容器中程序特征分析业界调研报告

本报告主要调研大数据平台下程序性能优化方法，就目前调研情况，共调研文章52篇，具体可参见附件A。根据论文与项目的相关性，我们将论文的方法分为三大类：基于搜索优化配置的方法、基于机器学习优化配置的方法、减少训练样本量的方法。以下关于调优配置的部分文章介绍。

## 一 基于搜索的方法

### 1.1背景

在大数据分析平台上为运行重复作业选择正确的配置参数是困难的，因为有数百个可能的配置参数可供选择，如果使用默认配置设置，则难以开发系统的性能潜力。最佳配置和最差配置之间的差异对性能的影响可能超过10倍。良好的配置可以在特定工作负载下大大提高已部署系统的性能。但是由于spark有两百个参数，决定哪种配置能带来最佳性能就成了一项非常昂贵的任务。

近年来，使用较为广泛的搜索方法主要包括：递归搜索、进化算法、贝叶斯搜索等。对于每个工作负载，这些参数中只有一小部分对整体性能有显著影响，试图调整无影响的参数只会浪费资源，并减缓接近最优解的搜索。因此，在对配置进行搜索之前，往往会伴随降维操作，即只对重要配置参数调优。目前使用较多的降维方式包括：专家建议、PCA、机器学习方法等。

目前，我们调研基于搜索调优系统的方法主要有：贝叶斯优化、遗传算法、递归搜索。

### 1.2学术界最新进展

#### 1.2.1 To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics

该论文发表在SIGKDD2020，提出了Tuneful配置调优框架用于调优spark配置。主要思想是**通过贝叶斯优化算法搜索spark的最优配置来减少负载的执行时间，降低调优成本。**流程包括探索重要参数、使用贝叶斯优化算法搜索执行时间较少的最优配置，以及相似性分析。

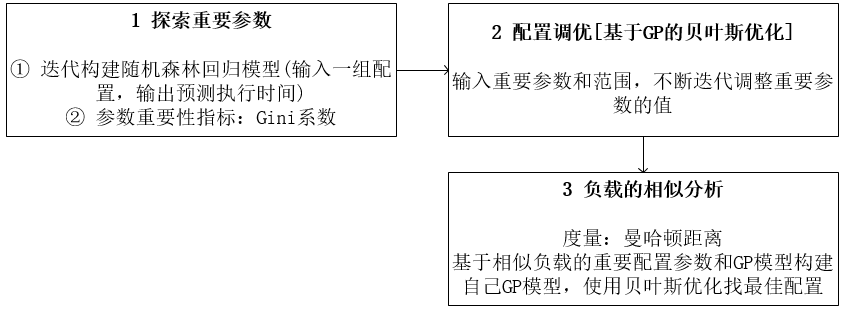


图1-1 Tuneful方法流程图

**探索重要参数**。构建随机森林回归PFR模型，用于预测给定配置的执行时间。**迭代进行随机森林建模，对剩余有影响的参数进行排序和筛选**，每一次迭代都会淘汰排名最低的几个参数，直到预测模型的误差<40%时**结束排序，剩余参数表示对性能影响最大的重要参数**，后续仅对这一步选出的重要参数搜索调优。

**配置调优[基于高斯过程的贝叶斯优化]**，直到期望改进降低至 10% 以下时停止搜索，输出执行时间最少的最优配置。

**负载的相似分析**。将新负载与已调优负载进行相似比对，**相似则延用已调优负载的高斯函数和重要参数，在这个基础上进一步针对当前负载进行贝叶斯优化**来降低新负载的调优时间。如果找不到与新的应用程序相似的已调优负载，则重复探索重要参数和贝叶斯调优过程。

**调优效果**。如表1-1所示Tuneful 搜索到额配置**可与最先进的调优系统达到相同效果**；与默认配置相比，Tuneful能够得到不错的优化结果。

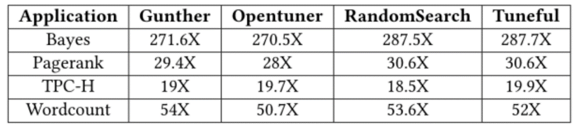


表1-1 与默认配置相比，四种调优算法搜索的最优配置的加速倍数

**调优成本**。如图1-2所示在不考虑已有知识情况下（不考虑相似分析）Tuneful与Gunther、Opentuner、RandomSearch相比，**平均搜索时间短 2.7 倍**。即Tuneful算法能够在较低的成本下实现和其他算法一样的效果。

