时序数据库预计算推荐系统研究与实现

夏小芳[1] 孙路明[2] 穆长春[1] 李东阳[1] 李 辉[1),2)] 崔江涛[1] 梁 炜[3]

[1)](西安电子科技大学计算机与科学技术学院 西安 710071)

[2)](上海沄熹科技有限公司 上海 200120)

[3)](中国科学院沈阳自动化研究所 沈阳 100016)

摘 要 时序数据库是一种专门用于处理和存储时间序列数据的数据库管理系统,如何提高其查询性能一直是数据分析领域重要的研究课题。在时序数据库中,聚合查询是数据分析的核心操作之一。然而,随着数据规模的不断增长,聚合查询的性能问题逐渐凸显,直接对原始数据进行聚合计算会导致查询响应时间过长,严重影响系统整体性能和用户体验。预计算技术通过在数据写入时预先计算并存储聚合结果,避免了在聚合查询时对原始数据进行实时计算,从而有效地减小查询延迟。然而,现有预计算机制创建和配置的过程复杂,严重依赖专业的数据库管理员,普通用户难以掌握。针对上述问题,本文设计了一种面向时序数据库的预计算推荐系统,该系统通过自动收集并分析数据库中的历史工作负载,生成预计算候选,并使用深度学习模型估计预计算候选的收益,最终将预计算选择问题建模为整数规划问题并求解以选择最优的预计算集合。实验结果表明,预计算推荐系统能够基于历史负载特征有效生成预计算推荐方案,使查询时间平均降低约44.3%。与其他推荐算法相比,本文提出的预计算推荐算法不仅能够灵活地设置成本上限,还能在相同成本约束条件下实现更优的查询收益,查询收益提升幅度从2.3%到32.18%不等。

关键词 时序数据库;预计算;聚合查询优化;深度学习;整数规划

中图法分类号 TP39 **DOI号** 10.11897/SP.J.1016.2026.01137

Research and Implementation of Precomputation Recommendation System for Time-Series Databases

XIA Xiao-Fang[1] SUN Lu-Ming[2] MU Chang-Chun[1] LI Dong-Yang[1]
LI Hui[1),2)] CUI Jiang-Tao[1] LIANG Wei[3]

[1)](School of Computer Science and Technology, Xidian University, Xi'an 710071)

[2)](Shanghai Yunxi Technology Co., Ltd., Shanghai 200120)

[3)](Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 100016)

Abstract A time series database (TSDB) is a specialized database management system designed for storing, managing, and processing large volumes of time-stamped data generated continuously over time. TSDBs are widely applied in domains such as Internet of Things monitoring, industrial automation, financial analytics, cloud infrastructure management, and smart city systems. These scenarios are characterized by high data ingestion rates, long-term data retention, and diverse analytical requirements, imposing stringent demands on database performance and scalability. Efficient query processing has long been a fundamental challenge in data management, which is par-

收稿日期:2025-05-30;在线发布日期:2026-01-29。本课题得到京津冀环境综合治理国家科技重大专项(No.2025ZD1200600)、国家自然科学基金面上项目(No.62372360、No.62372352)、国家自然科学基金青年科学基金项目(No.62403456)、陕西省创新能力支撑计划项目(No.2024ZC-KJXX-021)、济南市科技计划(No.202428001)资助。**夏小芳(通信作者)**,博士,副教授,博士生导师,主要研究领域为物联网、数据库、数据安全、云边协同等。E-mail:xiaxiaofang@xidian.edu.cn。**孙路明**,博士,主要研究领域为智能数据库查询优化。**穆长春**,硕士研究生,主要研究领域为时序数据库查询优化。**李东阳**,硕士研究生,主要研究领域为数据库查询优化。**李 辉**,博士,教授,博士生导师,主要研究领域为数据挖掘、保护隐私的数据查询和检索等。**崔江涛**,博士,教授,博士生导师,主要研究领域为数据库管理与内核技术、数据工程等。**梁 炜**,博士,研究员,博士生导师,主要研究领域为工业无线传感器网络、多传感器数据融合、工业检测与故障诊断等。

ticularly pronounced in time series databases due to the massive scale and continuous growth of time-stamped data.

Among various query types, aggregation queries play a central role in time series analysis. They are commonly used to compute summary statistics, including averages, sums, and extrema, over specified time intervals and dimensions, forming the basis for monitoring, reporting, and decision-making. However, as data volumes grow, executing aggregation queries directly over raw time series data becomes increasingly costly. Frequent scans of large datasets and on-the-fly computation of aggregates often result in high query latency, adversely affecting system throughput, scalability, and user experience, especially under high-concurrency workloads.

Precomputation is an effective optimization technique to mitigate the performance overhead of aggregation queries. By precomputing and materializing aggregated results during data ingestion or offline processing, a database system can significantly reduce query execution cost, as queries can be answered using precomputed summaries instead of raw data. Despite its effectiveness, existing precomputation mechanisms in TSDBs are often difficult to configure and maintain. They usually require database administrators to manually analyze query workloads, select appropriate aggregation dimensions and time granularities, and tune system parameters. This process is labor-intensive, heavily reliant on expert knowledge, and lacks adaptability to evolving workloads, thereby limiting the practical adoption of precomputation strategies.

To address these limitations, this paper proposes an automated precomputation recommendation system for time series databases, aiming to reduce manual configuration effort while dynamically adapting to changing query workloads. The system continuously collects and analyzes historical workload information, including query patterns, aggregation types, accessed time ranges, and execution frequencies. Based on this analysis, it automatically generates candidate precomputations expected to improve future query performance. To accurately estimate the effectiveness of each candidate, the system employs a deep learning-based prediction model to predict potential query benefits, such as latency reduction and overall performance improvement. Practical constraints, including storage space and maintenance overhead, are explicitly considered. The selection of precomputations is formulated as an integer programming problem, with the objective of maximizing query benefits under a user-defined cost upper bound. This approach enables systematic balancing of performance gains against resource consumption, identifying an optimal set of precomputations that satisfies both efficiency and cost constraints.

Extensive experiments demonstrate the effectiveness of the proposed system. Results indicate that it can derive precomputation schemes from historical workload characteristics, reducing average query latency by approximately 44.3%. Furthermore, compared with existing recommendation algorithms, the proposed approach supports flexible cost upper bounds and achieves superior query benefits under identical cost budgets, with improvements ranging from 2.3% to 32.18%.

Keywords time-series database; precomputation; aggregate query optimization; deep learning; integer programming

1 引 言

近年来,随着工业物联网技术的迅猛发展[1],工业制造正加速迈入数字化时代。各类设备和传感器持续生成海量时间序列数据(简称“时序数据”),逐渐成为数字经济中的核心生产要素。尽管某些关系型数据库也能用于管理时序数据,但由于时序数据具有数据量大、增长速度快等显著特点,传统通用数据库在面对大规模时序数据时,性能会迅速下降,难

以支撑大规模传感网络的高效数据处理需求。为此,专门面向时序数据存储与管理的时序数据库应运而生,有效弥补了通用数据库在时序数据管理方面的性能瓶颈与功能局限。早在 20 世纪 90 年代末,时序数据库 RRDtool[2] 就已问世,主要用于处理网络带宽、温度监测等场景中产生的时序数据。然而,直到近年来,时序数据库才真正迎来了快速发展期。根据 DB-Engines[3] 排名,从 2018 年起,时序数据库的关注度显著上升,其受欢迎程度跃升至数据库类别中的第二位[4]。

时序数据库广泛应用于物联网、金融、监控等领域。在这些应用场景中,聚合查询(如求和、平均值、最大值、最小值等)是数据分析的核心操作之一。其重要性体现在能够从海量的时序数据中提取关键信息,帮助用户快速理解数据趋势、发现异常,进而提供强有力的决策支持[5]。然而,随着数据规模的持续扩大,聚合查询的性能瓶颈日益显现。由于时序数据通常具有高写入吞吐量和长时间跨度的特点,若直接基于原始数据进行聚合计算,往往会导致查询响应时间过长,从而影响系统整体性能和用户体验。因此,如何高效聚合查询,已然成为时序数据库设计中的关键技术挑战之一[6]。

为加速聚合查询,预计算技术被广泛应用于时序数据库中。该技术通过在数据写入阶段预先计算并存储聚合结果,避免了查询时的实时计算,从而显著降低了查询延迟。例如,在需要频繁获取某一时间段内的平均值、最大值或总和等指标时,数据库系统可以提前计算这些聚合结果并存储,查询时直接读取预计算值,省去了对大量原始数据的重复扫描与计算。对于高频查询场景,这种方式尤为有效,不仅能大幅提升响应速度,还能显著减轻系统负载。预计算技术的引入,不仅极大提升了时序数据库的查询性能,还降低了系统资源的消耗,为大规模时序数据的高效分析提供了坚实支撑。

尽管预计算技术在提升查询性能方面表现出色,但其实际应用仍面临诸多挑战。首先,预计算的创建与配置过程较为复杂,涉及多个关键参数的设置,如选择需要预计算的列、确定适用的聚合函数(如求和、平均值、最大值等),以及设定合适的聚合粒度(如按秒、分钟或小时进行汇总)。这些参数的合理设定通常依赖于对数据特征和查询模式的深入理解,严重依赖经验丰富的数据库管理员,导致预计算技术的使用门槛较高。此外,现有的预计算推荐算法往往缺乏对存储成本的综合考量,灵活性不足,难以根据实际业务需求优化预计算推荐。对于普通用户而言,预计算创建与配置的复杂性限制了其在实际场景中的广泛应用。

