络.

**算法 1**. N-MODDPG.

1. $\theta^Q$ 和 $\theta^\pi$ 随机初始化 Critic 网络 $Q(s,a|\theta^Q)$ 和 Actor 网络 $\pi(s|\theta^\pi)$
2. $\theta^{Q'}\leftarrow\theta^Q$, $\theta^{\pi'}\leftarrow\theta^\pi$ 初始化目标网络权重参数
3. 初始化优先经验回放池 $R$
4. **for** episode = 0 → $E$ **do**
5. 随机噪声 $\mathcal{N}$ 初始化
6. 获得初始状态 $s_0$
7. **for** $t=0\to T$ **do**
8. $a_t=\pi(s_t|\theta^\pi)+\mathcal{N}$
9. 执行动作 $a_t$, 得到向量奖励 $r_t=<r_T,r_L>$ 和状态 $s_{t+1}$, 将 $<s_t,a_t,r_t,s_{t+1}>$ 存入 $R$
10. 从 $R$ 中采样批量为 $n$ 的多维数组 $<s_i,a_i,r_i,s_{i+1}>$
11. **if** *update* **then**
12. $y_i=r_i+\gamma Q'(s_{i+1},\pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
13. //最小化损失函数 $L$ 来更新 Critic 网络

14. $Loss = \frac{1}{n}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
15. //采样策略梯度更新 Actor 策略网络
16. $\nabla_{\theta^\pi} J(\theta^\pi) \approx \frac{1}{n}\sum_i \nabla_{\theta^\pi} \pi(s_i|\theta^\pi)\nabla_a Q(s_i, a|\theta^Q)|a = \pi(s_i)$
17. //更新目标网络
18. $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
19. $\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$
20. **end**
21. **end**
22. **end**

## 4 对原生多目标强化学习方法的优化

N-MODDPG 通过改进强化学习的奖励机制, 以适用于多目标优化场景, 避免了传统强化学习在多目标优化过程中需要反复确定不同目标的权重这一弊端. 但是, 由于强化学习模型中的价值网络的更新方法是基于传统的 Bellman 方程, 应用到多目标场景下, 无法有效表达多个优化目标的相对重要性 (偏好), 难以将偏好与最优策略对齐, 难以动态适应或转移到不同偏好的相关任务. 因此, N-MODDPG 需要进一步改进.

马尔可夫决策过程 (Markov decision process, MDP) 是一种数学框架, 用于描述具有随机性和不确定性的决策问题. 在 MDP 中, 系统被建模为一组状态, 每个状态之间通过某些动作进行转移, 转移可能伴随着奖励或成本. MDP 具有马尔可夫性质, 即状态转移的概率只依赖于当前状态和执行的动作, 而与之前的状态序列无关.

而强化学习是用于解决决策问题的机器学习方法, 传统的强化学习中的 Bellman 方程是针对马尔可夫决策过程 (MDP) 提出的公式化解决方案.

马尔可夫决策过程 (MDP) 表示为: $<S, A, P, r, \gamma>$, 其中 $S$ 和 $A$ 分别为状态和动作空间, 状态转移矩阵 $P(s', r|s, a)$, 标量奖励函数 $r(s,a)$, 折扣奖励因子 $\gamma \in [0,1]$.

基于马尔可夫决策过程理论的 Bellman 方程表示为:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') q_\pi(s', a') \tag{6}$$

状态动作价值函数 $q_\pi(s,a)$ 的贝尔曼最优方程 (Bellman optimality equation) 记为:

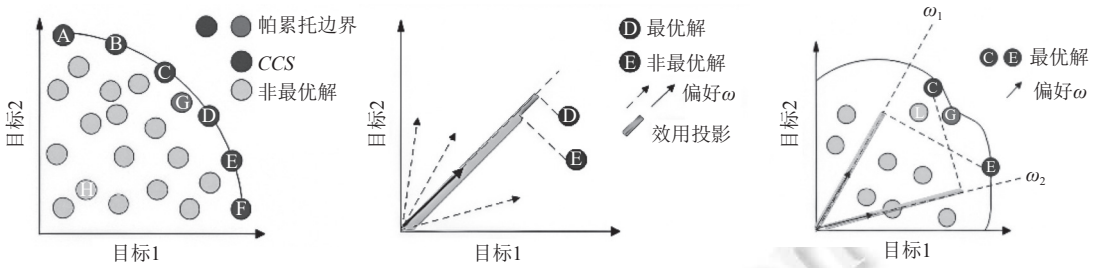
$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} q_*(s', a') \tag{7}$$

多个优化目标的决策问题与单一优化目标的决策问题有所不同, 需要重新设计一个多目标的马尔可夫决策过程[27]数学框架用以描述多个优化目标的决策问题, 进而给出多目标强化学习针对该多个优化目标的决策问题的公式化解决方案 (多目标的 Bellman 方程).

本文设计的多目标的马尔可夫决策过程 (MOMDP) 表示为: $<S, A, P, r, \gamma, W, f_W>$, 与 MDP 不同的是, 奖励函数 $r(s,a)$ 是向量表示形式, 同时引入了偏好空间 $W$ 体现多目标决策问题的多目标特性, 引入效用函数 $f_W$ 以体现在多目标偏好空间中偏好对探索最优解的影响[28]. 其中效用函数 $f_W$ 可定义为: $f_\omega(r(s,a)) = \omega^{\mathrm{T}} r(s,a)$, 即效用函数的值体现为在多目标场景中当前策略探索到的回报 (使用贝尔曼方程探索到的奖励的累积) 在偏好方向上的投影长度, 如图 8(b) 中所示. 如果式中的 $\omega$ 和 $r(s,a)$ 为标量, 则该 MOMDP 变为标准 MDP. 为了说明单目标强化学习针对单一优化目标的决策问题给出的解决方案 (贝尔曼方程) 在多目标优化场景中的缺陷, 进一步引出多目标贝尔曼方程的优势, 下面引入帕累托边界和凸覆盖集 (*CCS*)[13]的概念.

MOMDP 所有可能的回报可用于构成一个帕累托边界 (Pareto frontier) $F^* = \{\nexists \vec{r}' \geqslant \vec{r}\}$, 其中回报 $\vec{r} = \sum_t \gamma^t r(s_t, a_t)$. 将帕累托边界中凸覆盖集 (convex coverage set, *CCS*) 定义为:

$$CCS=\left\{\hat{r}\in F^{*}\mid\exists\omega\in W \text{ s.t. } \omega^{\mathrm{T}}\hat{r}\geqslant\omega^{\mathrm{T}}\hat{r}',\forall\hat{r}'\in F^{*}\right\} \tag{8}$$




(a) *CCS*和帕累托边界　(b) 偏好与最优解对齐　(c) 偏好与最优解未对齐

图 8　偏好空间中 *CCS* 上的最优解

图 8(a) 显示了 *CCS* 和帕累托边界的一个例子, *CCS* 是帕累托边界 (点 A 到 F 和 G) 的一个子集, 包括其外凸边界上的所有解 (不包括点 G). 当给定特定的线性偏好 $\omega$ 时, *CCS* 上沿相对重要性权重 (偏好) 方向的投影长度最大的点将是最优解如图 8(b) 所示.

如果在多目标优化场景中使用单目标强化学习方法, 即将多个优化目标简单线性组合 (提前设置各个目标的权重系数即提前固定偏好) 为单目标任务, 该情况如图 8(c) 中所示. 因为该固定偏好 (如 $\omega_1$ 或 $\omega_2$) 是人为设定的具有一定盲目性, 很有可能与当前策略所探索到的最大回报不对齐, 即该探索到的最大回报很有可能不是当前偏好上的最优解.

为了更好地说明单目标强化学习在多目标应用场景下探索最优解的过程中存在的问题, 进而引出多目标强化学习的解决方案, 给出了如图 8 所示的偏好空间中最优解的示意图.