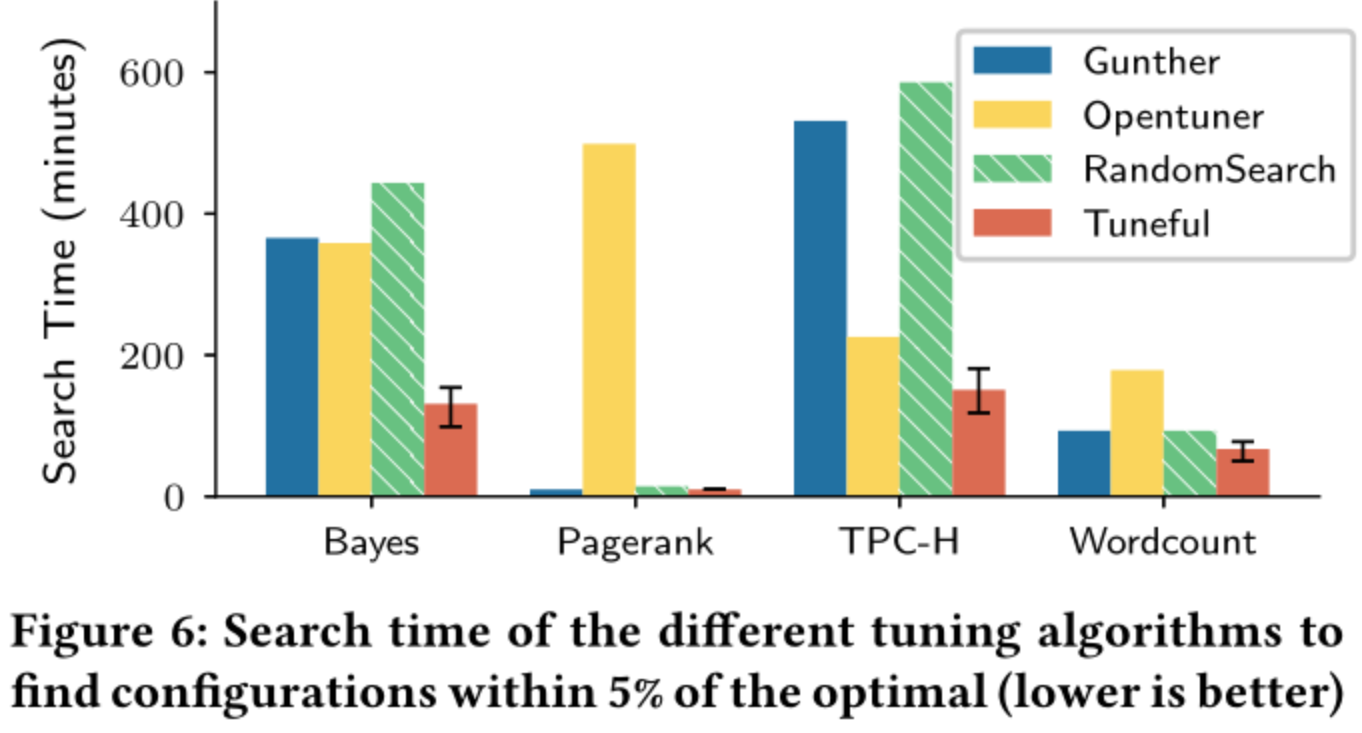


图1-2 四种调优算法搜索时间对比

#### 1.2.2 Genetic Learning Particle Swarm Optimization

该论文发表在IEEE Transactions on Cybernetics2016，通过**将粒子群算法和遗传算法结合进行spark配置调优，提高算法的全局搜索能力和搜索效率。**流程包括利用遗传算子生成样本、粒子从中学习、粒子的历史搜索信息反过来为样本的演化提供指导。这样构建的样本不仅多样化，而且质量高。通过学习遗传算法构造的样本，使粒子的搜索更加多样化，不再简单地由gbest和pbest引导，有助于避免粒子群算法的过早收敛。同时由于遗传算法中选择算子的作用，幸存样本的质量较高，能够对粒子提供有效指导，从而提高粒子群算法的搜索效率。

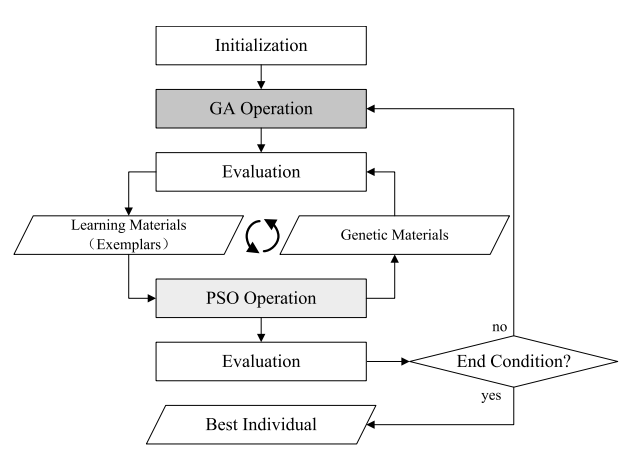


图1-3 GL-PSO流程图

**遗传算法生成初始种群**。初始化样本点后先使用遗传算法产生第一代种群。

**粒子群算法更新粒子**。粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。之后粒子群将更新后的样本传播给遗传算法。

**遗传算法执行选择、交叉和变异操作来改进样本**。交叉算子利用粒子群算法中粒子的历史搜索经验来提高基因质量，一个好粒子更有可能在其pbest位置和gbest位置之间进行算术交叉。随机选择20%M（种群规模）的样本加入竞赛，其中适应度最好的赢家将取代粒子当前的样本。

表1-2给出了求解多模态函数（包含了大量的局部最优值，可能会导致算法过早收敛）时，八种算法获得的误差值的平均值、最佳值和标准偏差（越低越好），其中最佳结果用粗体标记。就解的精度而言，GL-PSO算法在七个多峰函数中的五个函数中表现最好。

