arXiv:2304.12747v1 [cs.DB] 25 Apr 2023

基于深度学习的数据库管理系统自动调优

KarthickGunasekaran Kajal Tiwari信息与计算机科学学院 信息与计算机科学学院 马萨诸塞大学阿默斯特分校 马萨诸塞大学阿默斯特分校

Rachana Acharya信息与计算机科学学院
马萨诸塞大学阿默斯特分校

摘要—**数据库系统配置管理是一项艰巨的任务，因为有数百个配置参数控制着系统的各个方面。由于这些参数并未标准化，也彼此不独立或通用，因此很难确定最优设置，这使问题更加复杂。为解决这一问题，我们实现了一种自动化方法，利用监督式和无监督式机器学习方法选择影响显著的参数、映射未见过的工作负载并推荐参数设置。该方法在一个名为 OtterTune 的新工具中实现，目前正在三个数据库管理系统上进行评估。结果表明，它推荐的配置与现有工具或人工专家生成的配置一样好，甚至更好。在这项工作中，我们扩展了一种基于 OtterTune [1] 的自动化技术，利用监督式和无监督式机器学习方法，复用以往会话中收集的训练数据，为新的数据库管理系统部署进行调优，从而提升延迟预测能力。我们的方法扩展了原论文提出的技术。我们使用高斯混合模型聚类来筛选指标，并将随机森林等集成模型与神经网络等非线性模型结合起来进行预测建模。**

I. 引言

A. 问题陈述

我们的主要目标是利用从先前调优中收集的数据训练机器学习（ML）模型，并将这些模型主要用于：（1）删减冗余指标；（2）将未见的数据库工作负载映射到先前的工作负载，以便迁移经验；以及（3）通过工作负载映射改进延迟预测。我们通过以下步骤实现了这一目标：

- 删减冗余指标：由于我们拥有数百个内部和外部数据库管理系统特定指标，因此将搜索空间限制为一组较小的指标，而这些指标会显著影响延迟。此步骤加快了整个过程，并确保模型能够装入内存。
- 工作负载映射： 工作负载映射是根据性能测量结果，将目标 DBMS 的工作负载与其存储库中最相似的工作负载进行匹配。具体做法是借助欧几里得距离寻找最近邻，并计算每个工作负载在所有指标上的距离平均值，作为其“得分”。重用经验可以节省时间和资源，并且在当前工作负载的观测值较少时，也有助于模型做出有依据的决策。
- 延迟预测： 最终目标是通过工作负载映射改进延迟预测。预测建模将通过回归模型（高斯过程回归）完成，但我们也会尝试使用神经网络。

II. 方法

整个系统的概览如下面的图1所示。该过程包含两个重要步骤。第一步是剪除重要指标，接下来通过工作负载映射进行自动调优。

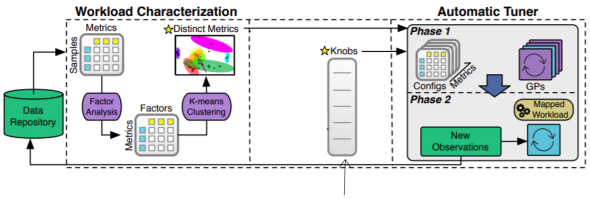


图1. 架构概览

A. 指标剪枝

在此阶段，剪除了冗余指标。为了捕捉系统性能的变异性并区分不同的工作负载，本文选取少量变异性较高的指标。系统需要捕捉少数指标，以便涵盖工作负载的所有区别性特征，同时缩短机器学习算法的整体运行时间。剪枝步骤非常重要，因为它有助于提升系统的整体性能和速度。剪枝分两步进行：第一步进行因子分析，随后进行K均值聚类。

1) 数据预处理： 我们需要对离线工作负载以及在线工作负载B和C中的数据进行预处理。数据预处理按以下步骤进行：

- 重复列删除：我们删除了在所有工作负载中都只有常量值的列或指标。
- 布尔调优参数转换：我们还进行了其他一些预处理步骤，例如将所有布尔调优参数的值转换为整数（0 和 1）。