图 8(a) 中帕累托边界在图中表示为点 A 到 F, 加上点 G, 而 *CCS* 是帕累托边界的子集 (点 A 到 F). 点 H 表示非最优解. 图 8(b) 中从 *CCS* 中选择效用最高的最优解, 用偏好向量的投影长度表示. 箭头表示不同的线性偏好, 点表示可能的回报. 在实线偏好下, 回报 D 的累积效用优于回报 E. 图 8(c) 中标量化的 SORL 算法在不与偏好对齐 (即在当前偏好下该解在偏好方向上投影长度不是最大的) 的阶段找到相对最优解, 例如 *CCS* 中的两个最优解 C 和 E, 与偏好 $\omega_2$ 和 $\omega_1$ 不对齐. 沿着 $\omega_1$ 方向搜索的更新过程, 不能利用 $\omega_2$ 方向上的最优解 C, 最终导致非最优 *L*, 反之亦然.

因此单目标强化学习针对单一优化目标的决策问题给出的解决方案 (贝尔曼方程) 在多目标优化场景中存在缺陷, 不能实现偏好与最优策略 (在该策略下可以探索到当前偏好的最优解) 的快速对齐, 即不能获得当前偏好下的最优解.

为了能在偏好空间中实现当前偏好与最优策略的快速对齐, 即获得当前偏好下的最优解, 需要修改贝尔曼方程使智能体在训练过程中能够近似 MOMDP 的整个 *CCS* 的策略, 在整个偏好空间中进行泛化, 使 Critic 网络在更新过程中能够在整个 *CCS* 中快速找到当前偏好下的最优解, 使该最优策略与当前偏好对齐, 进而在测试时能够适应任何给定 $\omega\in\Omega$ 的最优策略. 修改后的多目标贝尔曼方程如下.

多目标贝尔曼期望方程:

$$q_{\pi}(s,a,\omega)=r(s,a)+\gamma\sum_{s'}p(s'|s,a)\sum_{a'\in A,\omega\in W}\pi(a'|s',\omega)q_{\pi}(s',a',\omega) \tag{9}$$

多目标贝尔曼最优方程:

$$q_{*}(s,a,\omega)=r(s,a)+\gamma\sum_{s'}p(s'|s,a)\max_{a\in A,\omega\in W}q_{*}(s',a,\omega),a=\pi'(s_{i+1},\omega) \tag{10}$$

相比于标准的贝尔曼方程, 多目标的贝尔曼方程考虑了偏好 $\omega$ 的影响, 多目标贝尔曼最优方程能够在偏好维度上探索最优 *Q* 值用以更新 Critic 网络, 同时将偏好 $\omega$ 作为策略函数 $\pi(a|s,\omega)$ 和价值函数 $q_{\pi}(s,a,\omega)$ 的输入, 如图 9

所示 (图中动作维度和奖励维度分别为 16 和 2, 偏好向量来源于偏好生成器), 策略函数 $\pi(a|s,\omega)$ 和价值函数 $q_\pi(s,a,\omega)$ 分别对应于 Actor 和 Critic 网络.

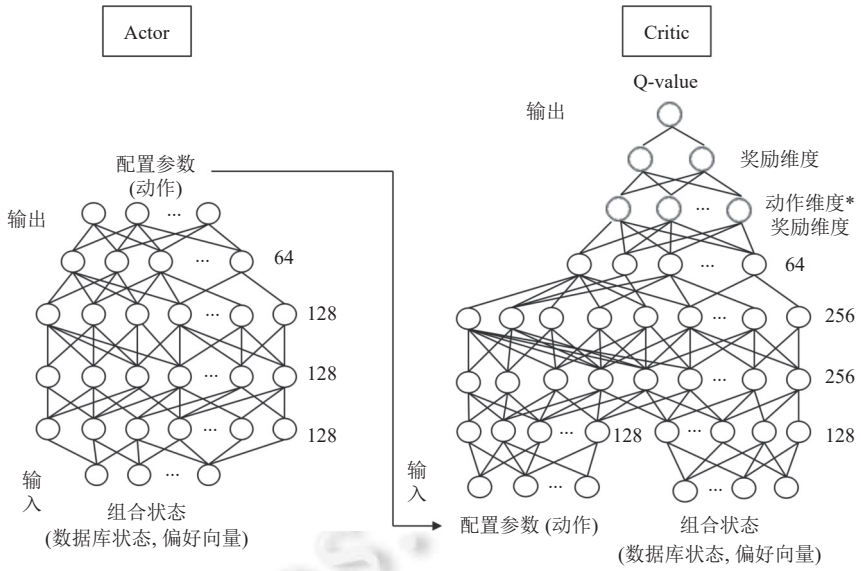



图 9 ON-MODDPG 的网络结构

Critic 损失函数如下:

$$L^A(\theta)=E_{s,a,\omega}\left[\left\|y-Q(s,a,\omega|\theta^Q)\right\|^2\right] \tag{11}$$

$$y=r(s,a)+\gamma\sum_{s'}p(s'|s,a)\max_{a\in A,\omega'\in W}\omega^{\mathrm{T}}q_*(s',a,\omega'),a=\pi'(s',\omega'|\theta^{\pi'}) \tag{12}$$

由于帕累托边界包含大量离散解, 这使得损失函数相当不光滑, 并且 $L^A$ 损失函数是基于均方误差 (MSE) 的, MSE 对于离群点 (outliers) 比较敏感, 因为平方误差会放大这些离群点的影响. 这可能导致损失函数表面出现较深的谷底或较高的山峰, 使得优化路径变得复杂. 即均方误差损失函数导致的局部最优解多, 使得优化过程容易陷入局部最优, 无法找到全局最优解.

因此直接优化 $L^A$ 在实践中具有挑战性. 为了解决这个问题, 采用一个辅助损失函数 $L^B$:

$$L^B(\theta)=E_{s,a,\omega}\left[\left|\omega^{\mathrm{T}}y-\omega^{\mathrm{T}}Q(s,a,\omega|\theta^Q)\right|\right] \tag{13}$$

$L^B$ 使用绝对误差损失函数会导致优化曲面过于平整, 使得梯度较小或为 0, 优化算法难以找到合适的方向进行参数更新.

综合起来, 最终损失函数是:

$$L(\theta)=(1-\lambda)L^A(\theta)+\lambda L^B(\theta) \tag{14}$$

其中, $\lambda$ 是在损失函数 $L^A$ 和 $L^B$ 之间进行权衡的权重. 为了避免使 Critic 损失函数含有过多局部最小值或存在梯度较小难以优化的情况, 综合 $L^A$ 和 $L^B$ 两种损失函数的特点, 采用同伦优化[29]的方法, 在训练过程中将 $\lambda$ 的值从 0 动态增加到 1, 将损失函数从 $L^A$ 转移到 $L^B$. $L^A$ 首先确保 $Q$ 的预测接近于真实的预期总奖励, $L^B$ 提供一个辅助拉力, 具有更好的效用, 如图 10 所示.

基于上述的理论和分析, 对 N-MODDPG 的 Critic 网络的更新方式进行了改进, 实现了 ON-MODDPG 算法, 具体实现细节如算法 2. 与算法 1 不同的是算法 2 第 7 行从偏好采样分布中采样偏好向量作为 Actor 网络输入之一; 第 9 行将状态 $s_t$ 和偏好向量 $\omega_t$ 共同作为 Actor 网络的输入, 第 10 行执行动作 $a_t$ 获得的奖励 $r_t=<r_T,r_L>$ 为向量形式.

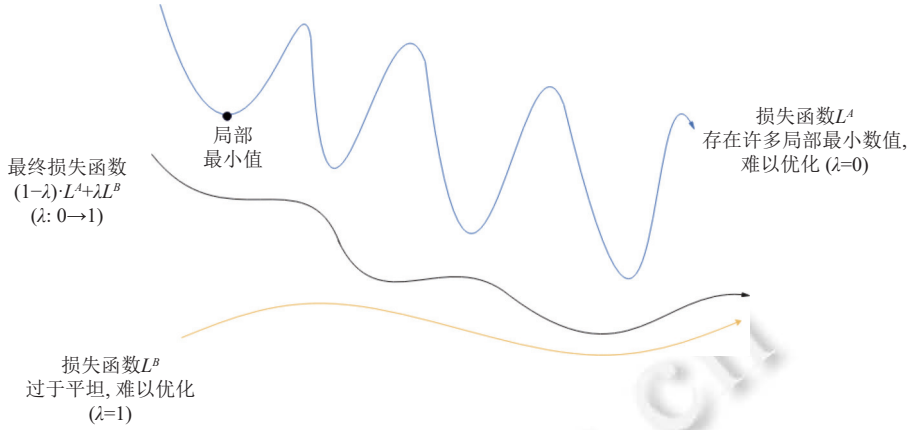



图 10 同伦优化的作用

**算法 2**. ON-MODDPG.

Input: 一个偏好采样分布 $D_\omega$, 平衡权重 $\lambda$ 从 0 增加到 1.