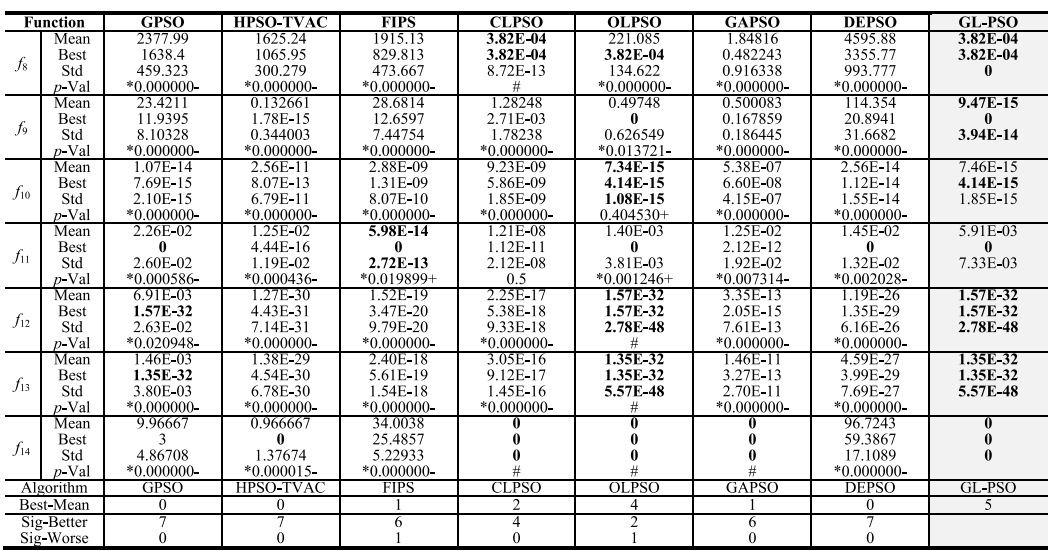


表1-2 八种算法在求解多模态函数上误差值的平均、最佳、标准差，以及p-Val（其他算法与GL-PSO对比）

比较算法在求解多模态函数时的搜索速度（FEs）和可靠性（SR%） 每行中的最佳值以粗体标记。从表1-3中可以看出，GL-PSO通常具有最高的搜索速度和可靠性。

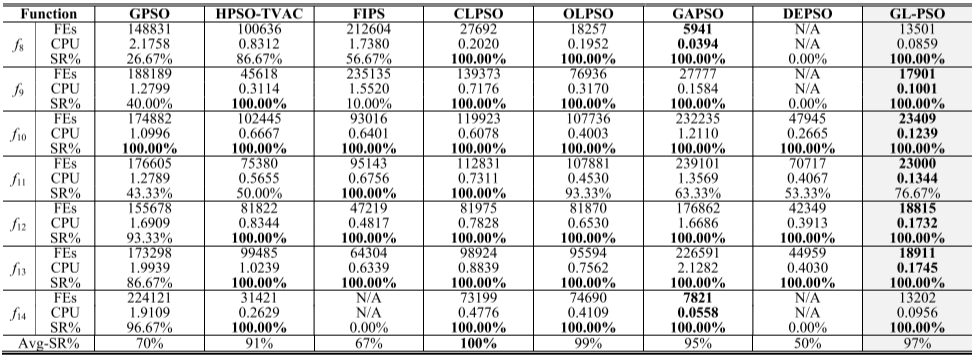


表1-3 八种算法在求解多模态函数的搜索速度和可靠性对比

#### 1.2.3 Gunther: Search-Based Auto-Tuning of MapReduce

该论文发表在European Conference on Parallel Processing2013，提出了一种基于搜索的方法Gunther，用于Hadoop MapReduce优化, **使用专门设计的遗传算法来搜索接近最佳作业执行时间的参数设置。**

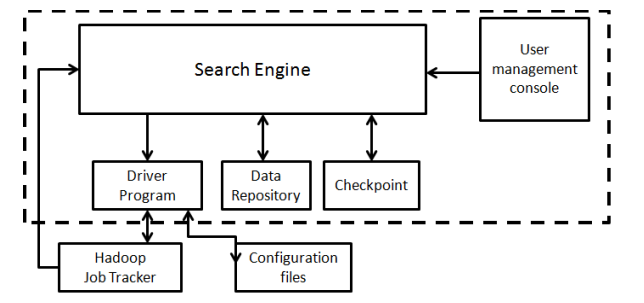


图1-4 Gunther方法流程图

**选择算子**。作者认为在真实的集群中，性能不佳往往是由于一些关键参数配置不当而导致，为了保证糟糕的配置不被继承，在选择的时候就应该尽快消除这种配置。经典GA通过轮盘赌策略选择父母样本来产生下一代个体，改进的遗传算法**计算每一代种群的平均适应度，并且只选择适合度得分超过平均值的个体作为父母**。

**更新算子**。作者认为更新算子如果在后代具有较低适应度值的时候仍直接替换父母，会导致算法并不总是保留更好的解，这会减慢收敛速度。经典GA直接使用产生的后代替换父母个体。改进的遗传算法**比较后代和父母的适应度大小，若后代的适应度更低，则不替换父母，若后代适应度更高则替换父母个体**。

**非冗余设计**。经典GA不记得先前的搜索，并且可能会多次评估某些区域。作者增强了GA以**记住搜索过的区域避免重复搜索**。

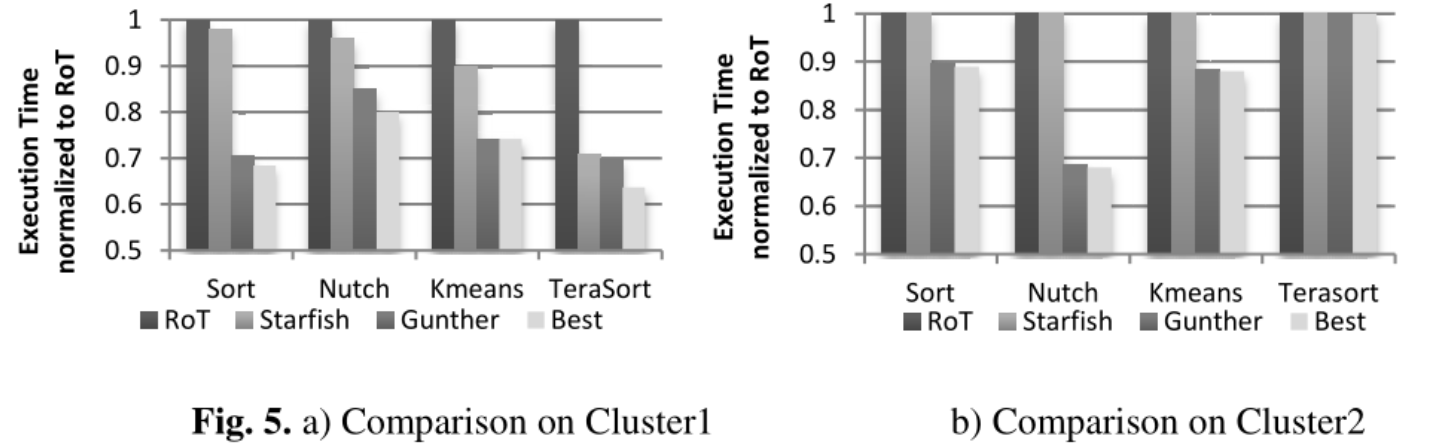


图1-5 与RoT（经验调优）相比，搜索时间的优化比例

Cluster1表示具有**存储瓶颈**的集群。如图1-5所示与经验法则（ RoT ）相比，Gunther在所有工作负载中的**性能（执行时间）平均提高了25%**。Terasort实现了30%的最大改进。