- 按工作负载划分文件：我们根据工作负载 ID 对所有文件进行了分类；因此，对于离线工作负载，我们有 58 个工作负载文件；对于在线工作负载 C，我们有 100 个工作负载文件；对于在线工作负载 B，我们将每个工作负载划分为 5 行用于工作负载映射，并从每个工作负载中取 1 行用于验证。

2）因子分析：冗余属性或指标包括彼此高度相关的指标，以及粒度不同的指标。在此步骤中，我们会移除高度相关的指标。我们采用一种称为因子分析 [2] 的降维技术，将高维数据转换为低维数据。

具体来说，给定一组具有任意相关性的实值变量，因子分析会将这些变量简化为一组规模更小的因子，以捕捉原始变量的相关模式。

每个因子都是原始变量的线性组合；因子系数与线性回归中的系数相似，也可以用相同的方式来解释。此外，每个因子的方差均为单位方差，且与其他所有因子都不相关。这意味着，可以根据各因子对原始数据变异性的解释程度对其排序。

因子分析算法的输入是一个矩阵 X，其行对应指标，列对应我们尝试过的调优参数配置。矩阵元素 X_{ij} 是配置 j 下指标 i 的值。因子分析会生成一个更小的矩阵 U：U 的行对应指标，列对应因子，而元素 U_{ij} 是指标 i 在因子 j 中的系数。我们可以使用 U 的第 i^{th} 行元素作为指标 i 的坐标，将各指标绘制成散点图。如果指标 i 和 j 在 U 中的系数相似——也就是说，它们在 X 中往往呈现强相关性——那么它们在散点图中的位置就会彼此接近。此时，移除冗余指标意味着移除散点图中彼此过于接近的指标。

这里使用来自离线工作负载的数据，共包含 58 个工作负载。首先移除所有与旋钮参数相关的数据，再将截取后的数据输入因子分析模块。查看生成的所有因子的特征值后发现，只有最初 30 个特征值大于 1 的因子对于数据库管理系统指标数据而言具有显著性。这意味着大部分变异性都由前 30 个因子解释。因此，仅考虑最显著的因子。图2展示了各因子的特征值，即将因子与其对应的特征值绘制在图中。

3) K 均值聚类：K 均值聚类用于识别指标并将其划分为有意义的组。它通过对数据进行聚类，将数据划分为 n 个方差相等的组。该算法致力于最小化簇内各数据点之间的距离。K 均值模型的输入是一个矩阵，其中包含通过因子分析得到的 30 个因子。可以选取距离各聚类中心最近的指标。使用了 Scikit-learn 对 K-means算法的实现 [3] 。

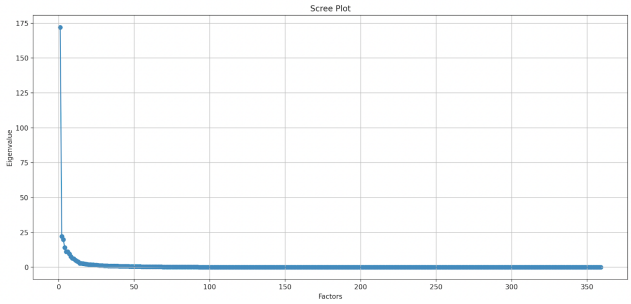


图2. 每个因子的特征值

K 均值无法确定最优簇数。首先，针对 2 至 15 个不同的簇数进行 K均值聚类，然后确定最优簇数。为了避免对簇进行人工解读，我们进行了轮廓分析，以研究簇之间的分离距离并确定最优簇。轮廓值通过测量某个簇中的点与其相邻簇的接近程度来确定。其取值范围为 -1 至 +1。值越接近 1，表示决策边界距离相邻簇越远；而值越接近 -1，则表示各簇彼此越近。因此，我们计算了簇配置数为 2 至 15 的 K-means 模型的轮廓系数。接近 +1 的最高轮廓系数表明该聚类配置最优，因此将其选定。