1. $\theta^Q$ 和 $\theta^\pi$ 随机初始化 Critic 网络 $Q(s, a|\theta^Q)$ 和 Actor 网络 $\pi(s|\theta^\pi)$
2. $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$ 初始化目标网络权重参数
3. 初始化优先经验回放池 $D_\tau$ 和偏好采样分布 $D_\omega$, 平衡权重 $\lambda$ 从 0 增加到 1
4. **for** episode = 0 → $E$ **do**
5. 随机噪声 $\mathcal{N}$ 初始化
6. 获得初始状态 $s_0$
7. 采样一个线性偏好 $\omega_0 \sim D_\omega$
8. **for** $t = 0 \rightarrow N$ **do**
9. $a_t = \pi(s_t, \omega_t|\theta^\pi) + \mathcal{N}$
10. 执行动作 $a_t$, 得到向量奖励 $r_t = <r_T, r_L>$ 和状态 $s_{t+1}$, 将 $<s_t, a_t, r_t, s_{t+1}>$ 存入 $D_\tau$
11. **if** *update* **then**
12. 采样批量为 $N_\tau$ 的多维数组 $<s_j, a_j, r_j, s_{j+1}> \sim D_\tau$
13. 采样 $N_\omega$ 个偏好向量, 构成 $W = \{\omega_i \sim D_\omega\}$
14. 计算 $y_{ij} = r_j + \gamma \max\limits_{a\in A,\omega'\in W} \omega_i^{\mathrm{T}} Q(s_{j+1}, a, \omega' \mid \theta^{Q'})$
15. $a = \pi'(s_{j+1,\,\omega'|\theta^{\pi'}})$
16. //其中 $0 \leqslant i \leqslant N_\omega$ and $0 \leqslant j \leqslant N_\tau$
17. //计算损失函数
18. $L^A(\theta^Q) = E_{s,a,\omega}[\|y - Q(s, a, \omega|\theta^Q)\|^2]$
19. $L^B(\theta^Q) = E_{s,a,\omega}[|\omega^{\mathrm{T}} y - \omega^{\mathrm{T}} Q(s, a, \omega|\theta^Q)|]$
20. //最小化损失函数 $L$ 来更新 Critic 网络
21. $\nabla_{\theta^Q} L(\theta^Q) = (1-\lambda)\nabla_{\theta^Q} L^A(\theta^Q) + \lambda \nabla_{\theta^Q} L^B(\theta^Q)$
22. 其中, $\lambda$ 从 0 增加到 1
23. //采样策略梯度更新 Actor 策略网络
24. $\nabla_{\theta^\pi} L(\theta^\pi) \approx \dfrac{1}{N} \sum\limits_{i,j} \nabla_{\theta^\pi} \pi(s_j, \omega_i|\theta^\pi) \nabla_a Q(s_j, a, \omega_i|\theta^Q)|a = \pi(s_j, \omega_i|\theta^\pi)$
25. //更新目标网络

$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$

$\theta^{\pi'} \leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi'}$

**end**

**end**

**end**

算法 2 中第 11–30 行更新 Actor 和 Critic 网络参数, 包括数据采样、偏好向量采样、对目标网络使用多目标 Bellman 最优方程计算 TD Target 来指导 Critic 网络的更新、计算 TD Target 和当前 $Q$ 值的差值 TD-error 构造损失函数 $L^A$、借助辅助损失函数 $L^B$ 增加损失函数的平滑性、使用随机策略梯度更新 Actor 网络、最后更新目标网络.

## 5 数据库性能与内存资源利用率协同优化

已有的相关工作大多是面向一个目标的优化 (吞吐量或延迟), 或者是将吞吐量和延迟两个目标进行线性转换为单目标. 为了阐述本文提出的原生多目标强化学习方法在数据库系统参数优化方面的优势, 本节将数据库系统的优化目标进行扩展, 实现数据库系统性能与内存资源利用率的协同优化, 实现高吞吐量、低延迟、高内存资源利用率 (或低内存资源占有率).

为了在优化吞吐量和延迟的同时减少内存的使用量, 本文为内存占用量也设置了相关的奖励函数.

$$\Delta M = \begin{cases} \Delta M_{t\to 0} = \dfrac{M_t - M_0}{M_0} \\ \Delta M_{t\to t-1} = \dfrac{M_t - M_{t-1}}{M_{t-1}} \end{cases} \tag{15}$$

公式 (1)、公式 (2)、公式 (15) 分别表示吞吐量、延迟和内存占用量变化量, 再根据公式 (4) 计算吞吐量和延迟的奖励.

将奖励形式向量化表示更新为:

$$r = < r_T, r_L, r_M > \tag{16}$$

数据库性能与内存资源利用率协同优化算法的实现与算法 2 相似, 不同之处在于奖励函数的表示和计算方式不同. 需要将算法 2 中的奖励函数的公式从公式 (5) 变为公式 (16).

## 6 实 验

在本节中, 我们评估本文提出的多目标强化学习模型的性能、训练效率、适用性和可扩展性, 并将其与现有的方法 CDBTune$^+$使用的 DDPG 模型, 以及 LlamaTune 中基于 BO 的方法 SMAC 进行比较.

(1) 验证原生多目标强化学习方法 N-MODDPG 模型及其改进的模型 ON-MODDPG 在 MySQL 和 PostgreSQL 数据库参数调优方面的有效性, 以及本文的方法在参数优化方面的效率优势. 实验中使用的数据库系统的参数见附录 A.

(2) 在不同的负载情况下和不同的关系数据库上, 验证本文方法在数据库性能提升方面的有效性和适用性.

(3) 对比不同方法应对用户需求变化的能力, 验证模型的适用性.

(4) 对比协同优化方法 Co-MODDPG 和其他方法, 在优化 MySQL 数据库性能时内存占用率的变化, 验证模型的可扩展性.

### 6.1 实验条件

实验使用了两种基准测试工具 SYSBench (https://github.com/akopytov/sysbench) 和 TPCC-MySQL (https://github.com/Percona-Lab/tpcc-MySQL), 在 4 个工作负载下进行了实验, 分别为 TPC-C 工作负载、SYSBench 的 RW (读写)、RO (仅读) 和 WO (仅写). 在 TPC-C 工作负载下, 生成了包含 200 个仓库 (约 18 GB) 的数据库 (tpcc_test),

将并发线程数设为 32. 对于 SYSBench, 使用 16 个表, 并发线程数设为 100, 详情见表 2. 单次测试时间为 150 s, 基准测试工具测试频率 5 s/次, 取后 10 次的平均值作为单次测试的结果.

实验在 docker 容器中进行, 容器实例详情见表 3, 使用 TPCC-MySQL 基准测试工具的容器实例在服务器 1 上进行, 使用 SYSBench 基准测试工具的容器实例在服务器 2 上进行, 服务器配置如表 4 所示.