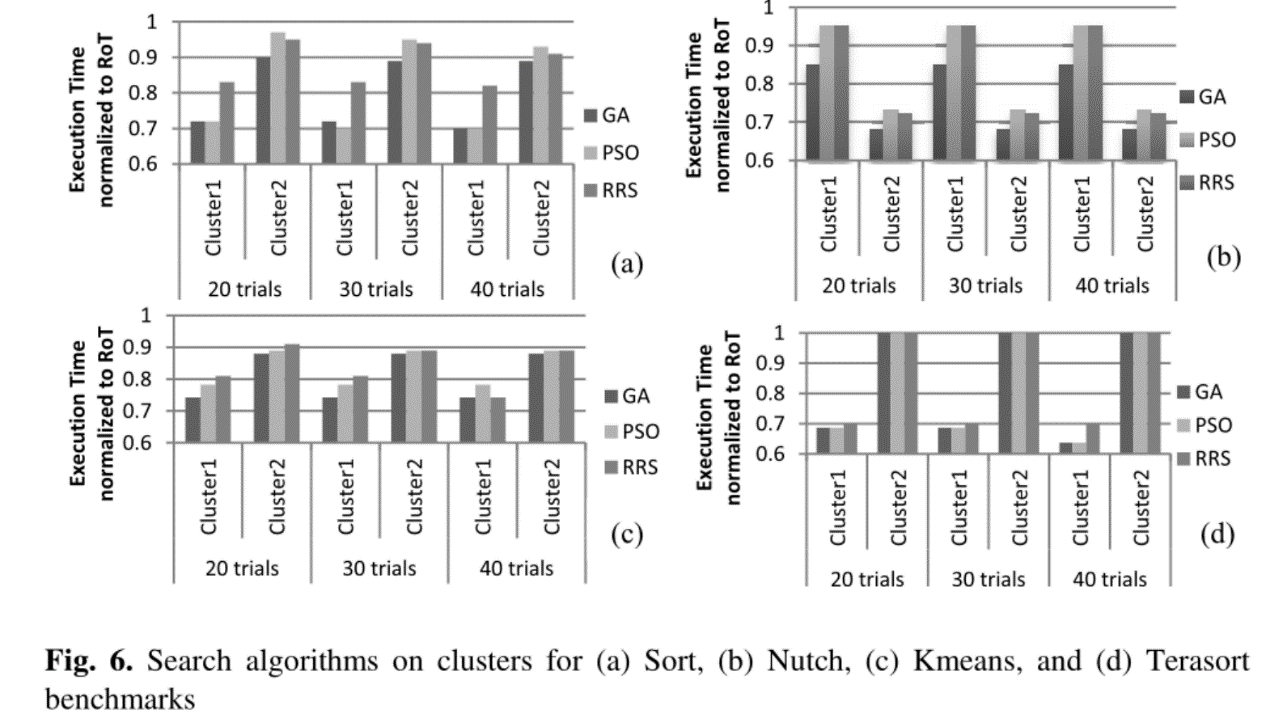
Cluster2表示具有**网络瓶颈**的集群。如图1-6所示Gunther能够将**性能（执行时间）提高了33%**，接近最佳水平。

图1-6 相比PoT（经验调优），三种算法在20次、30次、40次搜索下性能（执行时间）提升比例

如图1-6所示Gunther比粒子群算法和RRS算法获得了更高的性能，并且在**大约20次试验中收敛**，而粒子群算法和RRS算法往往需要40多次试验。

#### 1.2.4 MonkeyKing Adaptive Parameter Tuning on Big Data Platforms with Deep Reinforcement Learning

该论文发表在Big data2019，提出了MonkeyKing用于spark配置调优。利用过去的经验，收集新的信息，调整大数据平台的参数配置。**根据工作类型选择对性能影响较大的关键参数，然后结合深度强化学习对关键参数进行优化，提高工作绩效。**

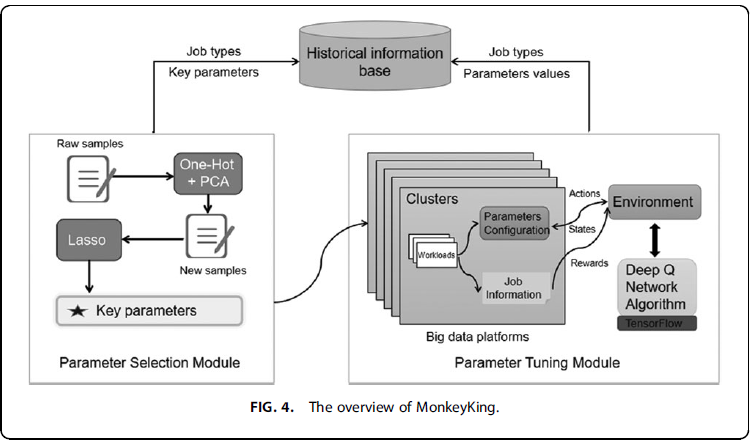


图1-7 MonkeyKing流程图

**探索重要参数**。使用参数数据处理和使用LASSO进行特征选择。通过**One-Hot编码**把所有参数转换为虚拟变量，然后结合**PCA主成分分析来降低维数**，之后对**数据规范化**。最后将剩下的候选参数带入**LASSO算法选择影响性能的最小候选参数集，输出重要参数以及对应的重要性权重**。