针对以上问题,本文旨在设计一种面向时序数据库的预计算推荐系统,该系统自动收集并分析时序数据库中的历史工作负载,基于负载分析结果,合并具有共同特征的查询以生成预计算候选。同时数据库系统以存储空间作为核心指标计算预计算候选的成本,并借助深度学习模型对预计算候选的潜在收益进行精准评估。最终通过将预计算选择问题建模为整数规划问题进行求解以选择最优的预计算集合。仿真实验结果表明,该系统显著提升了复杂查询的效率,与现有的预计算推荐算法相比,本文提出的算法能够有效控制成本上限,从而实现更强的灵活性。此外,预计算推荐系统降低了预计算技术的使用的门槛,使得普通用户无需具备深厚的数据库背景知识即可进行查询优化。传统方式下,用户需逐条手工撰写并配置包含聚合函数、粒度等多种参数的预计算创建语句;而借助预计算推荐系统,用户仅需执行一条无需任何参数设置的推荐指令,系统即可自动生成并输出一组收益显著的最优预计算方案。

综上所述,本文研究的面向时序数据库的预计算推荐系统对于提高大规模时序数据场景下的查询效率有重要的研究意义,对物联网、金融、监控等领域的数据管理具有重要意义。

2　相关工作

2.1　预计算技术

预计算技术是时序数据库中一项重要的功能,它能够自动定期执行时序聚合查询,并将查询结果存储到指定的数据表中。例如,用户可以设置每小时对过去一小时的数据进行聚合计算,从而实现对数据的实时监控和分析。预计算技术有两个重要用途:首先,它可以用于数据降采样,将高频率采样的数据进行降采样处理后保存,从而减少存储空间的占用;其次,它能够提前计算一些代价昂贵的复杂查询,并将结果保存起来,这样可以显著加快后续查询的速度。这两个用途使得预计算技术在物联网、金融等领域应用非常广泛。

基本上所有的时序数据库都支持预计算功能,OpenTSDB[7] 中实现了预聚合,预聚合是一种预先计算并存储好的聚合数据,在 OpenTSDB 中,用户

可以通过定义预先计算的聚合函数(例如,求和、平均值、最大值、最小值等)和时间间隔来创建预聚合。一旦创建了预聚合,系统就会在后台定期计算并存储好这些聚合数据,用户可以在查询时直接使用这些预先计算好的聚合数据,而不需要实时计算。InfluxDB[8]支持连续查询的功能,连续查询通过定期拉取原始时序数据进行计算,将计算结果存储在内部特殊的存储结构中。用户通过创建连续查询来实现对数据预处理。TDengine[9]中同样实现了预计算的功能,提供基于数据块的自定义预计算功能。如果查询处理涉及整个数据块的全部数据,直接使用预计算结果,完全不需要读取数据块的内容。这些预计算技术的共同点在于通过预先计算和存储聚合数据,减少实时计算的开销,同时优化存储效率和查询性能,为大规模时序数据处理提供了强有力的支持。

预计算的实现方式有很多种,块级索引[10]、物化视图和 Cube 预计算[11]是数据库中预计算实现的三种方式,它们各自具有独特的特点和适用场景。块级索引适合大规模数据集的轻量级索引需求,物化视图适合复杂查询的加速,而 Cube 预计算则专为多维分析设计。在实际应用中,可以根据具体的业务需求和数据特点,选择合适的技术或组合使用,以在存储开销和查询性能之间取得最佳平衡。

下面介绍预计算的使用方法,创建预计算的语法结构通常包括预计算名称、聚合函数、聚合粒度和原始表。以 KaiwuDB[12]为例,以下是一条创建预计算的 SQL 语句示例:"CREATE Precomputing pre AS SELECT max(usage) FROM cpu WHERE device='device1' Interval('30s')"。这条语句会创建一个名为 pre 的新表,之后系统会每隔 30 秒自动计算最近 30s 内 device1 设备上的 cpu 表中 usage 字段的最大值,并将结果写入 pre 表。通过匹配预计算,数据库能够高效地对时序数据进行聚合查询,为用户提供强大的数据处理能力。预计算的创建与配置过程较为复杂,涉及多个关键参数的设置,对普通用户极为不友好,借助本文实现的预计算推荐系统,用户只需要输入一条推荐语句:Recommend Precomputings For Table table,便会得到大量收益显著的预计算结果,相比于之前复杂的创建流程,极大降低了预计算的使用门槛。

当一条聚合查询被执行时,数据库会根据以下规则匹配相应的预计算:(1)聚合粒度匹配:查询的聚合粒度必须和预计算聚合粒度相等或者是其整数倍。例如,预计算的聚合粒度为 60 秒,查询的聚合粒度为 300 秒,则查询可以匹配到该预计算。(2)设备匹配:查询的目标设备是否与预计算的目标设备一致或者查询包含预计算的目标设备。例如,查询的设备 ID 为 device1 和 device2,预计算的设备为 device1、device2 或者为 device1 和 device2,那么查询可以匹配到该预计算。(3)聚合函数匹配:查询所需的聚合函数是否与预计算一致或者是否是预计算聚合函数的子集。例如,如果预计算中的聚合函数是 max(col1)和 sum(col2),而查询语句中的聚合函数是 max(col1)、sum(col2)或者为 max(col1)和 sum(col2),则该查询可以匹配到该预计算。

2.2 物化视图自动生成

在设计预计算推荐系统的过程中,本文主要参考了关系型数据库物化视图自动生成的设计,物化视图自动生成可以划分为三个部分,分别是(1)物化视图候选生成;(2)物化视图估计;(3)物化视图选择。本节会从这三个部分来介绍物化视图自动生成,第三章会介绍预计算推荐系统的设计与实现。

物化视图候选生成。物化视图候选生成是自动生成物化视图的第一步,目标是生成一组可能对查询优化有帮助的视图集合。研究主要集中在如何高效生成候选视图,同时避免生成过多无效候选。传统的物化视图候选生成方法通过分析历史查询日志,提取频繁出现的查询模式或子查询,生成候选视图[13-17]。为了使物化视图更通用,一些研究通过重写子查询以找到等价的子查询[13],进而合并子查询,有两种主要的方法用于查询等价性检查和重写,使用图[18]和规则[19]。具体而言,使用图的方法是通过构建查询的图表示,利用子图匹配和图同构算法来检查查询等价性,并借助图编辑操作对查询进行重写优化;而使用规则的方法则是定义等价性和重写规则,通过模式匹配和规则应用来检查查询等价性并进行重写。关系型数据库中物化视图候选生成方法在时序数据库中是无法直接使用的,因为时序场景下的工作负载带有时间信息,这使得传统的物化视图候选生成方法难以适应。本文需要针对时序数据库中的工作负载特点设计预计算候选生成算法。

物化视图估计。物化视图估计的目标是评估候选视图的潜在收益,包括查询性能提升、存储成本和维护开销等。研究主要集中在如何准确估计视图的收益和代价。传统的方法是通过构建代价模型,估计物化视图对查询性能的提升以及存储和维护的开

销,例如,Baralis[20]等人提出了基于查询重写代价的估计方法。为了使估计更准确,一些研究开始采用基于基数估计来估计物化视图的收益和成本,Horng 等人[21]将查询和更新频率分配给每个物化视图候选,并使用基数来估计查询和物化视图的成本。最近,大量基于深度学习的收益估计方法[22-26]被提出以更准确地估计成本和基数,因为它们可以捕获查询和视图之间的相关性,所以取得了不错的效果。因此,本文将深度学习方法引入时序数据库的预计算候选收益的计算中。

物化视图选择。物化视图选择是从候选视图中选择最优的子集,以最大化查询性能提升,同时最小化存储和维护成本。这是一个典型的组合优化问题。物化视图选择有许多启发式方法,Horng 等人[21]提出了一种遗传算法,通过在染色体表示中编码查询计划和物化视图状态来选择物化视图。Sohrabi 等人[27]提出了一种频繁项集挖掘方法,用于从查询中选择物化视图。Gosain 等人[28]为物化视图磁盘空间和维护成本设计了一个惩罚函数,以使启发式方法更好地了解物化视图成本。然而,启发式方法依赖于对数据/工作负载分布的一些假设,当数据分布或查询模式发生变化时,启发式方法可能需要重新调整启发式规则或参数,否则性能会大幅下降。因此,深度学习方法通过从经验中学习来解决这一问题,Han 等人[29]介绍了一种强化学习模型以选择物化视图,取得了不错的效果。物化视图的候选种类和数量相较于预计算候选而言规模庞大,上述求解算法主要针对大规模组合优化问题而设计。然而,预计算选择问题的求解相对较为简单,因此本文采用求解器来实现高效求解。

2.3　KaiwuDB 时序数据模型

KaiwuDB 是一款由浪潮集团打造的面向 AIoT(Artificial Intelligence of Things)场景的分布式、多模融合、支持原生 AI 的数据库产品,支持同一实例同时建立时序库和关系库并融合处理多模数据,具备时序数据高效处理能力,具有稳定安全、高可用、易运维等特点。面向工业物联网、数字能源、车联网、智慧产业等领域,KaiwuDB 提供一站式数据存储、管理与分析的基座。

本节将详细介绍 KaiwuDB 中时序数据模型,并以数字能源场景下的智能电表作为时序数据的典型应用场景。如表 1 所示,某型号的智能电表能够采集电流、电压和相位这 3 个模拟量。同时每块智能电表还具有位置和分组等静态属性。基于此,