表 2 基准测试工具设置

| 工作负载 | 仓库 (表) 个数 | 数据库名称 | 数据库大小 (GB) | 并发线程数 | 单次测试时间 (s) |
|---|---|---|---|---|---|
| TPC-C | 200 | tpcc_test | 18 | 32 | 150 |
| RW | 16 | rw_sbtest | 7.9 | 100 | 150 |
| RO | 16 | ro_sbtest | 7.2 | 100 | 150 |
| WO | 16 | wo_sbtest | 7.2 | 100 | 150 |

表 3 容器实例、硬件配置及工作负载情况

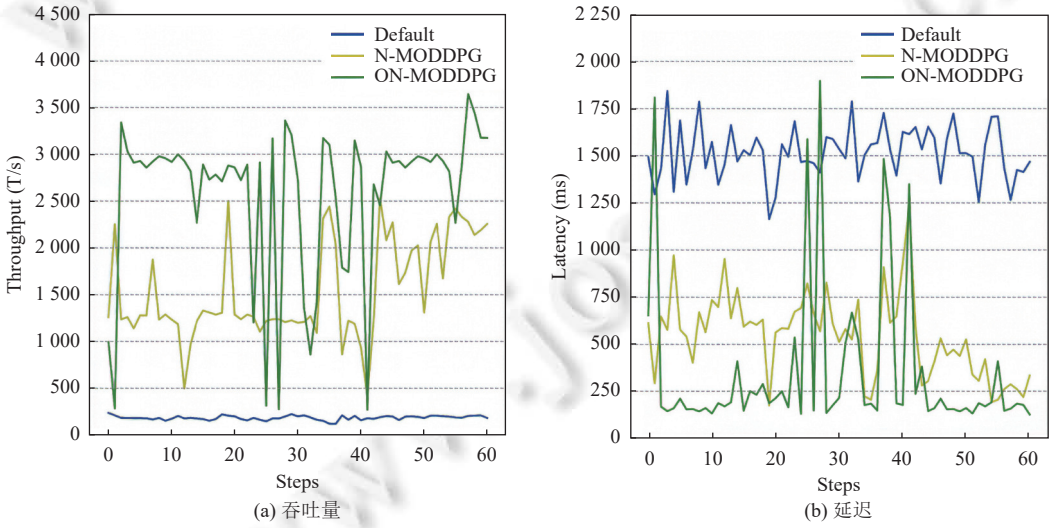
| 容器实例 | 测试工具 | 存储媒介 | 内存大小 (GB) | 磁盘大小 (GB) | 工作负载 |
|---|---|---|---|---|---|
| Tuner1 | TPC-C | SSD | 32 | 250 | TPC-C |
| Tuner2 | SYSBench | SSD | 32 | 300 | RW |
| Tuner3 | SYSBench | SSD | 20 | 150 | RO |
| Tuner4 | SYSBench | SSD | 20 | 300 | WO |

表 4 服务器配置

| 服务器 | CPU核数 | CPU主频 (GHz) | 内存大小 (GB) | 磁盘大小 (TB) |
|---|---|---|---|---|
| 服务器1 | 16 | 2.10 | 94 | 1.7 |
| 服务器2 | 32 | 2.19 | 125 | 1.8 |

### 6.2 数据库参数调优方法有效性验证

本实验在 MySQL 数据库上使用 TPC-C 负载对本文提出的 N-MODDPG 算法与其优化算法 ON-MODDPG 进行了有效性验证, 以上两个算法均部署在容器实例 Tuner1 中. 在图 11 中给出了模型训练阶段的使用参数优化的 MySQL 数据库性能 (Throughput、Latency) 与使用默认配置参数的 MySQL 数据库的性能对比.




(a) 吞吐量 (b) 延迟

图 11 MySQL 数据库参数优化方法的有效性

从图 11 显示的实验结果来看, 在 TPC-C 工作负载下, 使用 N-MODDPG 方法推荐的参数使得 MySQL 数据库系统性能明显优于使用默认配置参数的 MySQL 数据库性能. 采用 N-MODDPG 方法推荐的参数, 可以使 MySQL

数据库的性能的峰值达到 (吞吐量: 2 528.2 T/s, 延迟: 215.9 ms), 相比于使用默认配置参数的 MySQL 数据库的性能 (吞吐量: 228.8 T/s, 延迟: 1 517.7 ms), 其吞吐量提升了 10 倍左右, 延迟大约降低至 1/7 (见表 5).

表 5　各种工作负载下不同方法的性能 (吞吐量 (T/s), 延迟 (ms)) 的平均值和峰值对比

| 工作负载 | MySQL default | DDPG | | ON-MODDPG | |
|---|---|---|---|---|---|
| | | 平均值 | 峰值 | 平均值 | 峰值 |
| TPC-C | (228.8, 1 517.7) | (1 470.4, 1 480.0) | (3 099.6, 190.0) | (2 299.0, 1 042.3) | (3 639.0, 111.9) |
| RW | (125.7, 1 479.4) | (1 611.8, 2 490.7) | (6 393.5, 37.4) | (2 829.3, 1 069.8) | (10 754.6, 23.3) |
| RO | (591.5, 223.3) | (12 268.8, 48.8) | (27 195.3, 9.0) | (12 219.8, 82.7) | (27 596.4, 8.7) |
| WO | (145.8, 1 352.0) | (3 319.6, 408.2) | (24 626.1, 10.9) | (4 133.3, 370.9) | (22 634.2, 10.0) |

使用 ON-MODDPG 方法推荐的优化参数的 MySQL 数据库性能的峰值达到了 (吞吐量: 3 639.0 T/s, 延迟: 111.9 ms), 相比于使用 N-MODDPG 方法推荐的优化参数的 MySQL 数据库, 其吞吐量提升了 44% 左右, 延迟大约降低了 48%.

使用 N-MODDPG 和 ON-MODDPG 方法推荐的参数使得 MySQL 数据库性能比使用默认参数的 MySQL 的性能得到了明显的提升, 主要是因为基于强化学习的数据库参数优化方法可以自动从数据库环境中提取当前的环境状态, 以试错的方式根据当前状态做出动作 (推荐参数) 进而得到对当前动作的奖励, 当数据库重启以使配置参数生效 (即当前数据库状态发生改变), 则重复上述过程以使累积的奖励最大化, 从而推荐可以使数据库达到更高性能的配置参数.

### 6.3　ON-MODDPG 在数据库参数优化方面的优势

为了验证多目标强化学习在数据库参数优化这一多目标优化任务方面的优势, 在容器实例 Tuner1 上对模型 DDPG 和 MODDPG 进行了对比实验. 通过表 6 的实验结果可以看出, 在样本数为 200 的情况下 ON-MODDPG 模型推荐的参数可以使 MySQL 数据库的性能比使用 DDPG 模型推荐的参数的 MySQL 数据库的吞吐量提升了 22%、延迟降低了约 40%; 在样本数为 400 的情况下, 数据库的吞吐量提升了 14%、延迟降低了约 34%. 因此, 多目标强化学习在同样的训练样本量的情况下具有明显的优势, 这得益于 ON-MODDPG 使用了多目标强化学习针对多个优化目标的决策问题的公式化解决方案 (多目标的 Bellman 方程), 其在 Critic 网络更新过程中能够在整个 *CCS* 中快速找到当前偏好下的最优解, 使该最优策略与当前偏好对齐. 如表 6 的实验结果所示, ON-MODDPG 方法可以在同样有限的训练样本下探索到更优的配置参数使数据库系统达到更优的性能.

表 6　探索更优配置参数效率对比

| 样本数 | DDPG最优性能 | | ON-MODDPG最优性能 | |
|---|---|---|---|---|
| | 吞吐量 (T/s) | 延迟 (ms) | 吞吐量 (T/s) | 延迟 (ms) |
| 200 | 2 848.20 | 236.14 | 3 486.10 | 143.43 |
| 400 | 3 071.99 | 192.68 | 3 516.20 | 127.26 |

### 6.4　不同负载下与已有参数调优方法的对比

(1) 基于 TPC-C 的性能对比

将本文使用 ON-MODDPG 算法的模型, 与使用 DDPG 算法的工具 CDBTune$^+$, 以及 LlamaTune 中基于 BO 的算法 SMAC 均部署在容器实例 Tuner1 中, 在模型训练阶段 MySQL 数据库性能 (Throughput、Latency) 变化对比见图 12.

从图 12 显示的实验结果来看, 在 TPC-C 工作负载下, 对 MySQL 数据库系统, 使用 DDPG 方法对其进行参数优化后, 其平均性能为 (1 470.4, 1 480.0), 使用 SMAC 方法对其进行参数优化后, 其平均性能为 (1 749.2, 1 053.1). 改进的 MODDPG 方法对其配置参数进行优化后, 其性能的均值达到了 (2 299.0, 1 042.3) (见表 5), 相比于 DDPG 方法, 平均吞吐量提升近似 56%, 平均延迟降低近似 30%; 相比于 SMAC 方法, 平均吞吐量提升近 31%.