**配置调优**。当工作负载运行时，它将生成相应的参数配置信息和作业日志。**基于DQN算法**根据配置参数和日志信息（获取该配置下程序的执行时间）交互，以获得相应的动作、状态和奖励，来调优配置。同时，这两个模块的计算的参数调优结果会保存在历史信息库中。

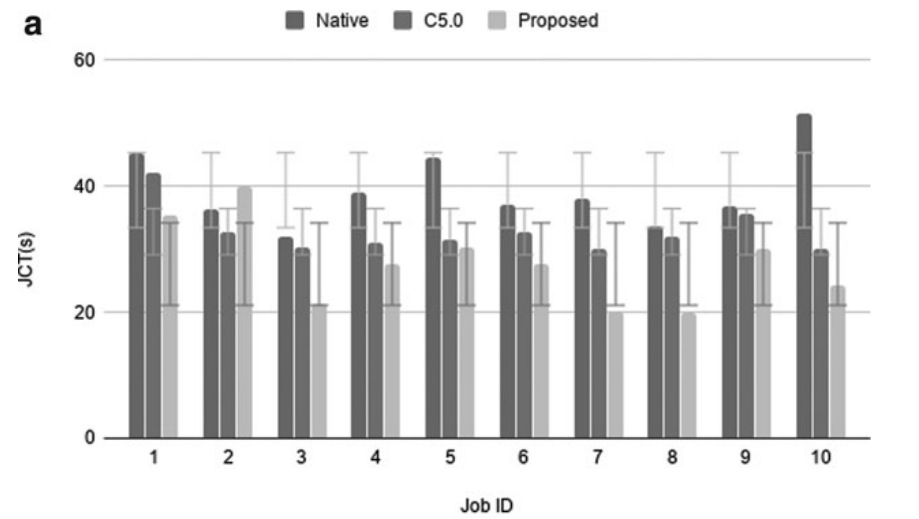


图1-8 MonkeyKing在WordCount上的标准差

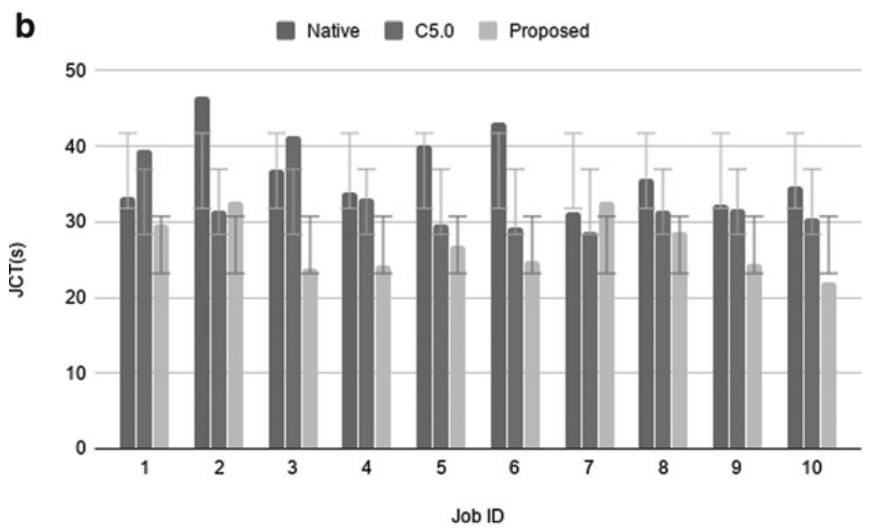


图1-9 MonkeyKing在Sort上的标准差

图中给出了专家经验与所提出方法的性能（执行时间）改进比例，如图1-8所示**WordCount 性能平均提高24.8%**，如图1-9所示**Sort性能平均提高19.7%**。

#### 1.2.5 BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning\*

该论文发表在SOCC2017，提出了BestConfig用于自动化调整系统的配置。为了在资源限制内调整系统配置，作者提出了DDS抽样方法和RBS递归界限搜索算法。

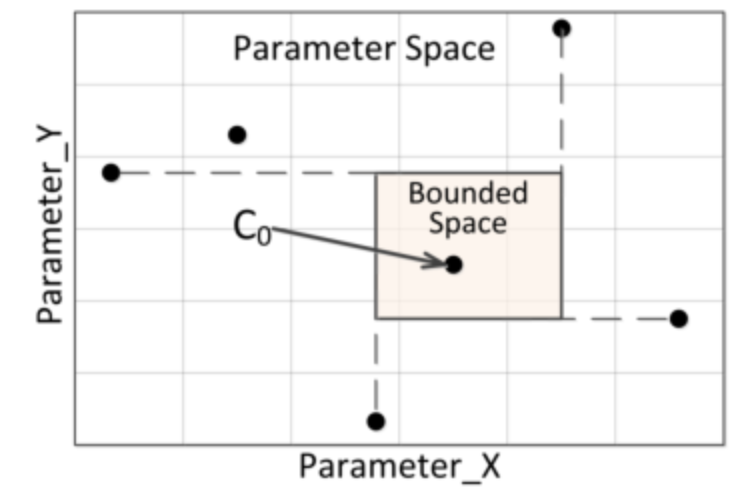


图1-10 DDS和RBS

**DDS（Divide and Diverge Sampling）抽样**。每个参数在取值范围内**划分为k个区间，参数在每个区间上进行一次抽样**，最终**得到k个样本**（无论一维、二维还是多维，都只会抽样k个样本）

**RBS（Recursive Bound and Search）搜索算法**，在DDS划分的值区间的基础上，在区域内进行最优性能点的搜索。给定一个初始样本集，**RBS找到性能最好的点C0。然后，在C0周围的有界空间中抽样另一组点。根据观察，很有可能会找到另一个表现更好的点(比如C1)。**我们可以在C1周围的有限空间中再次抽样。递归地执行这个搜索样本步骤，直到在样本集中找不到性能更好的点。