表 1　智能电表数据

设备	时间戳	电流	电压	相位	位置	分组
d1001	1538548685000	10.3	219	0.31	city_a	2
d1002	1538548684000	10.2	220	0.23	city_b	3
…	…	…	…	…	…	…

KaiwuDB 中时序数据模型由以下几点构成:

(1)采集量

采集量是指通过各种传感器、设备或其他类型的采集点所获取的物理量,如电流、电压、温度、压力、GPS 等。由于这些物理量随时间不断变化,因此采集的数据类型多样,包括整型、浮点型、布尔型以及字符串等。随着时间的积累,存储的数据将持续增长。以智能电表为例,其中的电流、电压和相位便是典型的采集量。

(2)标签

标签是指附着在传感器、设备或其他类型采集点上的静态属性,这些属性不会随时间发生变化,例如设备型号、颜色、设备所在地等。标签的数据类型可以是任意类型。在智能电表的示例中,位置和分组就是典型的标签。

(3)数据采集点

数据采集点是指在一定的预设时间周期内或受到特定事件触发时,负责采集物理量的硬件或软件设备。一个数据采集点可以同时采集一个或多个采集量,但这些采集量都是在同一时刻获取的,并拥有相同的时间戳。在智能电表的示例中,d1001、d1002 等标识符即代表了不同的数据采集点。

(4)表

KaiwuDB 采用传统的关系型数据库模型来管理数据。同时,为了充分发挥时序数据的特性,KaiwuDB 采取了“一个数据采集点一张表”的设计策略,即要求为每个数据采集点单独建立一张表。例如,若有千万块智能电表,则在 KaiwuDB 中需要创建相应数量的表。在智能电表的示例数据中,设备 ID 为 d1001 的智能电表对应着 KaiwuDB 中的一张表,该电表采集的所有时序数据均存储于此表中。这种设计方式既保留了关系型数据库的易用性,又充分利用了时序数据的独特优势。

KaiwuDB 的时序数据模型为本文开展时序数据库的预计算推荐系统研究奠定了坚实的基础。

3　系统架构

本节介绍面向时序数据库的预计算推荐系统的

整体框架。如图 1 所示，该框架主要分为三个模块，分别是数据采集模块、负载分析模块和预计算推荐模块。在数据采集模块中，系统通过采集历史工作负载信息和预计算元信息，从而为后续的负载分析和预计算推荐提供数据支持。在负载分析模块中，系统通过历史工作负载信息和预计算元信息进行分析，以深入理解用户的行为模式，进而为设计预计算推荐算法提供坚实的基础。在预计算推荐模块中，设计了一种基于收益的预计算推荐算法，该算法采用了深度学习方法对预计算收益进行计算，能够根据负载分析的结果推荐出收益最高的预计算结果。通过上述三层架构，构成了时序数据库的预计算推荐系统，用户仅需执行一条预计算推荐语句，即可获得兼顾成本与性能的最优预计算方案，显著降低了使用门槛。下文将详细介绍每一个模块的具体实现。

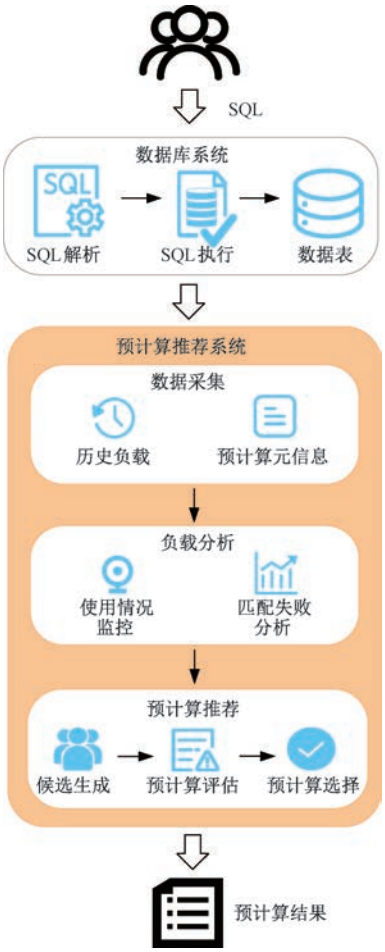

图 1 预计算推荐系统整体框架

3.1 数据采集模块

数据采集模块是预计算推荐系统中的入口，负责采集数据库中的历史负载信息和预计算元信息，这些信息的收集对于深入理解数据库的使用模式、用户行为以及优化查询性能至关重要。

采集的信息可以分为两类：(1)历史工作负载信息：历史工作负载信息主要是用户的历史查询 SQL 记录，通过对历史负载信息的分析，系统能够识别出常见的查询模式和用户偏好，这对于设计好的预计算推荐算法而言至关重要。本文收集的工作负载类别为："SELECT 聚合函数 FROM 目标表 WHERE device=设备编号 AND 时间范围=[起始时间，终止时间]Interval(聚合粒度)"，以 TSBS(Time Series Benchmark Suite)[30] 中 CPU-ONLY 场景下中的一条查询"SELECT max(usage)，max(usage_system) FROM cpu Where device in('device 1'，'device2') AND timestamp>'2017-01-01 00:00:00' AND timestamp<'2017-01-02 01:00:00' Interval('30s')"为例，该查询语句表示从设备编号为 device1 和 device2 中的 cpu 表中提取时间范围[2017-01-01 00:00:00，2017-01-02 00:00:00]内，每间隔 30s 的 max(usage)和 max(usage_system)的聚合结果。在时序数据库中，此类查询极为常见，由于原始数据打点粒度小但聚合窗口大，导致聚合查询效率低下。因此，针对研究此类查询的优化具有极其重要的意义。

数据采集模块采用异步策略，当用户发出 SQL 查询后，数据库立即响应并返回结果，然后对该 SQL 数据进行缓存，后台任务读取缓存信息并异步将相应的信息存储到表中。

(2)预计算元信息：预计算元信息提供了对预计算使用情况的全面视图，包括预计算的使用次数、最近使用时间、资源消耗情况等基本信息。使用次数能够反映预计算的活跃度和用户依赖程度；最近使用时间则有助于识别那些可能已经过时或不再符合当前查询需求的预计算；资源消耗情况，如 CPU 时间、内存使用量和 I/O 操作，能够揭示预计算对系统资源的实际影响，为资源管理和性能调优提供依据。

总而言之，数据采集在预计算推荐系统发挥着重要作用，数据采集的深度和广度直接影响着预计算推荐系统的性能和效果，数据采集得越深(包括详细的查询特征、系统资源使用情况)和越广(包括多种业务场景下的各类查询模式)能够为后续的负载分析和推荐算法提供更丰富的特征信息，从而做出更合理的推荐决策。

3.2　负载分析模块

负载分析模块是预计算推荐系统中的中间环节,它通过对采集的数据进行深入分析,挖掘用户使用数据库的规律性,为预计算推荐算法提供关键支持,如图 2 所示。

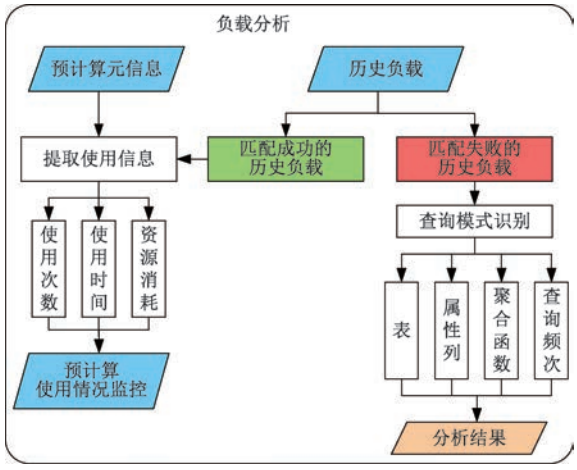

图 2　负载分析

负载分析主要分析两大类信息:(1)对于成功匹配到预计算的历史负载,分析的重点在于评估预计算的使用效果,包括使用次数、使用时间和资源消耗情况。通过这种分析,可以识别出哪些预计算是高效的,哪些可能需要进一步的调整或优化。此外,对成功匹配的负载进行分析,还可以帮助用户更好地理解预计算的实际效益,从而做出更加合理的使用决策,这些分析的结果用于支持未来的预计算监控设计。

(2)对于匹配失败的预计算的历史负载,分析的重点则更加注重于识别和理解查询模式,包括涉及的表、列、聚合函数和查询频次等关键信息,这些信息被转化为相应的数据结构,为预计算推荐算法提供了丰富的输入,预计算推荐算法可以推荐新的预计算来覆盖这些未被匹配的查询,实现查询效率的提升。

总而言之,负载分析是预计算推荐系统中不可或缺的一环,通过对采集数据的全面分析,负载分析不仅为预计算推荐算法提供了坚实的数据基础,还为用户提供了宝贵的洞察,帮助他们更好地理解和利用预计算,从而实现数据库查询性能的持续优化和提升。

3.3　预计算推荐模块

预计算推荐模块是预计算推荐系统中最重要的部分,旨在通过预计算推荐算法自动生成最优的预计算集合,从而显著提升时序数据库聚合查询的性能。预计算推荐算法与预计算推荐系统框架解耦,可以轻易改进或切换算法引擎,本文采用基于收益的预计算推荐算法,未来可以设计更为有效的算法。