由于确定最佳簇数量为 8，因此将指标划分为 8 个簇，并计算这些簇的质心。从每个簇中选取一个与质心距离最近的指标。该指标将代表整个簇。确定目标指标后，我们会从数据中移除所有其他冗余指标，然后继续下一步。通过此过程筛选出的指标包括：

- `driver.jvm.pools.Code-Cache.committed.avg_inc`
- `driver.jvm.pools.Code-Cache.committed.avg_period`
- `driver.BlockManager.memory.maxMem_MB.avg`
- `driver.BlockManager.memory.onHeapMemUsed_MB.avg`
- `driver.LiveListenerBus.queue.executorMgmt.size.avg`
- `executor.jvm.pools.PS-Old-Gen.cmt.avg`
- `worker_1.Disk_transfers_per_second.dm-0`
- `worker_1.Paging_and_Virtual_Memory.pgpg`

B. 自动调优

1) 预处理：除“延迟”指标外，所有经过剪枝的指标和旋钮参数都通过去除均值并缩放至单位方差进行了归一化。在机器学习算法中使用数据之前进行零均值缩放是一种非常常见的做法。这一步对于避免计算均方误差时数值溢出也至关重要。

2) 高斯过程回归：高斯过程回归（GPR） [4] 是一种基于概率的监督学习方法，用于训练模型。GPR 需要指定先验。本例中的先验是此前见过的离线工作负载，并在初始阶段对其进行拟合。先验的协方差由所选核函数指定。拟合过程中，通过最大化对数边际似然来优化超参数。我们尝试了两种不同的核函数，并选择了表现最佳的一种。与其他核函数相比，径向基函数（RBF）的表现更好。使用缩放后的旋钮参数和指标来拟合模型。此外，还使用了 Scikit-learn 对高斯过程回归的实现 [4] 以及 OtterTune 对 GPR 的实现。

3) 工作负载映射：理想情况下，下一个组件是选择对系统影响最大的旋钮参数。这一步已经由我们完成，所提供的数据集也已经包含了重要的旋钮参数。

在工作负载映射中，当前目标工作负载会映射到之前收集的工作负载。在我们的案例中，我们通过借助欧几里得距离寻找最近邻，并计算每个工作负载的“得分”（即所有指标上的这些距离的平均值），将离线工作负载映射到在线工作负载B和C。

工作负载映射所遵循的逐步过程如下：

- 计算目标工作负载（第 1 次迭代中的工作负载 B）的指标向量与离线工作负载中每个工作负载对应向量之间的欧几里得距离。对于每个指标，都会执行此计算。
- 通过计算这些距离在所有指标上的平均值，为每个工作负载 i 计算一个“得分”。然后，算法选择得分最低的工作负载，认为它与目标工作负载最相似。
- 将目标工作负载扩展到最近的源工作负载；如果最近源工作负载与目标工作负载之间存在调节参数配置冲突，则保留目标工作负载的配置。
- 使用“增强工作负载”训练模型，并预测基于在线工作负载 B 创建的验证集上的延迟。
- 重复此过程，将源工作负载设为“增强工作负载”，将目标工作负载设为“工作负载 C”，并创建最终的增强工作负载，用于训练模型以及在 test.csv 上进行最终预测。

4) 延迟预测：延迟预测分为两个阶段。第一阶段使用在线工作负载B中所有工作负载的第6行来预测延迟，训练数据则为离线工作负载以及附加了来自在线工作负载B的工作负载映射数据。随后估算延迟值。在下一阶段，使用基于工作负载B和工作负载C附加数据训练的模型，对 test.csv 工作负载进行预测。此过程为高斯过程回归模型增加了更多数据，从而实现更准确的预测。