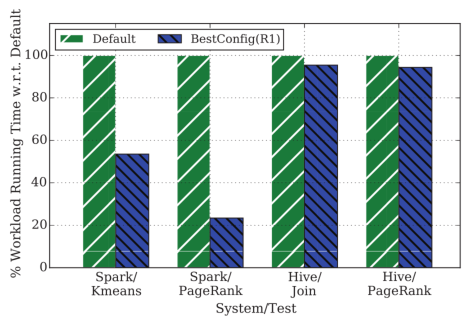


图1-11 与默认配置相比，BestConfig搜索的配置的性能（执行时间）提升比例

如图1-11所示通过配置调整与默认配置相比BestConfig将Hive join作业的运行时间降低了约50%，将Spark作业的**运行时间降低了约80%。**

### 1.3 技术方案对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | 方案一 | 方案二 | 方案三 |
| **技术方案介绍** | 贝叶斯优化 | 遗传算法 | 递归搜索 |
| **当前最新现状** | 贝叶斯优化(BO)被广泛用作各种领域中配置调整的有效的最新解决方案 | 目前在调优领域广泛使用遗传算法 | 前几年使用该算法文章较多，近3年使用很少 |
| **扩展性** | 更改替换模型 | 修改选择、交叉、变异算子 | 可以融进各个算法中 |
| **成本** | 搜索时间与数据维度成正比 | 搜索时间与种群迭代次数成正比 | 搜索速度快，但容易忽略潜在的更好的配置 |
| **方案优势** | 通过替换模型模拟需要优化的目标函数；相比其他算法收敛较快 | 基本没有什么前提假设，只需计算目标函数的值即可，对优化问题本身的性质要求是非常低 | 速度快，不会浪费时间在性能不好的区域进行重复抽样 |
| **方案劣势** | 在高维配置空间(超过10个参数)中提供快速收敛的问题仍然存在 | 容易陷入局部最优，过早收敛 | 效果在大多数情况下并不好。在当前表现最好的样本点附近试图寻找更好的点，可能会带来提升但提升不会很大，抛弃了其他地方性能更好的点。 |

表1-4 技术方案对比

## 二 基于机器学习的方法

### 2.1背景

随着Apache Spark的应用越来越广泛，也暴露出一些问题，最重要的方面之一是性能问题。Apache Spark有超过180个配置参数，用户可以根据自己的具体应用进行调整以优化性能。目前这些参数都是通过人工试错或者专家建议来调整的，但由于参数空间大，参数之间的交互复杂，效果不佳。

目前使用较广的配置调优方法之一是基于学习的方法，主要包括三个部分：收集数据、构建预测模型和搜索算法。收集数据用于收集具有不同配置的工作负载对应的执行时间。预测模型是利用收集的训练数据训练模型，使用各种机器学习算法来预测给定工作负载在任何一组配置下的性能（执行时间）。搜索算法在参数空间中根据预测模型的预测结果来探索最优配置。

目前，我们调研基于学习调优系统的方法主要包括使用决策树、随机森林、Adaboost模型预测负载执行时间，结合搜索算法进行配置调优。

### 2.2 学术界最新进展

#### 2.2.1 A Novel Method for Tuning Configuration Parameters of Spark Based on Machine Learning

该论文发表在HPCC2016，**使用决策树构建分类性能模型预测spark在给定配置参数下与参数默认值相比性能的提高比例，并结合随机递归搜索算法调优spark配置参数**。

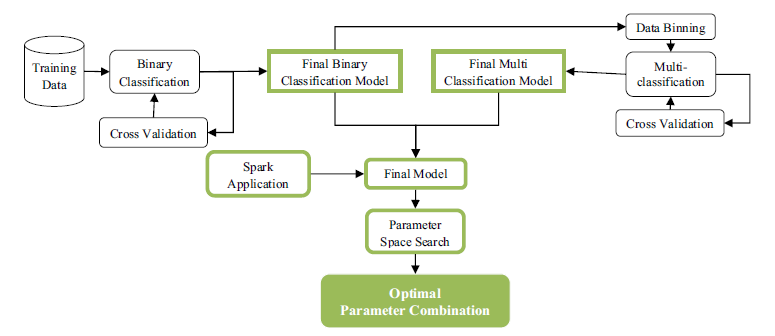


图2-1 方法流程图

**构建性能预测模型**。首先构建一个**二分类预测模型**，如果预测时间短于默认参数下的执行时间，则标签为1，否则标签为0。其次构建一个**多分类模型**，预测spark在给定一组参数下的执行时间相对于默认参数减少的百分比，比如将数据分为5类，它们的执行时间相对于默认参数减少5%、10%、15%、20%和25%。

**调优配置（递归随即搜索）**。得到最终模型后，利用递归随机搜索算法，首先识别出包含高概率最优参数组合的有希望区域，然后在这些区域中递归抽样，这些区域基于采集到的样本逐渐移动或收缩到局部最优参数组合，最后重新开始随机抽样，找到更有希望的区域重复递归搜索。