如图 3 所示,基于收益的预计算推荐算法可以分为三个部分,分别是:(1)预计算候选生成;(2)预计算评估;(3)预计算选择。预计算候选生成基于负载分析结果,合并具有共同特征的查询以生成预计算候选集。预计算评估对预计算候选的成本和收益进行计算,成本是指预计算本身占用的存储空间;收益则通过预计算对查询效率的提升程度来衡量,具体以查询时间的减少作为量化依据。预计算成本和收益计算旨在评估预计算候选的质量,为后续的预计算选择提供依据。预计算选择作为核心步骤,其目标是在存储空间约束条件下,从预计算候选中选取收益最高的预计算集合进行实现,从而最大化后续工作负载的查询效率提升。

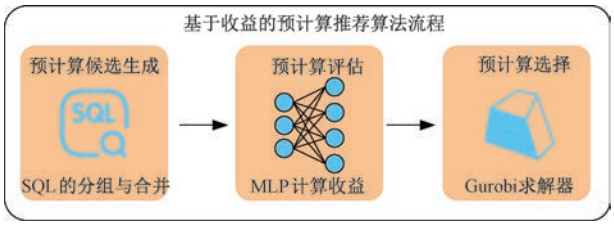

图 3　基于收益的预计算推荐算法流程

4　预计算推荐算法

第 3 节介绍了预计算推荐系统的整体架构,包括数据采集、负载分析和预计算推荐三大模块。本节介绍基于收益的预计算推荐算法的实现细节,包括预计算候选生成、预计算评估以及预计算选择三大部分。通过以上三个部分,预计算推荐算法可以自动生成相应的预计算结果,从而提高数据库的查询效率。

4.1　预计算候选生成

本节将详细阐述预计算候选生成规则的设计与实现,主要任务是根据负载分析的结果,生成相应的预计算候选集合。

通过 3.1 节的分析可知,本文针对的时序场景下工作负载的差异主要体现在以下四个方面:查询时间范围、聚合函数、聚合粒度以及设备数量与种类。本节基于设备数量和种类的区别将工作负载进一步划分为三大类别:单个设备工作负载、固定数量设备工作负载以及全部设备工作负载。接下来介绍每一类工作负载的预计算候选生成规则。

单个设备工作负载是指查询仅针对某一特定编号设备进行聚合计算，而不涉及多个设备的聚合操作。单个设备工作负载间的区别在于：设备编号的不同，时间范围的不同，聚合函数的不同，聚合粒度的不同。为了精准地识别出具有相似特征的工作负载以提高候选质量，本节设计了如图 4 所示的预计算候选生成规则，分为三大步骤：(1)设备分组：首先对负载分析结果按照设备编号进行分组，每组内的查询都是针对同一设备；(2)聚合粒度分组：在之前分组的基础上再次按照聚合粒度进行分组形成最终的分组，后续的处理都是针对组内进行；(3)聚合函数取并集：为了提高预计算候选的通用性，期望每条预计算候选包含尽可能多的聚合函数。因此采用组内聚合函数取并集的策略。

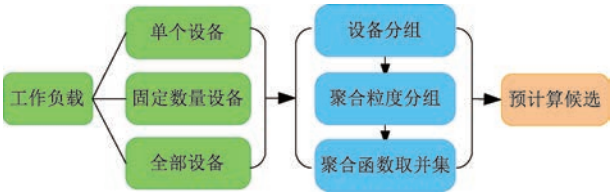

图 4 预计算候选生成规则

考虑到时序数据的价值通常随时间递减，新数据的价值远高于旧数据。若将预计算的时间范围限定为固定时间段，会导致预计算数据的价值逐渐降低。因此，预计算候选不设定固定的时间范围，而是作为一个定时任务，随着新数据的不断流入持续进行计算和存储，从而始终保持较高的数据价值。

固定数量设备工作负载与单个设备工作负载的区别在于，其查询对象并非单一设备，而是多个设备的组合。然而，在实际场景中，设备数量通常极为庞大，若考虑所有可能的设备组合，其计算复杂度过高将不可执行。通过观察发现，用户查询的设备组合往往具有一定的固定性和规律性，而非完全随机。因此，本文将第二类工作负载定义为固定数量设备工作负载，其特点是设备的数量固定(本文统一设置为 8)，差异主要体现在设备组合、时间范围、聚合函数以及聚合粒度四个方面。针对此类查询，本文在单个设备预计算候选生成规则的基础上进行改进，将设备分组依据从单一设备编号调整为设备编号组合，其余步骤保持不变。

全部设备工作负载是指针对所有设备进行聚合计算，其差异主要体现在时间范围、聚合函数以及聚合粒度三个方面。针对此类工作负载，本文在单个设备预计算候选生成规则的基础上进行调整，取消按照设备编号分组的步骤，其余步骤保持不变。

通过以上的预计算候选规则生成了最终的预计算结果，表 2 以 CPU-ONLY 场景为例展示了三种不同的预计算候选示例。

表 2 预计算候选示例

预计算候选类别	预计算候选示例
单个设备预计算候选	SELECT max(usage_user) FROM cpu Where device='device 1' Interval('60s')
固定数量设备预计算候选	SELECT max(usage_user), max(usage_system) FROM cpu Where device in ('device 2', 'device3') Interval('600s')
全部设备预计算候选	SELECT avg(usage_user) FROM cpu Interval('3600s')

4.2 预计算评估

上一节通过对历史工作负载的分析生成了预计算候选。然而，如何对这些预计算候选进行有效评估是一个关键问题。为此，本节引入成本和收益两个评估指标，用于衡量预计算候选的质量，从而为后续的预计算选择提供依据。本节将详细阐述预计算候选的成本和收益计算方法。

4.2.1 预计算成本计算

预计算成本计算的选择有很多种，包括：(1)预计算结果生成所耗费的时间；(2)预计算结果所占用的存储空间；(3)预计算结果生成所消耗的计算资源成本(比如 CPU、内存等)；(4)预计算维护与更新的时间等。不同的成本计算方法在不同的场景下起到的作用不同。其中，预计算存储空间的占用问题尤为突出，因为用户往往倾向于通过创建大量预计算来提升查询速度，这可能导致磁盘利用率显著下降。因此，为了简化并聚焦于最显著的存储空间因素，本文选择以预计算占用的存储空间大小作为成本评估的主要指标，而暂时忽略其他因素。

预计算本身是数据库中的一张表，以预计算 p_j 为例，该表的成本主要受以下三种因素影响：(1)原始表 T 的行数 N_T：原始表的行数直接影响预计算存储空间的占用。具体而言，原始表的行数越多，生成的预计算结果也越多，进而导致存储空间的占用显著增加。(2)p_j 的聚合粒度 I_j：预计算的聚合粒度越小，表示单位时间内的计算结果越多，存储空间占用越多。(3)p_j 聚合函数集合 g_j 的数量 $|g_j|$：聚合函数数量越多，存储空间占用越多。此外，c_t 为数据库中时间戳占用的存储空间。c_o 为其他列数据类型占用存储空间。预计算候选成本可通过公式

(1)进行计算:

$$c=\frac{N_T}{I_j}(c_o\times|g_j|+c_t),\tag{1}$$

其中,$(c_o\times|g_j|+c_t)$表示预计算结果中一行数据占用的存储空间大小,N_T/I_j 表示预计算结果中行数的大小,两者相乘即为预计算结果占用的存储空间大小(忽略了其他相关信息,包括预计算的元信息等)。

4.2.2　预计算收益计算

本节介绍收益的计算方式,以查询 q_i 和预计算 p_j 为例,收益的计算方式为 q_i 匹配 p_j 带来的查询时间提升。具体地,为直观反映预计算对查询性能的优化效果,将 q_i 运行两次:第一次未为 q_i 匹配 p_j,标记此次执行时间为 t_i;第二次为 q_i 匹配 p_j 后,重写后的查询标记为 $q_{i,j}$,其执行时间标记为 $t_{i,j}$;上述两次执行时间的差值即为收益 $B_{i,j}$,具体如公式(2)所示:

$$B_{i,j}=t_i-t_{i,j}\tag{2}$$

然而 $B_{i,j}$ 的计算太过于繁琐,需要频繁地执行查询和创建删除预计算,导致大量资源和时间上的浪费。因此本节构建一个深度学习模型来计算 $B_{i,j}$,下文将详细介绍这个方法的实现细节。

(1)特征处理

首先,对 q_i 和 p_j 进行特征提取。提取的特征包括聚合函数集合 g_i 和 g_j、聚合粒度 I_i 和 I_j、设备数量 b_i 和 b_j 以及查询的时间范围 s。表 3 展示了一个特征提取的示例。

表 3　特征提取示例

	SQL	特征提取
查询	SELECT max(usage_user) FROM cpu WHERE device = 'device 1' AND timestamp > '2017-01-01 00:00:00' AND timestamp < '2017-01-02 01:00:00' Interval('3600s')	g_i: [[max,usage_user]] I_i:3600 b_i:1 s:86400
预计算	SELECT max(usage_user), max(usage_system) FROM cpu WHERE device = 'device 1' Interval('60s');	g_j: [[max,usage_user], [max,usage_system]] I_j:60 b_j:1