III. 扩展

A. 高斯混合模型聚类

原始的 OtterTune 论文 [1] 使用 K-means 对指标进行聚类。然而，使用 K-means 存在一些缺点。它假设簇始终呈球形。此外，它只考虑簇的均值，而不考虑其方差。各个簇中的观测值数量也大致相等。另外，它还对离群值非常敏感。为克服这些不足，我们将 K-means 聚类替换为高斯混合模型聚类 [5]。

高斯混合模型聚类通过假设每个簇都对应一个不同的高斯分布，并将属于同一分布的数据点归为一组，克服了这些缺点。因此，由于它兼顾了簇的方差和均值，生成的簇效果更好。可以将混合模型视为对 K 均值聚类的推广：它不仅利用潜在高斯分布的中心，还纳入了数据协方差结构的信息。我们使用的 Scikit-learn 中的 GaussianMixture [5]实现了用于拟合高斯混合模型的期望最大化（EM）算法。期望最大化是一种成熟的统计算法，通过迭代过程解决这一问题。首先，假设存在随机分量（其中心可随机选取自数据点、通过 K 均值学习得到，甚至仅假设围绕原点呈正态分布），并计算每个点由模型各分量生成的概率。然后，调整参数，使给定这些分配时数据的似然最大化。重复此过程可确保算法始终收敛到局部最优。GaussianMixture 模块提供了不同选项，用于约束所估计的各类别协方差：球形、对角、共享或完整协方差。我们使用完整协方差选项，此时每个分量都有自己的通用协方差矩阵，因此各个簇可以独立地采用任意位置和形状。

利用轮廓系数和高斯混合模型的 BIC 分数，确定了最佳簇数量。

最终获得的指标集如下：

- `driver.BlockManager.memory.onHeapMemUsed_MB.avg`
- `driver.BlockManager.memory.remainingMem_MB.avg`
- `driver.DAGScheduler.job.allJobs.nb_change`
- `driver.DAGScheduler.stage.waitingStages.avg`
- `driver.jvm.non-heap.usage.avg`
- `driver.jvm.pools.Code-Cache.usage.avg`

B. 随机森林算法

本文尝试采用随机森林算法替代高斯过程回归。与高斯过程回归相比，随机森林的优势之一是能够很好地处理高维数据。此外，随机森林基于装袋算法，并采用集成技术，在数据子集上构建许多树，然后将所有输出进行组合。通过这种技术可以减少过拟合，而高斯过程回归不具备这样的机制，因为先验是在初始阶段拟合的。尝试随机森林的另一个原因是利用

频率学派方法，而不是像高斯过程回归这样的概率方法，因为频率学派方法在数据量较多时表现更好。此外，由于模型复杂度较高，高斯过程回归的训练时间比随机森林长得多。尽管本文使用的数据相对较少，但在数据量较多时，随机森林可能是合适的技术，并且可能更为可取。随机森林算法 [6] 的实现深度最高为 50 层，并包含 200 个估计器。

C. 神经网络

OtterTune 论文[1] 中使用高斯过程回归的一个主要缺点是，它在处理高维数据时表现不佳。因此，在 OtterTune 中必须对指标进行筛选。为克服这一缺点，我们尝试使用神经网络作为替代方案，以观察其在高维数据上的表现。人工神经网络能够学习并建模包含可训练参数的非线性复杂关系。此外，人工神经网络的输出并不完全受限于专家系统最初提供的输入和结果。人工神经网络还具有泛化输入的能力。在从初始输入及其关系中学习后，它还能够推断未见数据中的关系，从而使模型能够泛化并对未见数据进行预测。

我们创建了一个简单的两层神经网络模型，其中包含一个全连接隐藏层。为此使用了 Keras库。由于我们希望直接预测数值而不进行转换，因此输出层不使用激活函数。我们采用高效的 ADAM优化算法，并对 MAPE损失函数进行优化。之所以特别选择这一指标，是因为我们将用它来比较所有模型。我们也尝试添加另一个隐藏层，但 MAPE 得分并未因此改善。