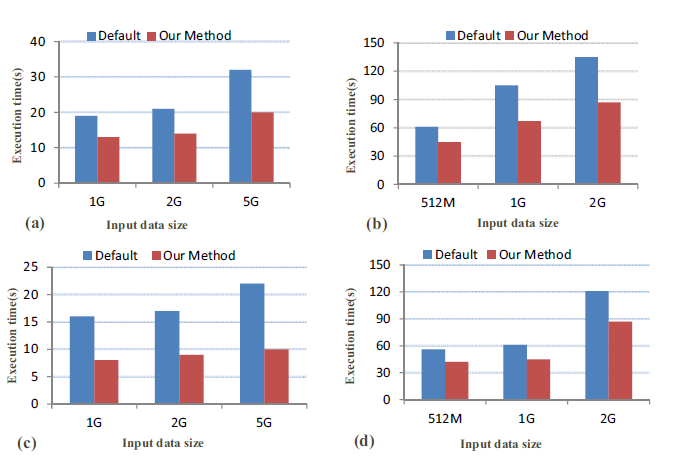


图2-2 与默认配置相比，提出方法在不同负载下的性能提升

（a）wordCount 使用我们的方法 1G提高了31%、5G提高了38%。平均提高34%（b）sort 性能最高提高36%，平均性能提高33%。（c）Grep 最高提升55%，平均性能提高51%（d）NavieBayes最高提高28%，平均26%。**总体来说我们的方法提高了接近36%**。数据量越大，提升就越明显。

#### 2.2.2 Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing

该论文发表在ASPLOS2018，提出了DAC数据感知的自动调优方法，可以有效地识别给定IMC程序的高维配置，从而在给定集群上实现最优性能。流程包括**专家经验选择41个重要配置，随机生成配置，收集训练数据分层建模构建性能模型、使用遗传算法搜索最优配置。**

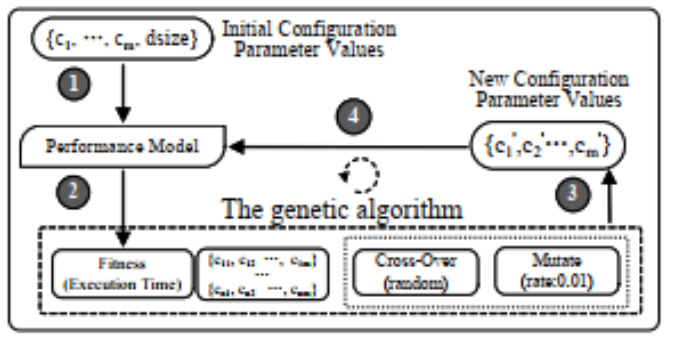


图2-3 DAC方法流程图

**分层构建性能预测模型**。以分层的方式将多个独立的子模型结合起来，递归地构建更高阶的模型。首先使用**回归树构建子模型M1**，然后**构建另一个不同的回归树子模型M2**反映M1没有捕捉到的spark作业执行时间的变化，之后将**前两个子模型组合生成尝试的一阶组合模型CM1**，之后递归生成二阶，甚至更高阶的模型，直到满足目标精度停止。

**配置调优（遗传算法）**。将初始种群的配置传入性能模型，将预测的执行时间传给GA，通过选择、交叉、变异操作生成一组新的配置参数值，将这些新生成的配置放入性能模型以获得预测的执行时间，再传入GA中搜索，重复该过程直到收敛找到最优配置。

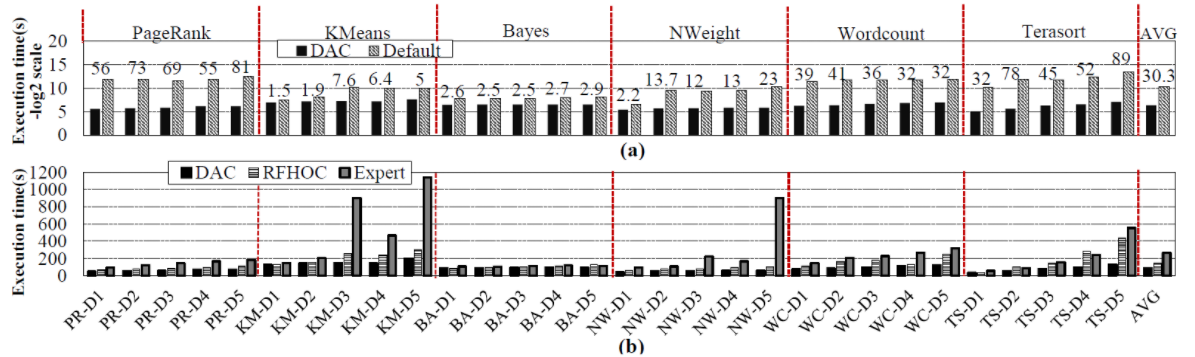


图2-4 （a）与默认配置相比，DAC在不同负载下性能（执行时间）优化比例

（b）DAC与RFHOC和专家经验相比性能的优化比例

如图2-4(a)所示DAC在默认配置下显著提高了30个程序输入对的性能，**平均提高了30.4倍**，最高提高了89倍。如图2-4(b)所示DAC的性能也明显优于RFHOC。与RFHOC相比，DAC的**平均加速为1.6倍**。

#### 2.2.3 ATCS: Auto-Tuning Configurations of Big Data Frameworks Based on Generative Adversarial Nets

该论文发表在IEEE Access2020，为了解决大数据框架构建性能预测模型时遇到的成本问题，**提出了基于生成对抗网络(Generative Adversarial Nets, GAN)的自动调整配置参数系统(ATCS)。**