在完成特征提取后,需对特征进行编码。对于聚合函数、设备数量等离散型数据,采用 One-Hot 向量编码方法进行转换,One-Hot 编码是一种将分类变量转换为数值型表示的方法,通过为每个类别创建一个独立的二进制向量来表示该类别。对于时间范围和聚合粒度等数值型数据,则使用 Z-Score 归一化方法进行标准化处理。Z-Score 归一化是一种常见的数据标准化方法,通过将每个数据点减去特征的平均值并除以标准差,将数据转换为均值为 0,标准差为 1 的分布。具体公式为 $Z=\sigma(X-\mu)$,其中,X 是原始数据点,μ 是特征的平均值,σ 是特征的标准差。这种标准化方法可以有效消除不同特征之间的量纲差异,使模型能够更公平地处理各个特征。

(2)模型介绍

对于聚合函数集合如何输入模型还存在两个问题。不同的查询和预计算的聚合函数集合数量不同,这将导致输入向量的长度不固定。并且集合中元素的顺序是任意的,如果直接输入模型将导致语义错误。例如(max(col1),sum(col2))和(sum(col2),max(col1))应当是相同的特征表示,而不受顺序的影响。

为了解决以上问题,本文的模型架构受到了 Deep Sets[31] 上的工作的启发,Deep Sets 是一个用于对集合进行操作的神经网络模块。Deep Sets 基于这样的观察:集合 S 上的任何函数 $f(S)$,如果对 S 中的元素是置换不变的,可以分解成 $\rho[\sum\varphi(x)]$ $(x\in S)$的形式,其中适当地选择了函数 ρ 和 φ。关于这个性质的更正式的讨论和证明,可以参考 Zaheer 等人[31]的文章。本文选择多层感知机(Multilayer Perceptron,MLP)来参数化函数 ρ 和 φ,并依靠它们的函数逼近性质来学习任意集合 S 的灵活映射 $f(S)$。最终,针对聚合函数集合,采用了如图 5 所示的 aggBlock 来表示其特征,该集合的最终表示由其元素的各个变换表示的平均值给出,选择一个平均值而不是简单地求和,以便于推广到集合中的不同数量的元素,否则输出的整体幅度将根据集合的元素数量而变化。

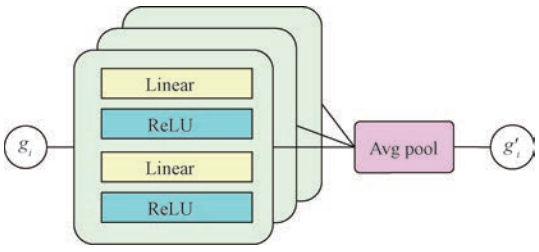

图 5　aggBlock 架构

选择 MLP 作为收益估计模型的核心,主要基于任务本质与实验证据的双重考量。预计算收益估

计是一个典型的连续值回归问题：给定查询-预计算对的特征（既包含无序的聚合函数集合，也包含连续的粒度与时间范围），要求输出精确的查询时间减少量。MLP 在理论上具备万能近似能力，能够以任意精度逼近这种连续映射；同时，通过与 Deep Sets 结构的级联，MLP 可以自然地处理无序集合特征，保持对聚合函数排列的置换不变性，从而避免了简单拼接带来的顺序敏感误差。

为进一步验证 MLP 的合理性，本文在同一训练/验证/测试集上补充了两种常用基线模型：线性回归（Linear Regression，LR）与梯度提升树（Gradient Boosting Decision Tree，GBDT）。三种模型的输入特征保持一致，均经过 Deep Sets 聚合后的集合表征向量与数值特征拼接而成。实验结果如表 4 所示。

表 4 收益预测对比实验

MAPE	Median	90th	95th	99th	Max	Mean
LR	15.54	40.67	42.36	151	261	37.32
GBDT	0.726	2.80	5.33	31.7	101	1.96
MLP	0.575	2.61	4.93	27.0	137	1.78

结果表明，收益预测任务属于非线性问题，所以线性回归模型 LR 表现不好。对于复杂非线性问题，神经网络能够更好地捕捉数据的复杂模式，效果最好。

最终的收益模型架构如图 6 所示，对于聚合函数集合 g 的处理，使用 aggBlock 进行特征表示。对于其他特征的处理，按照之前描述的方法进行特征表示。最终通过级联合并各个特征表示，并将其输入 MLP_{out}，以输出最终的收益值。

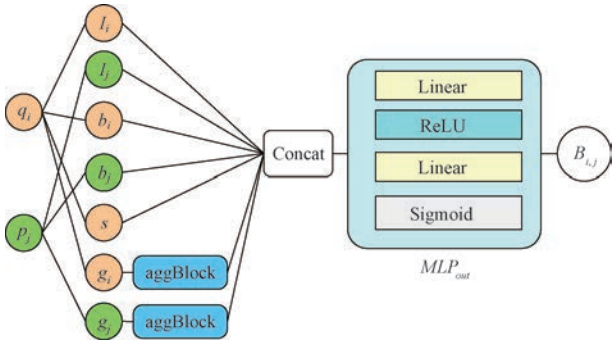

图 6 收益模型

预计算收益 $B_{i,j}$ 计算如公式(3)所示：

$$B_{i,j}=MLP_{out}([I_i,I_j,b_i,b_j,s,g'_i,g'_j]) \tag{3}$$

其中，g'表示查询和预计算的聚合函数集合 g 经过 $aggBlock$ 模块后的特征向量，$B_{i,j}$ 是由聚合粒度 I，设备数量 b，查询时间范围 s 和聚合函数 g'特征向量级联，并经过一个 MLP_{out} 模块计算得来。

所有 MLP 模块都是采用 $ReLU=\max(0,x)$激活函数的多层全连接神经网络。隐藏层使用 $ReLU$ 激活函数，因为其在实践中表现出较高的计算效率。最后一层使用 $Sigmoid=1/(1+e^{-x})$激活函数，并且输出标量值，因此 $B_{i,j}\in[0,1]$。所有其他表示向量 g'和 MLP 的隐藏层激活被选择为维度 d 的向量，其中 d 是一个超参数，通过网格搜索在一个单独的验证集上进行优化。

对 $B_{i,j}$ 进行归一化处理的具体方法如下：首先对目标值取对数以使其分布更均匀；随后利用从训练集获得的最小值和最大值归一化到区间$[0,1]$。由于归一化是可逆的，因此可以从模型的预测 $B_{i,j}\in[0,1]$中恢复未归一化的收益。

(3)模型训练

训练数据由一批工作负载 Q 组成。利用第 4.1 节中的预计算候选生成规则可以得到预计算候选集合 PC。此外，可以添加一些手动创建的预计算来模拟真实环境中用户可能创建提供自己使用的预计算场景。首先，判断 $p_j\in PC$ 是否可以用来匹配 $q_i\in Q$，从而为每个查询 q_i 选择可匹配的预计算集合 P_i。对于 q_i 和相应的 P_i，采样对可用$\{(q_i,p_j)\}$表示，其中 $p_j\in P_i$，q_i 可以匹配 p_j，重写后的查询是 $q_{i,j}$。为了获得真实标签值，将查询在数据库中运行，并从真实的运行日志获得数据。

训练集、验证集和测试集的划分为 7∶2∶1。训练使用平均绝对百分比误差（Mean Absolute Percentage Error，MAPE）度量作为损失函数。MAPE 计算方式如公式(4)所示，作用是衡量预测收益值 $B_{i,j}$ 与真实收益值 $\widehat{B_{i,j}}$ 之间的差异，其中 n 为样本数量。

$$MAPE=\frac{1}{n}\sum_{i=1}^{n}\left|\frac{\widehat{B_{i,j}}-B_{i,j}}{B_{i,j}}\right| \tag{4}$$

训练过程使用 Adam[32]（$\beta_1=0.9,\beta_2=0.999$）优化器进行训练。批大小设置为 512；训练 40 epoch，采用早停（验证集 MAPE 连续 5 epoch 无改善即停止）。

4.3 预计算选择

预计算选择的目标是从预计算候选中，在存储空间上限（即预计算总成本）的约束下，选择收益最高的预计算集合进行推荐，从而实现工作负载查询效率的最大化。本节将预计算选择问题建模为整数规划问题，并对其进行求解。

4.3.1 问题建模

首先需要明确目标函数，对于用户而言，目标是

最大化工作负载的收益。此外需要注意的是,仅当查询与预计算匹配时才会产生收益。因此,最终目标函数如公式(5)所示:

$$\underset{e_{i,j}}{\operatorname{argmax}}\sum_{i=1}^{n}\sum_{j=1}^{k}B_{i,j}e_{i,j} \tag{5}$$

在公式(5)中,$e_{i,j}$ 是一个二元变量,表示查询 q_i 和预计算 p_j 是否匹配。如果匹配,则 $e_{i,j}=1$,否则 $e_{i,j}=0$。$B_{i,j}$ 表示 q_i 和 p_j 匹配时的收益。本节的目标是通过优化 $e_{i,j}$ 的值,使得所有匹配对的收益总和最大化。

其次需要确定约束条件,在选择预计算过程中有许多约束条件,包括:

(1)存储空间约束:用户会输入一个存储空间的上限 τ,而每个预计算候选 p_j 也有其对应的成本 c_j,最终选择的预计算集合的总成本不得超过存储空间上限。因此,存储空间约束的定义如公式(6)和(7)所示:

$$\sum_{j=1}^{k}x_jc_j\leqslant\tau \tag{6}$$

$$x_j=\max\{e_{i,j}\mid i\in[1,n]\} \tag{7}$$

在公式(6)中,$\sum_{j=1}^{k}x_jc_j$ 表示所有被选择的预计算的总成本,这个总和必须小于或等于用户提供的存储空间上限 τ。在公式(7)中,x_j 的值由 $e_{i,j}$ 决定,即如果至少有一个查询 q_i 与预计算 p_j 匹配($e_{i,j}=1$),则 $x_j=1$,表示该预计算被选择;否则 $x_j=0$,表示未被选择。这些约束确保了在满足存储空间限制的前提下,选择最优的预计算集合。