(2) 基于 SYSBench 的性能对比

基于 DDPG 和 ON-MODDPG 的模型, 均使用容器实例 Tuner2, 在 SYSBench 的 RW 工作负载下, 在模型训练阶段, 两者性能对比见图 13.

从图 13 显示的实验结果来看, 在 SYSBench 的 RW 工作负载下, 对 MySQL 数据库系统, 使用 DDPG 方法对其进行参数优化后, 其平均性能为 (1 611.8, 2 490.7); ON-MODDPG 方法对其配置参数 (Knobs) 进行优化后, 其性能的均值达到了 (2 829.3, 1 069.8) (见表 5), 相比于 DDPG 方法, 平均吞吐量提升约 75%, 平均延迟降低约 57%.

(3) 不同工作负载下, 平均性能对比

为了对比本文提出的方法与 CDBTune$^+$使用的 DDPG 方法在数据库参数调优方面的平均性能, 将基于 TPC-C 以及 SYSBench 两个测试基准的 RO、WO 和 RW 工作负载下实验数据处理为平均值的形式展示在图 14 中. 从图 14 显示的实验结果来看, 相比于 DDPG 方法, 在 TPC-C 工作负载下, ON-MODDPG 方法平均吞吐量提升近似 56%, 平均延迟降低近似 30%, 在 SYSBench 的 RW 工作负载下, 平均吞吐量提升近似 75%, 平均延迟降低近似 57%.

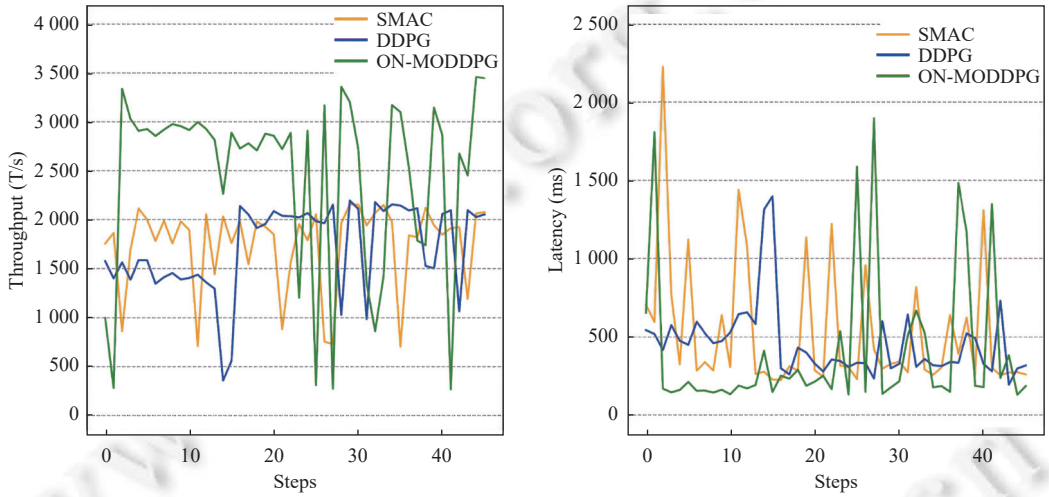



图 12 基于 TPC-C 的 MySQL 数据库参数优化的性能对比

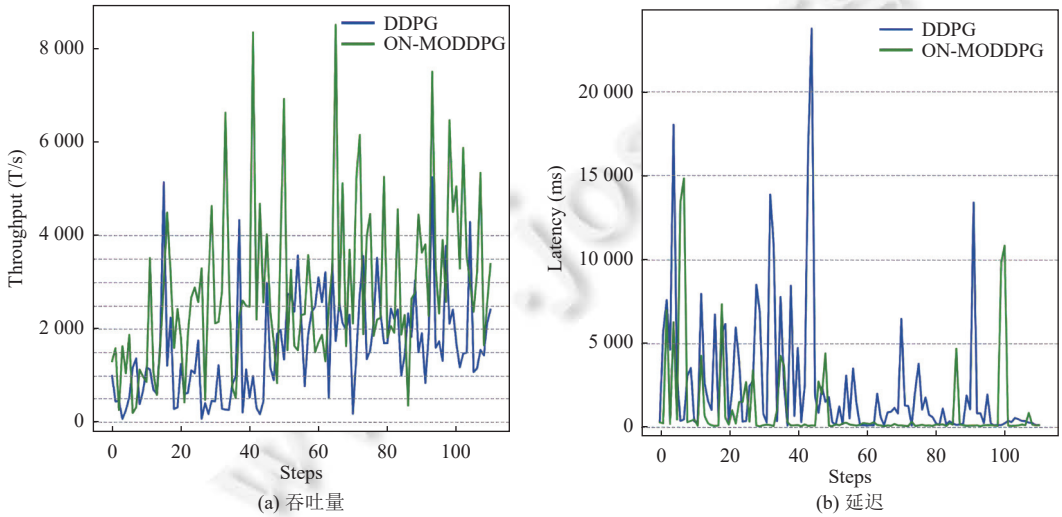



图 13 基于 SYSBench RW 工作负载的 MySQL 数据库参数优化的性能对比

### 6.5 在 PostgreSQL 数据库上与相关的方法进行对比

在本实验中, 将 CDBTune+的 DDPG 算法及本文的 ON-MODDPG 算法部署均在容器实例 Tuner1 中, 对 PostgreSQL 数据库分别使用了 DDPG 和 ON-MODDPG 推荐的优化参数, 并与使用默认参数的 PostgreSQL 数据库进行了性能对比. 在模型训练阶段 PostgreSQL 数据库系统的性能 (吞吐量、延迟) 变化与使用默认参数的 PostgreSQL 数据库的性能对比如图 15 所示.

从图 15 显示的实验结果来看, 在 TPC-C 工作负载下, 对 PostgreSQL 数据库系统使用 ON-MODDPG 方法对其配置参数 (Knobs) 进行优化后, 其性能明显优于 PostgreSQL 的默认配置参数, 同时优于使用 DDPG 方法推荐的参数. 使用 ON-MODDPG 方法推荐的优化参数后, PostgreSQL 数据库的性能的峰值为 (吞吐量: 3 631.7 T/s, 延迟: 8.6 ms), 相比于使用默认配置参数的平均性能 (吞吐量: 414.4 T/s, 延迟: 484.0 ms), 吞吐量提升了 8.7 倍左右, 延迟降低为原来的 1/60.