IV. 实验

A. 评估

在 K均值聚类中，采用轮廓系数分析来评估数据中的最佳簇数量。

在工作负载映射阶段，分别尝试了均方误差和平均绝对百分比误差用于评估。根据延迟预测结果可以观察到，在工作负载映射中，采用均方误差作为评估标准的表现更好。在延迟预测评估中，则使用平均绝对百分比误差进行评估。

B. 缩放和超参数调优

分别在缩放和不缩放的情况下进行了实验。实施并分析了两种不同的缩放变体。在第一种变体中，仅对工作负载映射和高斯过程回归模型的输入进行了缩放。在第二种变体中，除延迟列之外的所有数据均进行了缩放。总体而言，将除延迟列之外的所有数据缩放为零均值和单位方差往往能够提升系统性能。在使用未缩放的数据时，观察到预测值总体上与真实值相差甚远。

对高斯过程回归进行了超参数调优。对于高斯过程回归，通过将 alpha参数设置为较低的值来控制噪声水平。从表 I 可以看出，随着 alpha 值降低，模型表现良好。针对不同深度构建了随机森林，并对不同数量的估计器进行了评估。评估得分最佳的配置被选作超参数。

表 I 高斯过程回归超参数调优

阿尔法	***MAPE***
1e+8	*168.63*
1e+7	*142.01*
1e+5	*123.67*
1e+3	*101.29*
1e+1	*98.45*
1e-1	*69.61*

我们还尝试为神经网络增加一个隐藏层，但这并未改善平均绝对百分比误差得分。

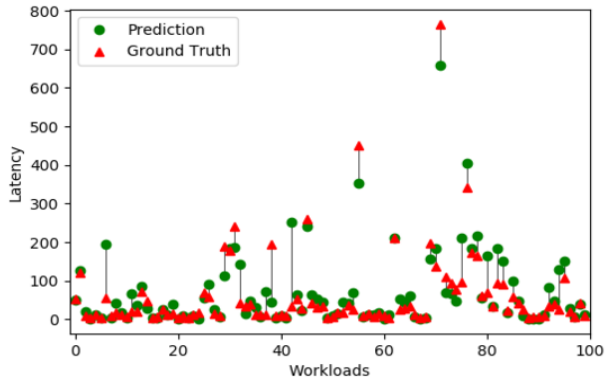
五、结果

表 II 展示了在在线工作负载 B 的测试集上评估不同模型所得的结果。从表中可以看出，用高斯混合模型聚类替代 K 均值后，模型在平均绝对百分比误差和均方误差这两个指标上的表现均略有改善。然而，随机森林和神经网络的表现都不如高斯过程回归。这主要是因为本研究使用的数据量较少。从表 II 中的均方误差值可以清楚地看出这一点。数据量较少时，神经网络更容易过拟合，这可以从较高的均方误差值中得到验证。总体而言，我们的实验表明，使用高斯混合模型聚类能够略微改善结果。

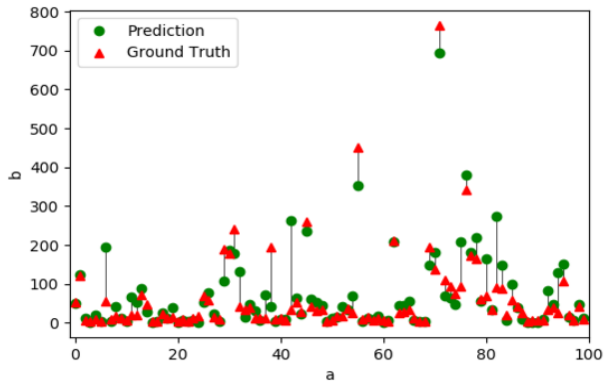
表 II 结果摘要

Type	***MAPE***	***MSE***
基线	*69.61*	*2118*
EM 聚类	*67.85*	*2329*
随机森林	*78.98*	*3817*
神经网络	*77.26*	*13426*