(2)预计算匹配限制:受现有预计算匹配机制的限制,每条查询 q_i 只能匹配一条预计算 p_j,不能同时匹配多条,所以预计算匹配限制定义如公式(8)所示。

$$\sum_{j=1}^{k}e_{i,j}=1,e_{i,j}\in\{0,1\} \tag{8}$$

在公式(8)中,$e_{i,1}+e_{i,2}+\cdots+e_{i,k}=1$ 确保了每条查询 q_i 只能与一个预计算 p_j 匹配,从而满足预计算匹配机制的限制条件。

基于以上的讨论,预计算选择问题完整模型的如公式(9)所示。

$$\begin{aligned}&\underset{e_{ij}}{\operatorname{argmax}}\sum_{i=1}^{n}\sum_{j=1}^{k}B_{ij}e_{ij}\\&\text{s.t.}\quad\sum_{j=1}^{k}x_jc_j\leqslant\tau,\\&x_j=\max\{e_{ij}\mid i\in[1,n]\},\\&\sum_{j=1}^{k}e_{ij}=1,e_{ij}\in\{0,1\}\end{aligned} \tag{9}$$

4.3.2　问题求解

对于 4.3.1 节的整数线性规划问题,本文选择使用商业的求解器来进行求解。最著名的商业求解器是 Gurobi[33]。Gurobi 是由美国 Gurobi Optimization 公司开发的新一代大规模优化器。在理论和实践中,Gurobi 优化工具都被证明是全球性能领先的大规模优化器,具有突出的性价比。

4.4　复杂度分析

对于本章中提出的算法,由于 4.1 节中仅涉及设备分组情况的讨论,4.2 节则是对评估指标的介绍,因此我们着重对 4.3 节中提出的算法进行了复杂度分析。同时,由于 4.3 节中提出的算法中,存储成本是预先设定好的阈值,因此只对其进行时间复杂度分析。

针对 4.3.1 节中提出的算法,在最坏的情况下,公式(5)中的二元变量 $e_{i,j}$($1\leqslant i\leqslant n$,$1\leqslant j\leqslant k$),需要遍历 2^{nk} 种情况后才能确定最大收益对应的预计算匹配情况。对于存储空间约束,约束条件公式(6)、公式(7)的时间复杂度为 $O(k)$;对于预计算匹配限制,约束条件公式(8)的时间复杂度为 $O(nk)$。因此,该算法的时间复杂度为 $O(2^{nk})$。

4.3.2 节中提到的商业求解器 Gurobi 使用分支定界法来求解整数线性规划问题,并在该框架下动态加入割平面进行优化。但是在最坏情况下,分支定界法的时间复杂度也为 $O(2^{nk})$[34-36]。

综上,本文提出的算法在最坏情况下的时间复杂度为 $O(2^{nk})$。虽然最坏情况下的时间复杂度为指数级,但 Gurobi 的剪枝、割平面生成和启发式技巧往往能将实际搜索空间大幅压缩,使得中小规模问题在可接受时间内(通常为秒到分钟级)得到求解。

5　实验分析

本节以大型国产数据库系统 KaiwuDB[12] 搭建实验环境,以验证本文研究内容的有效性。

本节对预计算推荐系统进行了实验验证,主要从以下两个方面展开:首先,评估了预计算推荐系统在查询性能提升方面的效果;其次,将预计算推荐算法与其他推荐算法进行了对比分析;通过以上两方面的实验,验证了预计算推荐系统的有效性。

5.1　实验设置

5.1.1　环境准备

本节将详细介绍实验与系统搭建所需的环境,所设计的解决方案最终在浪潮公司研发的数据库

KaiwuDB 上成功实现。本文使用的实验环境为单节点的 KaiwuDB 服务器，具体硬件环境如表 5 所示。实验中用到的软件环境如表 6 所示。

表 5 硬件环境

机器类别	内存	处理器	硬盘
KaiwuDB 数据库	64G	Intel(R)Core (TM)i5-12600K CPU	U.2 Intel P4510 2T

表 6 软件环境

软件	版本
Python	Python3.6
Golang	Golang1.14
Pycharm	Pycharm2022.3
Goland	Goland2022.3
操作系统	Ubuntu 20.04 focal
数据库	KaiwuDB1.2
Apache JMeter	Apache JMeter 5.4.1
Docker	Docker20.10.2

5.1.2 数据集

TSBS 是当前认可度最高、使用范围最广泛的时序数据库评测基准，从 2018 年开始，已有多个开源时序数据库厂商将 TSBS 作为基准测试平台，该评测基准具有可扩展性，可对多种时序数据库的读写性能进行基准测试，支持多种类别的典型查询，并能自动汇总测试结果。当前 TSBS 支持 2 种场景：IoT 和 CPU-ONLY：

(1)IoT 场景：该场景模拟了物联网设备生成的时间序列数据，主要用于评估数据库在处理大量设备数据时的性能。物联网设备通常会产生高频、多样化的数据，例如传感器读数、设备状态信息等。

(2)CPU-ONLY 场景：CPU-ONLY 场景模拟了从服务器或虚拟机中收集的 CPU 使用率数据。它生成的数据模型相对简单，主要围绕 CPU 使用率这一指标展开，同时包含一些标签来标识数据的来源，例如主机名、数据中心、操作系统等。

使用 TSBS 进行基准测试包括 3 个过程：(1)数据与查询生成。使用 TSBS 进行基准测试，需要预先生成测试数据和查询用例，以避免因即时生成数据和查询用例对测试结果产生影响。(2)数据加载与写入。该过程将加载先前预先生成的模拟数据，写入待测试数据库，同时也支持即时模拟和加载数据。(3)查询执行。该过程用于评估待测数据库的查询性能，需要先加载预先生成的模拟数据和查询用例，可以通过参数设置并行查询线程数量。最后将查询用例的执行结果输出。

本实验中的数据集统一采用 TSBS 中 CPU-ONLY 场景数据集。这是因为 CPU-ONLY 场景是一种简化版的 IoT 的场景，相比于 TSBS 支持的另一种 IoT 场景，没有数据缺失、空数据和乱序数据的影响。具体参数如表 7 所示。实验设置了两种不同的场景，通过两种场景模拟时序数据库的不同使用场景。两种场景的区别主要在于数据集所包含的设备数量和持续时间的不同，数据时间间隔均维持在 10 秒。在场景 2 中，由于设备数量相对较多，所以数据集仅覆盖了 4 天的时间跨度。

表 7 数据集参数

参数	场景 1	场景 2
设备数量	100	4000
持续时间	31 天	4 天
单个设备记录数	2678400	34560
数据间隔	10 秒	
指标	10	

实验中的查询基于 TSBS 中 CPU-ONLY 场景自带的查询模板。然而，由于该模板太过于简单，本文对其进行了如下的扩展和补充，具体参数如表 8 所示。

表 8 查询数据集参数

参数	场景 1 参数值	场景 2 参数值
聚合粒度(1min 的倍数)	[1min—12h]	[1min—4h]
查询时间范围(1h 的倍数)	[1h—12h]	[1h—4h]
设备数量	[1,8,100]	[1,8,4000]
聚合函数	max,min,sum,avg,count	
生成条数	180	

5.1.3 模型离线训练

整个系统可分为两个主要部分：在线预计算推荐部分和预计算收益模型离线训练部分。模型的训练是在离线训练阶段完成的，而预计算的推荐则是在在线推荐阶段进行的。因此在实验进行之前，需要对预计算收益模型完成训练。

查询数据集按照表 8 进行生成，并在数据库中进行运行以获得真实的标签值。具体模型的训练过程在 4.2.2 节中已经有详细的描述。

5.2 查询效果实验

本节将评估预计算推荐系统在查询性能提升方面的效果，重点探讨以下问题：(1)是否能够根据历史负载有效推荐预计算，并显著降低工作负载的查询时间。(2)成本上限对于查询效果会产生何种影响。针对以上两个问题，本节设计了实验并进行了验证。

如表 8 所示，生成的数据集中共 180 条查询。

实验将该查询数据集划分为两部分:随机选择其中90条作为历史负载用于生成预计算;剩下90条作为工作负载用于实际性能测试,且为方便记录,我们给每条工作负载设置一个1～90范围内的编号。

首先将历史负载在数据库中执行并在数据采集模块进行记录保存,这将用于后续的预计算推荐。接着工作负载被执行两次:第一次在预计算推荐前运行,第二次在预计算推荐后运行,分别记录两次执行的查询时间,以对比分析预计算对查询效率的提升效果。

实验结果如图7所示,横轴表示查询(即相应工作负载)的编号,纵轴表示查询的时间。黑色柱状图表示查询不使用预计算的查询时间,红色柱状图表示查询使用预计算的查询时间。可以观察到,在进行预计算推荐后,两种场景下工作负载的查询时间均显著降低。这一结果证明了预计算推荐系统的有效性。