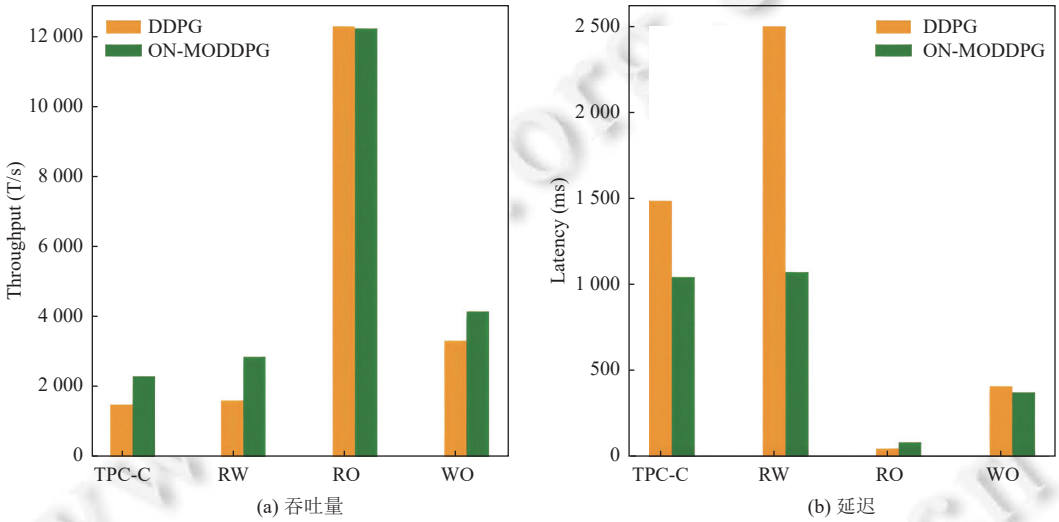



图 14 不同工作负载下 MySQL 数据库参数优化的平均性能对比

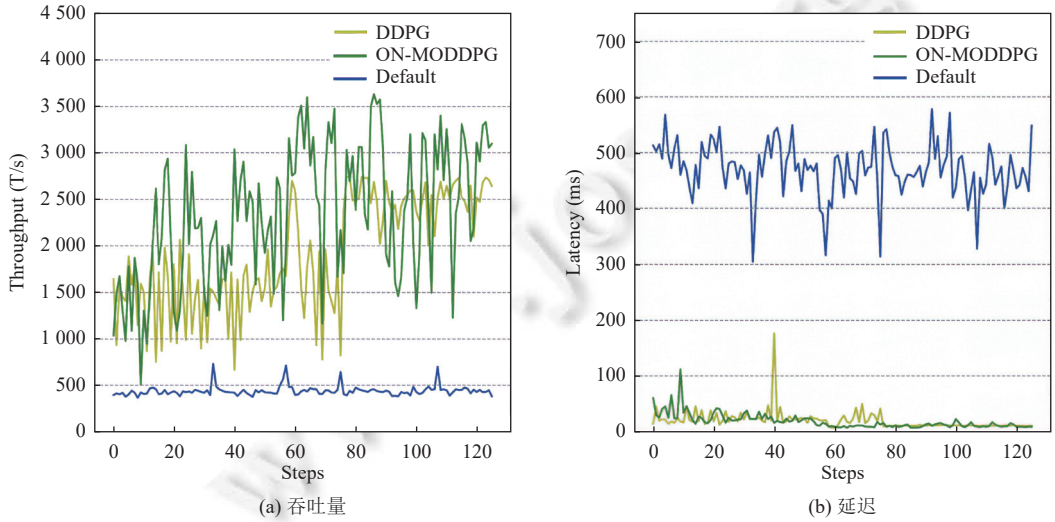



图 15 基于 TPC-C 的 PostgreSQL 数据库参数优化的性能对比

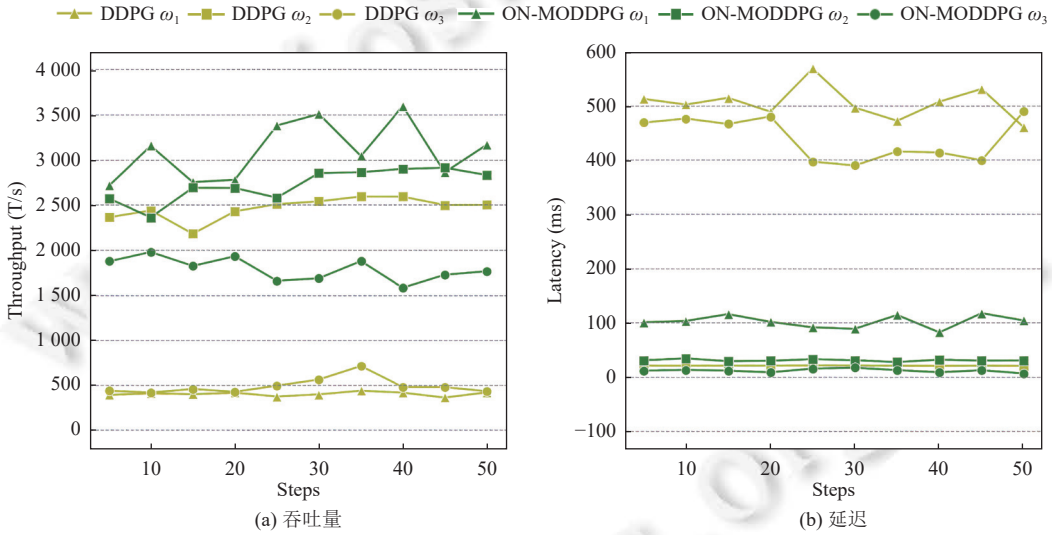
使用 ON-MODDPG 方法对 PostgreSQL 配置参数进行优化后, 与使用 DDPG 方法的性能的峰值相比, 其吞吐量也提升了 32% 左右, 延迟也有明显下降.

### 6.6 参数优化方法应对用户需求变化的适应性对比

当在线调优为数据库系统用户推荐配置参数时, 需要回放用户的工作负载进行压力测试, 传统的单目标强化学习方法 (SORL) 在面对数据库系统环境中硬件配置或工作负载的变化, 可以自动提取在不同环境中真正起作用的特征, 从而简化调优难度, 有效地做出决策, 并适应当前环境, 经过有限步的微调后使数据库系统达到良好的性能.

但是 SORL 无法感知用户对数据库系统调优目标 (吞吐量、延迟) 的偏好变化, 不能快速适应新的目标偏好以满足用户对良好性能的期望. 当偏好发生变化时, 如果使用传统的单目标的强化学习方法, 需要重新设置目标的偏好系数 (见公式 (4)), 即重设奖励函数, 重新进行模型的训练过程; 而本文方法 ON-MODDPG 在训练阶段, 偏好改变是动态发生的, 每个 episode 开始时由偏好发生器生成一个偏好向量, 在模型训练完成后, 只需将需要的偏好向量 $\omega$ 输入模型即可得到应对新偏好的参数配置, 使数据库系统性能保持在较优水准.

在容器实例 Tuner1 中对 PostgreSQL 数据库系统配置参数进行在线调优测试, 分别取偏好向量 $\omega_1 =< 0.9, 0.1 >$、$\omega_2 = < 0.5, 0.5 >$ 和 $\omega_3 = < 0.1, 0.9 >$. 本实验中 CDBTune$^+$使用的 DDPG 模型在训练的过程中使用的偏好为 $\omega_2$, 然后将该模型应用到两个不同的偏好 $\omega_1$ 和 $\omega_3$ 的场景下. 实验结果的对比如图 16 所示.




(a) 吞吐量 (b) 延迟

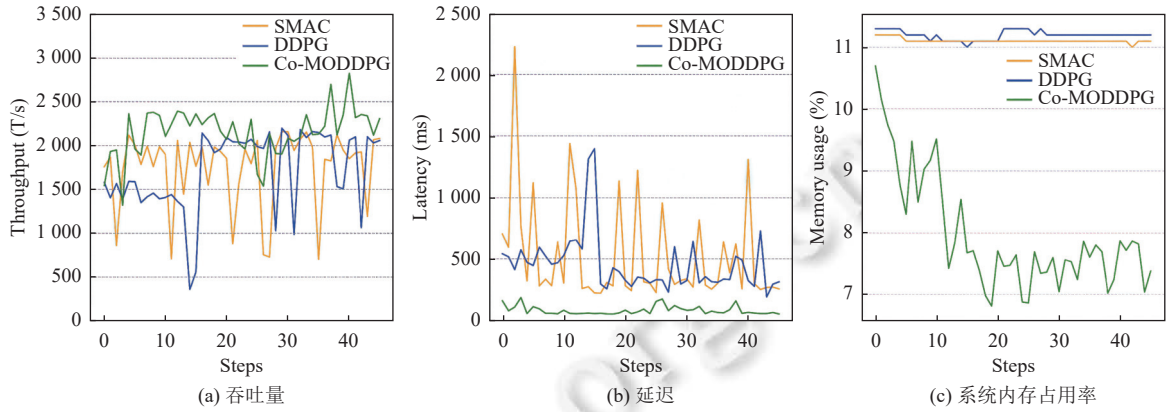
图 16 数据库系统参数优化方法对不同负载的适应性对比

从图 16 显示的实验结果可知, DDPG 方法无法应对偏好的变化. 在偏好 $\omega_2$ 下使用 DDPG 方法对数据库系统参数进行优化可以使其达到良好的性能. 当偏好向量发生变化, 变为 $\omega_1$ 和 $\omega_3$ 时, 如果继续使用偏好 $\omega_2$ 下推荐的参数数据库的性能会有明显的下降, 如图 16 中的吞吐量从 2 500 T/s 下降到了 500 T/s, 系统的延迟也有了大幅的提高. 这说明当偏好向量发生变化时, DDPG 方法需要从数据库系统的默认参数开始进行优化, 模型需要重新训练.