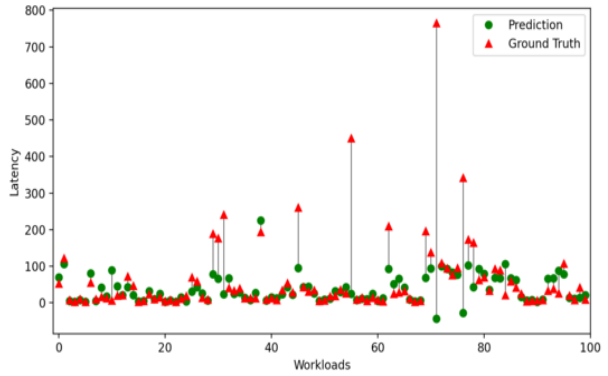
图 3 展示了基线、EM 聚类和神经网络在在线工作负载 B 上对延迟的真实值和模型预测。图中显示，与基线方法相比，使用 EM 聚类算法的指标所做出的预测更接近真实值。然而，神经网络的预测值则相差甚远。



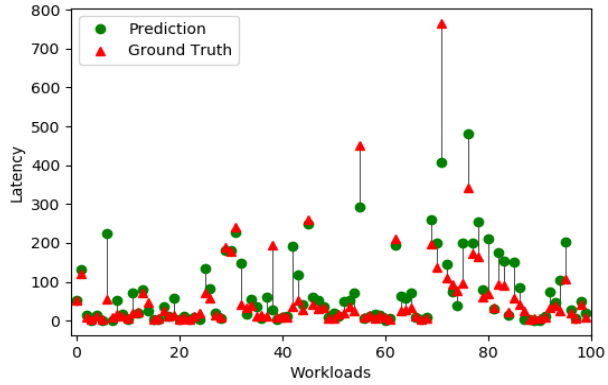
(a)



(b)



(c)



(d)

图 3. 在线工作负载 B 测试集上的预测值与真实值延迟预测（a）基线（b）EM 聚类（c）神经网络（d）随机森林

VI. 结论

自动数据库管理系统调优仍是一个活跃的研究领域。我们提出了一种自动化方法，该方法利用以往经验并收集新信息来调优数据库管理系统配置。在实验中，我们使用基线实现达到了 69% 的平均绝对百分比误差；该实现使用因子分析、K-means 进行指标剪枝，并使用高斯过程回归进行预测建模。

我们的实验还表明，通过用 EM 聚类替代 K 均值聚类，我们能够将平均绝对百分比误差降低到 67%，这表明可以进一步探索高斯混合模型聚类，将其作为 K 均值聚类的替代方案。尽管我们所使用的数据集非常小，无法明确断言 EM 聚类优于 K 均值聚类，但我们建议进一步研究 EM 聚类，将其作为 K 均值聚类的替代方案，因为 EM 聚类适用于非线性几何分布，并且不像 K 均值聚类那样假设簇的形状为球形。

我们还试验了随机森林等集成方法，其平均绝对百分比误差为 78%；同时也试验了神经网络等非线性模型，其平均绝对百分比误差为 77%。然而，在这两种情况下，采用高斯过程回归的基线模型都优于神经网络模型和随机森林模型。未来一项有益的工作是收集更多数据，并采用相同的方法进行研究。

参考文献

[1] Dana Van Aken、Andrew Pavlo、Geoffrey J. Gordon 和 Bohan Zhang。2017 年。通过大规模机器学习进行自动数据库管理系统调优。载于 *2017 年 ACM*数据管理国际会议论文集，SIG-MOD '17，第 1009–1024 页，美国纽约州纽约市。美国计算机协会。[2] Scikit-learn 中的因子分析实现。https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FactorAnalysis.html [3] Scikit-learn 中的 K 均值聚类实现。https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html [4] Scikit-learn 中的高斯过程回归实现。https:// scikit-learn.org/stable/modules/gaussian process.html [5] Scikit-learn 中的高斯混合模型（EM 聚类）实现。https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture[6] Scikit-learn 随机森林实现。https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html