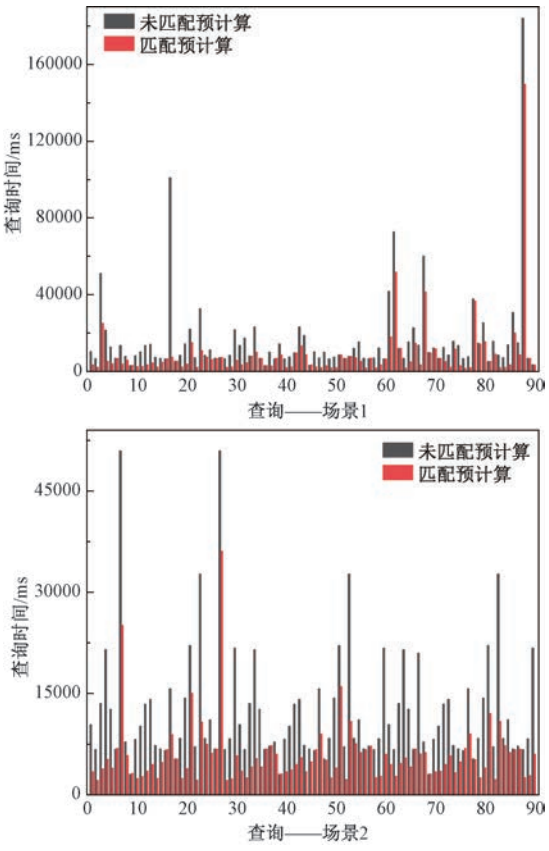

图7　查询提升效果

如图8所示,本实验探讨了成本上限对于查询时间的影响。横轴表示用户定义的成本上限,纵轴表示测试负载在使用预计算时的查询收益,即未进行预计算推荐时的查询时间与进行预计算推荐时的查询时间之差。

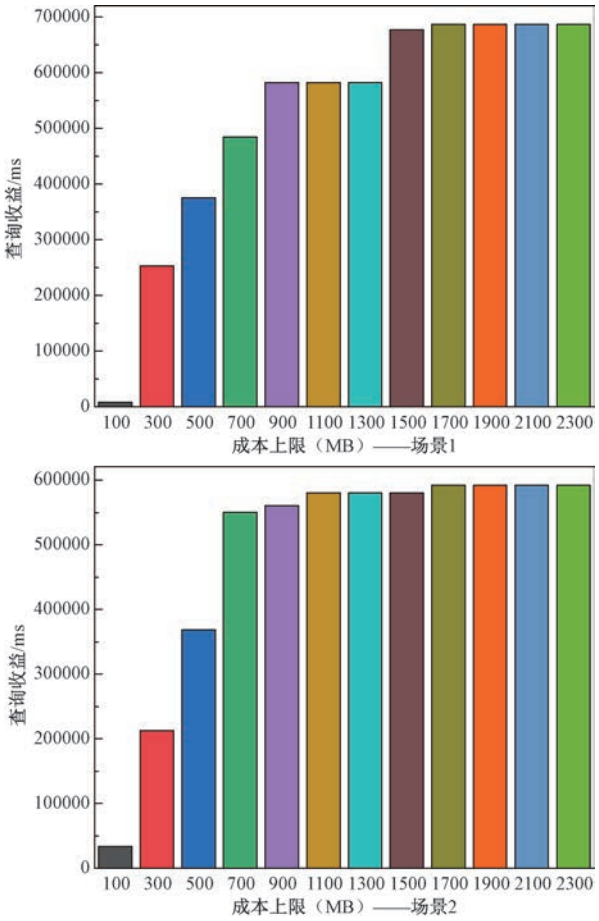

图8　存储空间上限对查询收益的影响

实验结果表明,在两种场景下,随着成本上限的增加,查询收益呈现上升趋势。原因是当成本上限逐渐增加时,预计算推荐系统推荐结果数量和质量提升,生成加速效果更好的预计算,使部分查询从预计算获得的收益增加,因此总收益不断增加。同时也注意到,当成本上限到达一定值后,预计算带来的查询收益不再增加。这表明在达到一定的空间资源开销后,预计算推荐的结果已经能够覆盖大多数高收益的查询场景,进一步增加成本上限仅会引入边际收益较低的预计算,从而导致整体收益曲线趋于平稳。可见成本上限是影响预计算推荐效果的关键因素,在实际应用中要综合考虑空间大小和预期收益大小,从而设置合理的成本上限进行预计算的推荐。

5.3　推荐算法对比实验

本节将本文提出的预计算推荐算法与其他推荐算法进行对比实验,验证所提出算法的优越性。

(1)对比算法

①TopkFreq:选择历史负载中出现频率前k高的查询语句作为预计算进行推荐。

②TopkOver:选择历史负载中成本前 k 小的查询语句作为预计算进行推荐。

③TopkBenefit:选择历史负载中收益前 k 高的查询语句作为预计算进行推荐。

④RuleBased:基于规则的预计算推荐算法:对所有历史负载按照聚合粒度分组,有倍数关系的组合并到小的组(比如 60s 和 600s 的聚合粒度会合并到 60s),组内聚合函数取并集,进行推荐。如果用户不设置成本上限,默认采用这种推荐算法。

⑤BenefitBased:基于收益的预计算推荐算法,即本文所采用的方法。

贪心推荐算法(TopkFreq、TopkOver 和 TopkBenefit)借鉴了论文[37]中物化视图自动生成领域的对比方法,并对其进行了改进与实现,以适应时序数据库中预计算的需求。RuleBased 为 KaiwuDB 中已有的基于规则的预计算推荐算法。

(2)指标

评估预计算推荐算法的指标是查询收益,计算方式如公式(2)所示。

(3)实验结果

实验将查询数据集划分为历史负载和工作负载,通过设置不同的成本上限大小,同时运行本文提出的预计算推荐算法与对比算法,在两种场景中记录每种算法的查询收益,具体的实验结果如图 9 所示,其中横轴为设置的成本上限,纵轴为查询收益。图中不同颜色的曲线代表如上所述的对比算法。实验结果表明,BenefitBased、RuleBased 相较于各种贪心算法能够带来的查询收益上限是更高的,原因是各种贪心算法只是从工作负载本身考虑,而没有考虑工作负载之间的合并。相比之下,BenefitBased 和 RuleBased 两种算法通过对负载的深入分析,能够识别出频繁查询的结果,并将相关的工作负载进行合并作为预计算的候选,从而达到了更优的效果。此外,尽管 BenefitBased 和 RuleBased 能够带来的查询收益上限近似,但 BenefitBased 具备更高的灵活性,能够选择在当前成本上限的前提下最大收益的预计算集合,RuleBased 未能实现这一点,它只适用于存储空间充足的情况。RuleBased 算法存在巨大变动的原因是它只有在存储空间充足时才会生效,否则没有推荐结果。

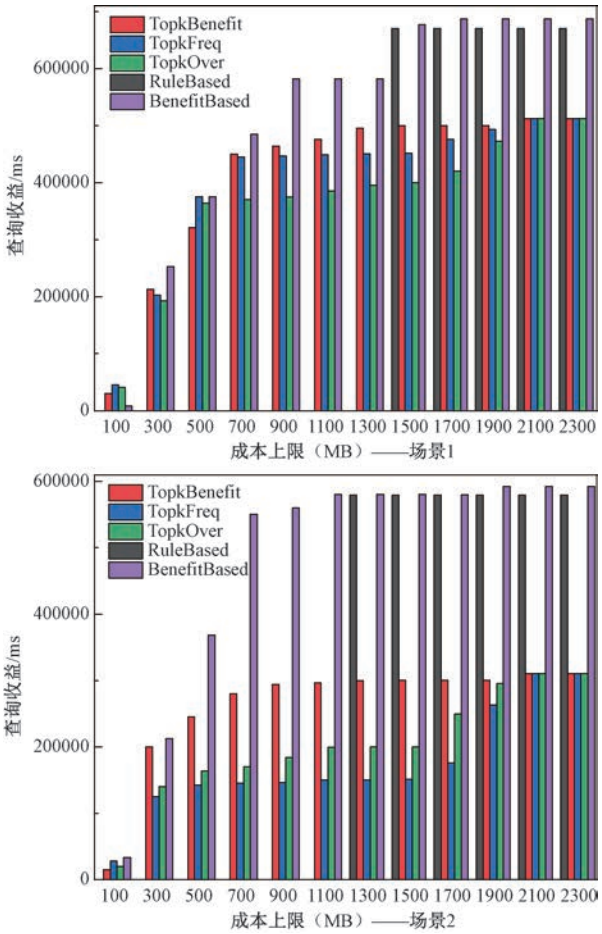

图 9 预计算推荐算法对比实验

6 总 结

本文设计并介绍了一种面向时序数据库的预计算推荐系统,旨在解决当前时序数据库中预计算的创建与配置过程复杂,严重依赖背景知识丰富的数据库管理员手动创建的问题。本文首先提出了预计算推荐系统的整体架构,包括数据采集、负载分析和预计算推荐三个模块。随后,对预计算推荐算法的三个部分做了详细说明,包括预计算候选生成、预计算评估、预计算选择。在预计算候选生成部分,基于对工作负载的分类,设计了预计算候选生成规则。接着,介绍了预计算候选成本和收益的计算方法,并构建了一个深度学习模型来计算收益。最后,将预计算选择问题建模为整数规划问题并通过求解器进行求解,以在不超过存储空间上限的条件下选择收益最高的预计算集合。实验结果表明,本文提出的预计算推荐系统能够根据历史负载有效推荐出预计算,显著降低了工作负载的查询时间。与其他推荐算法相比,本文提出的预计算推荐算法不仅能够灵