ON-MODDPG 方法应用到不同的偏好 $\omega_1$、$\omega_2$ 和 $\omega_3$ 下, 无论是系统的吞吐量还是系统延迟都可以保持在一个相对稳定的水平. 不同偏好下的系统的性能变化远小于 CDBTune$^+$使用的 DDPG 方法. 这说明 ON-MODDPG 可以动态适应用户对系统的不同偏好. 当用户对系统的偏好发生变化时, 只需要根据当前系统的负载进行有限步的调整, 就能使数据库系统性能保持在较优水平.

### 6.7 Co-MODDPG 在数据库性能和资源利用率的协同优化

将本文提出的 Co-MODDPG 算法模型, 与使用 DDPG 算法的工具 CDBTune$^+$, 以及 LlamaTune 中基于 BO 的算法 SMAC 均部署在容器实例 Tuner1 中, 在模型训练阶段 MySQL 数据库性能 (Throughput、Latency、Memory usage) 变化对比如图 17.




(a) 吞吐量 (b) 延迟 (c) 系统内存占用率

图 17 基于 TPC-C 的 MySQL 数据库性能和资源利用率协同优化的性能对比

从图 17 显示的实验结果来看, 在 TPC-C 工作负载下, 对 MySQL 数据库系统, 使用 DDPG 方法对其进行参数优化后, 其平均性能为 (1 470.4, 1 480.0, 11.2), 使用 SMAC 方法对其进行参数优化后, 其平均性能为 (1 749.2, 1 053.1, 11.1). 改进的 Co-MODDPG 方法对其配置参数进行优化后, 其性能的均值达到了 (2 218.2, 386.9, 7.4), 相比于 DDPG 方法, 平均吞吐量提升约 50%, 平均延迟降低约 73%, 内存占用率降低约 34%; 相比于 SMAC 方法, 平均吞吐量提升约 27%, 平均延迟降低约 63%, 内存占用率降低约 33%.

从图 17 的结果可知, 相比于 DDPG 和 SMAC, ON-MODDPG 扩展后的协同优化算法 Co-MODDPG, 在多目标协同优化方面亦有较好的效果, 能够在取得较好性能 (吞吐量、延迟) 的同时, 尽量减少内存占用率.

## 7 总 结

本文针对数据库系统参数优化的多目标特性提出了一种基于原生多目标的深度强化学习的自动化参数优化方法. 本文的方法改变了以往相关工作中将数据库系统参数优化的多目标任务简化为单目标优化问题带来的实现成本高、适用性差等缺陷, 并将其扩展到更多目标实现了多目标协同优化.

本文首先提出了一种原生多目标的深度强化学习方法 N-MODDPG, 该算法改进了传统强化学习方法中的奖励机制, 避免了传统单目标优化的强化学习方法模型的重复训练问题. 为了实现目标偏好和相应的最优策略的快速对齐, 使得价值函数的更新过程能够考虑整个偏好空间, 本文对 N-MODDPG 方法进行了进一步的优化, 实现了 ON-MODDPG, 避免次优解的产生. 为了在优化数据库性能的同时减少内存资源的占用, 实现了协同优化算法 Co-MODDPG.

为了验证本文所提方法的先进性, 与最新的相关工作在不同的应用场景下进行了对比实验; 为了验证方法的有效性, 在两个主流的关系数据库 MySQL 和 PostgreSQL 中进行了参数优化的对比实验; 为了验证本文方法应对数据库系统需求动态变化的适应性, 针对不同的优化目标的需求与最新的相关方法进行了适应能力的对比. 为了验证本文方法在多目标优化方面的可扩展性, 设置了新的优化目标 (内存占用率), 在关系数据库 MySQL 中和其他方法进行了内存占用对比. 实验结果表明, 本文所提的方法具有一定的先进性、有效性、对多目标优化任务的具有快速适应能力和可扩展性.

未来将探索数据库系统运行时度量参数 (Metrics) 和调优参数 (Knobs) 之间的关系, 进一步确定有效的 Metrics 和 Knobs 的种类和数量, 减少模型的复杂度, 提升模型的训练效率, 增加模型的普遍适用性.

**References:**


[1] Zhao XY, Zhou XH, Li GL. Automatic database knob tuning: A survey. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(12): 12470–12490. [doi: 10.1109/TKDE.2023.3266893]

[2] Zhang J, Zhou K, Li GL, Liu Y, Xie M, Cheng B, Xing JS. CDBTune$^+$: An efficient deep reinforcement learning-based automatic cloud database tuning system. The VLDB Journal, 2021, 30(6): 959–987. [doi: 10.1007/s00778-021-00670-9]

[3] Zhang XY, Chang Z, Li Y, Wu H, Tan J, Li FF, Cui B. Facilitating database tuning with hyper-parameter optimization: A comprehensive experimental evaluation. Proc. of the VLDB Endowment, 2022, 15(9): 1808–1821. [doi: 10.14778/3538598.3538604]

[4] Li GL, Zhou XH. XuanYuan: An AI-native database systems. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 831–844 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

[5] Zhu YQ, Liu JX, Guo MY, Bao YG, Ma WL, Liu ZY, Song KP, Yang YC. BestConfig: Tapping the performance potential of systems via automatic configuration tuning. In: Proc. of the 2017 Symp. on Cloud Computing. Santa Clara: ACM, 2017. 338–350. [doi: 10.1145/3127479.3128605]

[6] Duan SY, Thummala V, Babu S. Tuning database configuration parameters with iTuned. Proc. of the VLDB Endowment, 2009, 2(1): 1246–1257. [doi: 10.14778/1687627.1687767]

[7] Van Aken D, Pavlo A, Gordon GJ, Zhang BH. Automatic database management system tuning through large-scale machine learning. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1009–1024. [doi: 10.1145/3035918.3064029]

[8] Pavlo A, Angulo G, Arulraj J, Lin HB, Lin JX, Ma L, Menon P, Mowry TC, Perron M, Quah I, Santurkar S, Tomasic A, Toor S, van Aken D, Wang ZQ, Wu YJ, Xian R, Zhang TY. Self-driving database management systems. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research. Chaminade: CIDR, 2017. 1.

[9] Li GL, Zhou XH, Li SF, Gao B. QTune: A query-aware database tuning system with deep reinforcement learning. Proc. of the VLDB Endowment, 2019, 12(12): 2118–2130. [doi: 10.14778/3352063.3352129]

[10] Gur Y, Yang DS, Stalschus F, Reinwald B. Adaptive multi-model reinforcement learning for online database tuning. In: Proc. of the 24th Int'l Conf. on Extending Database Technology. Nicosia: EDBT, 2021. 439–444.

[11] Hayes CF, Rădulescu R, Bargiacchi E, Källström J, Macfarlane M, Reymond M, Verstraeten T, Zintgraf LM, Dazeley R, Heintz F, Howley E, Irissappane AA, Mannion P, Nowé A, Ramos G, Restelli M, Vamplew P, Roijers DM. A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-agent Systems, 2022, 36(1): 26. [doi: 10.1007/s10458-022-09552-y]

[12] Kanazawa T, Gupta C. Latent-conditioned policy gradient for multi-objective deep reinforcement learning. In: Proc. of the 32nd Int'l Conf. on Artificial Neural Networks. Heraklion: Springer, 2023. 63–76. [doi: 10.1007/978-3-031-44223-0_6]

[13] Yang RZ, Sun XY, Narasimhan K. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 1311.