活地设置成本上限,还能在相同条件下实现更高的收益。

参 考 文 献

[1] Sandeep M, Chandavarkar B R. Data processing in IoT, sensor to cloud: Survey//Proceedings of the 2021 12th International Conference on Computing Communication and Networking Technologies. Kharagpur, India, 2021: 1-7

[2] Tobias Oetiker, About RRDtool, http://oss.oetiker.ch/rrdtool 2017, 2, 20

[3] Redgate Software, DB-Engines-Knowledge Base of Relational and NoSQL Database Management Systems, https://db-engines.com/en/ 2024, 6, 18

[4] Liu Shuai, Qiao Ying, Luo Xiong-Fei, et al. Key technologies of time series database: a survey. Journal of Computer Research and Development, 2024, 61(03): 614-638 (in Chinese)
(刘帅, 乔颖, 罗雄飞, 等. 时序数据库关键技术综述. 计算机研究与发展, 2024, 61(03): 614-638)

[5] Bian H, Yan Y, Tao W, et al. Wide table layout optimization based on column ordering and duplication//Proceedings of the 2017 Association for Computing Machinery International Conference on Management of Data. Houston, USA, 2017: 299-314

[6] Pelkonen T, Franklin S, Teller J, et al. Gorilla: A fast, scalable, in-memory time series database. Proceedings of the VLDB Endowment, 2015, 8(12): 1816-1827

[7] Travis CI Status, OpenTSDB-A Distributed, Scalable Monitoring System, https://opentsdb.net/ 2021, 9, 2

[8] InfluxDB Time Series Data Platform | InfluxData, https://www.influxdata.com/ 2023, 5, 20

[9] TDengineDB, TDengine | Time-Series Database for Industrial IoT, https://www.tdengine.com/ 2021, 2, 16

[10] PostgreSQL Development Team, BRIN Indexes, https://www.postgresql.org/docs/current/brin.html 2025, 2, 20

[11] Gray J, Chaudhuri S, Bosworth A, et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data mining and knowledge discovery, 1997, 1: 29-53

[12] Lang Chao. KaiwuDB distributed multimodal database—one database for multiple modes, faster than others. Inspur KaiwuDB, 2025-01-08 (in Chinese)
(浪潮. 浪潮 KaiwuDB 分布式多模数据库—多模一库,快人一步. 浪潮 KaiwuDB, 2025-01-08)

[13] Agrawal S, Chaudhuri S, Narasayya V R. Automated selection of materialized views and indexes in SQL databases//Proceedings of the 26th International Conference on Very Large Data Bases. Cairo, Egypt, 2000: 496-505

[14] Camacho-Rodríguez J, Colazzo D, Herschel M, et al. Reuse-based optimization for pig latin//Proceedings of the 25th Association for Computing Machinery International Conference on Information and Knowledge Management. Indianapolis, USA, 2016: 2215-2220

[15] Jindal A, Karanasos K, Rao S, et al. Selecting subexpressions to materialize at datacenter scale. Proceedings of the VLDB Endowment, 2018, 11(7): 800-812

[16] Jindal A, Qiao S, Patel H, et al. Computation reuse in analytics job service at microsoft//Proceedings of the 2018 Association for Computing Machinery SIGMOD/PODS Conference. Houston, USA, 2018: 191-203

[17] Perez L L, Jermaine C M. History-aware query optimization with materialized intermediate views//Proceedings of the 2014 IEEE 30th International Conference on Data Engineering. Chicago, USA, 2014: 520-531

[18] Dokeroglu T, Bayir M A, Cosar A. Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries. Applied Soft Computing, 2015, 30: 72-82

[19] Bello R G, Dias K, Downing A, et al. Materialized views in Oracle//Proceedings of the 24th International Conference on Very Large Data Bases. Edinburgh, UK, 1998: 24-27

[20] Baralis E, Paraboschi S, Teniente E. Materialized view selection in a multidimensional database//Proceedings of the 23rd International Conference on Very Large Data Bases. Athens, Greece, 1997: 156-165

[21] Horng J T, Chang Y J, Liu B J. Applying evolutionary algorithms to materialized view selection in a data warehouse. Soft Computing, 2003, 7: 574-581

[22] Kipf A, Kipf T, Radke B, et al. Learned cardinalities: Estimating correlated joins with deep learning. arXiv preprint arXiv:1809.00677, 2018

[23] Kumar A, Kumar T V V. Materialized view selection using self-adaptive perturbation operator-based particle swarm optimization. International Journal of Applied Evolutionary Computation (IJAEC), 2020, 11(3): 50-67

[24] Marcus R, Negi P, Mao H, et al. Neo: A learned query optimizer. arXiv preprint arXiv:1904.03711, 2019

[25] Marcus R, Papaemmanouil O. Plan-structured deep neural network models for query performance prediction. arXiv preprint arXiv:1902.00132, 2019

[26] Ortiz J, Balazinska M, Gehrke J, et al. Learning state representations for query optimization with deep reinforcement learning//Proceedings of the Second Workshop on Data Management for End-to-End Machine Learning. Houston, USA, 2018: 1-4

[27] Sohrabi M K, Ghods V. Materialized view selection for a data warehouse using frequent itemset mining. Journal of Scientific Computing, 2016, 11(2): 140-148

[28] Gosain A, Sachdeva K. Handling constraints using penalty functions in materialized view selection. International Journal of Natural Computing Research (IJNCR), 2019, 8(2): 1-17

[29] Han Y, Li G, Yuan H, et al. An autonomous materialized view management system with deep reinforcement learning//Proceedings of the 2021 IEEE 37th International Conference on Data Engineering. Helsinki, Finland, 2021: 2159-2164

[30] timescale, Time Series Benchmark Suite, a tool for comparing and evaluating databases for time series data, https://github.com/timescale/tsbs 2023, 5, 20

[31] Zaheer M, Kottur S, Ravanbakhsh S, et al. Deep sets. Advances in Neural Information Processing Systems, 2017, 30

[32] Kingma D P, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014

[33] Gurobi Optimization, The leader in decision intelligence technology-gurobi optimization, https://www.gurobi.com/ 2023, 5, 20

[34] Christos H. Papadimitriou. On the complexity of integer programming. Association for Computing Machinery, 1981, 10: 765-768

[35] Xu Rui, Gu Shou-Zhen, Sha Xing-Mian, Zhuge Qing-Feng, Shi Liang, Gao Si-Yuan. Optimization study for multi-read/write head magnetic domain wall memory. Journal of Software, 2020, 31(9): 2723-2740 (in Chinese)
(许瑞, 谷守珍, 沙行勉, 诸葛晴凤, 石亮, 高思远. 面向多读/写头磁畴壁存储器的优化研究. 软件学报, 2020, 31(9): 2723-2740)

[36] Gurobi Optimization, MILP worst-case Complexity order. https://support.gurobi.com/hc/en-us/community/posts/8729463179409-MILP-worst-case-Complexity-order/2023, 8, 7

[37] Han Y, Li G, Yuan H, et al. AutoView: An autonomous materialized view management system with encoder-reducer. IEEE Transactions on Knowledge and Data Engineering, 2022, 35(6): 5626-5639

XIA Xiao-Fang, Ph. D., associate professor, Ph. D. supervisor. Her research interests focus on internet of things, database, data security, cloud-edge collaboration.

SUN Lu-Ming, Ph. D., His research interests focus on database and machine learning.

MU Chang-Chun, M. S candidate. His research interests focus on query optimization of time series databases.

LI Dong-Yang, M. S candidate. His research interests focus on database query optimization.

LI Hui, Ph. D., professor, Ph. D. supervisor. His research interests focus on data mining, privacy-preserving data query and retrieval.

CUI Jiang-Tao, Ph. D., professor, Ph. D. supervisor. His research interests focus on database management and kernel technology, data engineering.

LIANG Wei Ph. D., Ph. D. supervisor. Her research interests focus on industrial wireless sensor networks, multi-sensor data fusion, et al.

Background

This research addresses a critical challenge in the field of time-series database management, specifically focusing on optimizing aggregation queries through precomputation techniques. Time-series databases are widely used in industrial IoT, financial monitoring, and other domains where efficient data analysis is essential. However, as data volumes grow exponentially, the performance of aggregation queries such as sums, averages, and maximum/minimum calculations becomes a bottleneck. While precomputation can significantly improve query performance by precalculating and storing results, its implementation remains complex and heavily reliant on database administrators, limiting accessibility for non-expert users.

Internationally, existing solutions like OpenTSDB, InfluxDB, and TDengine support precomputation but require manual configuration, which is time-consuming and error-prone. Recent advancements in materialized view automation for relational databases offer inspiration, yet these methods are not directly applicable to time-series workloads due to differences in data patterns and query structures.

This paper proposes an innovative precomputation recommendation system for time-series databases, leveraging deep learning and integer programming to automate the selection of optimal precomputation sets. Our system analyzes historical workloads, estimates the benefits of precomputation candidates, and selects the most cost-effective solutions under storage constraints. Experimental results demonstrate significant reductions in query latency, outperforming traditional rule-based and greedy algorithms.

This work is part of a larger project aimed at enhancing the performance and usability of KaiwuDB, a leading domestic database system developed to bridge the gap with international counterparts. Our study builds on prior achievements in query optimization and extends them to the unique demands of time-series data. The outcomes contribute to the broader goal of making advanced database technologies more accessible and efficient for real-world applications.