[14] Zhang XY, Wu H, Chang Z, Jin SW, Tan J, Li FF, Zhang TY, Cui B. ResTune: Resource oriented tuning boosted by meta-learning for cloud databases. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 2102–2114. [doi: 10.1145/3448016.3457291]

[15] Kanellis K, Ding C, Kroth B, Müller A, Curino C, Venkataraman S. LlamaTune: Sample-efficient DBMS configuration tuning. In: Proc. of the VLDB Endowment, 2022, 15(11): 2953–2965. [doi: 10.14778/3551793.3551844]

[16] Trummer I. DB-BERT: A database tuning tool that "reads the manual". In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 190–203. [doi: 10.1145/3514221.3517843]

[17] Ge JK, Chai YF, Chai YP. WATuning: A workload-aware tuning system with attention-based deep reinforcement learning. Journal of Computer Science and Technology, 2021, 36(4): 741–761. [doi: 10.1007/s11390-021-1350-8]

[18] Cai BQ, Liu Y, Zhang C, Zhang GY, Zhou K, Liu L, Li CH, Cheng B, Yang J, Xing JS. HUNTER: An online cloud database hybrid tuning system for personalized requirements. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 646–659. [doi: 10.1145/3514221.3517882]

[19] Wang JX, Trummer I, Basu D. Demonstrating UDO: A unified approach for optimizing transaction code, physical design, and system parameters via reinforcement learning. In: Proc. of the 2021 Int'l Conf. on Management of Data. New York: ACM, 2021. 2794–2797. [doi: 10.1145/3448016.3452754]

[20] Saglam B, Mutlu FB, Cicek DC, Kozat SS. Actor prioritized experience replay. Journal of Artificial Intelligence Research, 2023, 78: 639–672. [doi: 10.1613/jair.1.14819]

[21] Trummer I. The case for NLP-enhanced database tuning: Towards tuning tools that "read the manual". Proc. of the VLDB Endowment, 2021, 14(7): 1159–1165. [doi: 10.14778/3450980.3450984]

[22] Liu X, Liu SY, Zhuang YK, Gao Y. Explainable reinforcement learning: Basic problems exploration and method survey. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2300–2316 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6485.htm [doi: 10.13328/j.cnki.jos.006485]

[23] Hu TM, Luo B. PA2D-MORL: Pareto ascent directional decomposition based multi-objective reinforcement learning. In: Proc. of the 38th AAAI Conf. on Artificial Intelligence. Vancouver: AAAI Press, 2024. 12547–12555. [doi: 10.1609/aaai.v38i11.29148]

[24] Nguyen TT, Nguyen ND, Vamplew P, Nahavandi S, Dazeley R, Lim CP. A multi-objective deep reinforcement learning framework. Engineering Applications of Artificial Intelligence, 2020, 96: 103915. [doi: 10.1016/j.engappai.2020.103915]

[25] Reymond M, Bargiacchi E, Nowé A. Pareto conditioned networks. In: Proc. of the 21st Int'l Conf. on Autonomous Agents and Multiagent Systems. Auckland: Int'l Foundation for Autonomous Agents and Multiagent Systems, 2022. 1110–1118.

[26] Castanet N, Sigaud O, Lamprier S. Stein variational goal generation for adaptive exploration in multi-goal reinforcement learning. In: Proc. of the 40th Int'l Conf. on Machine Learning. Honolulu: JMLR, 2023. 151.

[27] Hu TM, Luo B, Yang CH, Huang TW. MO-MIX: Multi-objective multi-agent cooperative decision-making with deep reinforcement learning. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2023, 45(10): 12098–12112. [doi: 10.1109/TPAMI.2023.3283537]

[28] Alegre LN, Bazzan ALC, Roijers DM, Nowé A, da Silva BC. Sample-efficient multi-objective learning via generalized policy improvement prioritization. In: Proc. of the 2023 Int'l Conf. on Autonomous Agents and Multiagent Systems. London: Int'l Foundation for Autonomous Agents and Multiagent Systems, 2023. 2003–2012.

[29] Watson LT, Haftka RT. Modern homotopy methods in optimization. Computer Methods in Applied Mechanics and Engineering, 1989, 74(3): 289–305. [doi: 10.1016/0045-7825(89)90053-4]

**附中文参考文献**:

[4] 李国良, 周煊赫. 轩辕: AI 原生数据库系统. 软件学报, 2020, 31(3): 831–844. http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

[22] 刘潇, 刘书洋, 庄韫恺, 高阳. 强化学习可解释性基础问题探索和方法综述. 软件学报, 2023, 34(5): 2300–2316. http://www.jos.org.cn/1000-9825/6485.htm [doi: 10.13328/j.cnki.jos.006485]

## 附录 A

本文中实验部分使用的 MySQL 的参数和 PostgreSQL 的参数见表 A1 和表 A2.

表 A1 MySQL 推荐参数 Knobs 调优详情

| Knobs | MySQL 默认值 | 初始值 | CDBTune+ | | | | ON-MODDPG | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | TPC-C | RW | RO | WO | TPC-C | RW | RO | WO |
| binlog_format | Statement | Mixed | Mixed | Mixed | Row | Row | Mixed | Mixed | Row | Mixed |
| innodb_buffer_pool_size | 128 MB | 4 GB | 6 GB | 8 GB | 8 GB | 8 GB | 8 GB | 8 GB | 8 GB | 8 GB |
| innodb_log_files_in_group | 2 | 2 | 8 | 12 | 1 | 2 | 12 | 12 | 11 | 2 |
| innodb_log_file_size | 48 MB | 0.5 GB | 3.3 GB | 0.1 GB | 15 GB | 39 GB | 8.3 GB | 8.3 GB | 3.9 GB | 32 GB |
| innodb_read_io_threads | 4 | 12 | 3 | 1 | 1 | 31 | 52 | 64 | 1 | 28 |
| binlog_cache_size | 32 KB | 32 MB | 0.7 GB | 0.4 GB | 4 KB | 4 KB | 1 GB | 4 KB | 0.7 GB | 0.8 GB |
| innodb_buffer_pool_instances | 8 | 8 | 8 | 1 | 1 | 8 | 14 | 2 | 6 | 25 |
| max_binlog_cache_size | 18.4 EB | 4 GB | 4 GB | 1.5 GB | 4.7 GB | 0.2 GB | 3.5 GB | 3 GB | 4.4 GB | 5 GB |
| binlog_checksum | CRC32 | None | None | CRC32 | CRC32 | CRC32 | None | CRC32 | CRC32 | None |
| innodb_purge_threads | 1 | 1 | 25 | 15 | 25 | 29 | 4 | 21 | 32 | 11 |
| max_binlog_size | 1 GB | 1 GB | 0.4 GB | 0.3 GB | 69 MB | 0.2 GB | 0.2 GB | 0.6 GB | 0.2 GB | 0.3 GB |
| innodb_write_io_threads | 4 | 12 | 52 | 1 | 2 | 64 | 64 | 64 | 1 | 8 |
| skip_name_resolve | Off | On | On | On | Off | On | Off | Off | On | On |
| innodb_file_per_table | On | On | Off | Off | Off | On | Off | On | On | Off |
| table_open_cache | 2000 | 512 | 524288 | 524288 | 397935 | 524288 | 442241 | 380371 | 16715 | 216686 |
| max_connections | 151 | 1600 | 50000 | 7144 | 7910 | 47208 | 50000 | 48731 | 20928 | 20860 |

表 A2 PostgreSQL 推荐参数 Knobs 详情

| Knobs | 默认值 | CDBTune+ TPC-C | ON-MODDPG TPC-C |
|---|---|---|---|
| shared_buffers | 128 KB | 6 GB | 6 GB |
| max_wal_size (GB) | 1 | 2 | 8 |
| min_wal_size (MB) | 80 | 39 | 39 |
| effective_cache_size (GB) | 4 | 4 | 0.4 |
| bgwriter_lru_maxpages | 100 | 854 | 31 |
| bgwriter_delay | 200 ms | 10 s | 1.5 s |
| checkpoint_completion_target | 0.5 | 0.5 | 0.5 |
| deadlock_timeout | 1 s | 1 ms | 1 ms |
| default_statistics_target | 100 | 5 101 | 10 000 |
| effective_io_concurrency | 1 | 1 000 | 952 |
| checkpoint_timeout (min) | 5 | 8 | 33 |

**荣垂田**(1981－), 男, 博士, 教授, CCF 专业会员, 主要研究领域为数据库系统, 大数据分析技术.

**杜方**(1974－), 女, 博士, 教授, CCF 高级会员, 主要研究领域为大数据分析, 人工智能, 生物医学多模态数据分析与挖掘.

**田浩辉**(2000－), 男, 硕士生, 主要研究领域为数据库系统智能优化.