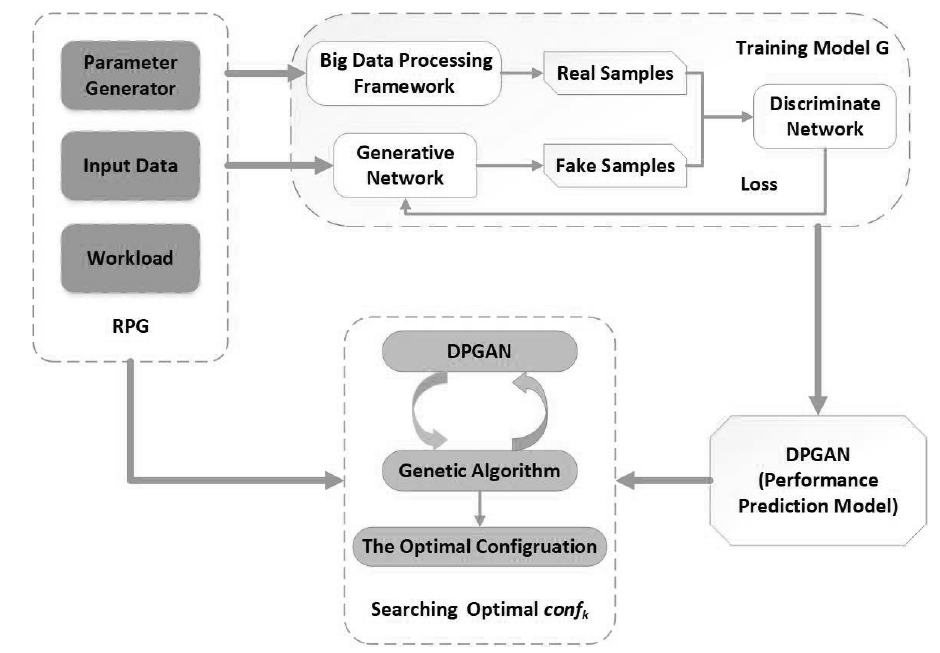


图2-5 ATCS方法流程图

**构建性能预测模型**。在GAN网络的基础上，对生成器进行修改，将参数的配置作为输入，执行时间作为生成样本，变相通过对抗训练的构建了执行时间的**预测模型**。最终实现**用较少的训练数据建立高精度的性能预测模型**。构建好性能模型后再采用GA算法搜索最优配置。相比于传统的树模型，深度学习模型，集成学习模型大幅度减少了抽样样本数，同时提高了模型性能。

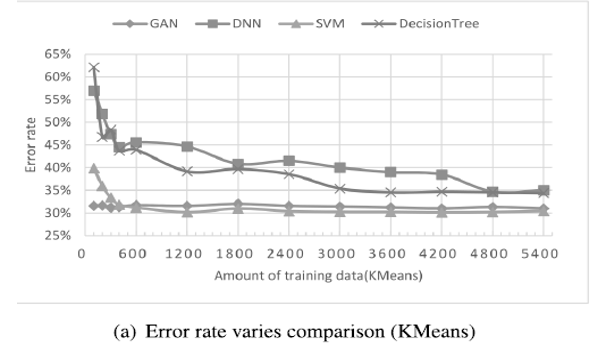


图2-6 比较四种性能模型的错误率和收敛速度

GAN要比其他模型**需要更少的数据**，更早收敛，且预测结果的错误率更低。

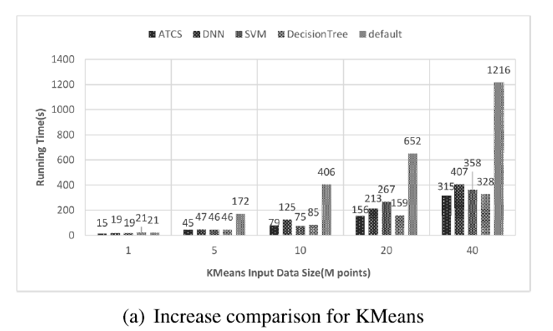


图2-7 与默认配置相比，使用四种方法搜索的最优配置性能提升比例以及所需训练样本量

如图2-6所示使用GAN的预测模型能够把**性能平均提升3.5倍**，最高提升6.9倍。如图2-7所示在相同性能下，**需要的训练数据只有DNN的6%，SVM的13%**，DT的18%。性能是DNN的1.7倍，SVM的1.6倍。

#### 2.2.4 Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model

该论文发表在Journal of Systems and Software2021，提出了一种**将多目标优化算法与性能预测模型相结合的优化配置搜索算法AB-MOEA/D**。其中使用Adaboost算法构建的性能模型来评估每个候选配置的适应度。采用**MOEA/D多目标优化算法**以最小化执行时间和成本为目标寻找配置。

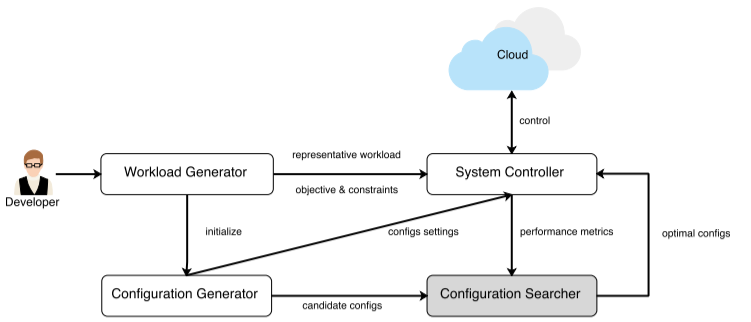


图2-8 AB-MOEA/D方法流程图

**性能预测模型**。Adaboost算法构建的模型用于评估每个解决方案的适应度值。构建一个性能模型来预测每个候选配置的执行时间和成本。该模型以Spark配置为输入，输出性能预测。

**配置调优[ MOEA/D多目标优化算法 ]**。将问题分解为N个单目标优化子问题，并同时求解，目标是找到最小化执行时间和成本的配置。使用交叉和变异算子产生新解，利用性能模型评估个体适应度。将遗传算子生成的新个体与原始个体进行比较，用较好的个体替换较差的个体。

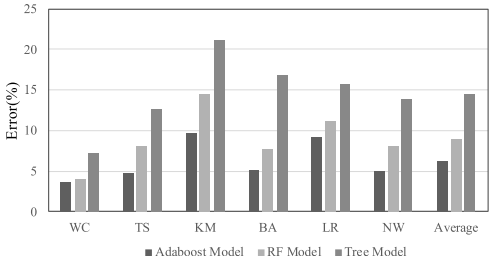


图2-9 三种性能模型的预测精度对比

Adaboost模型**在六个基准上的平均误差仅为6.23%**，其他两个模型的误差分别为8.97%、14.57%。

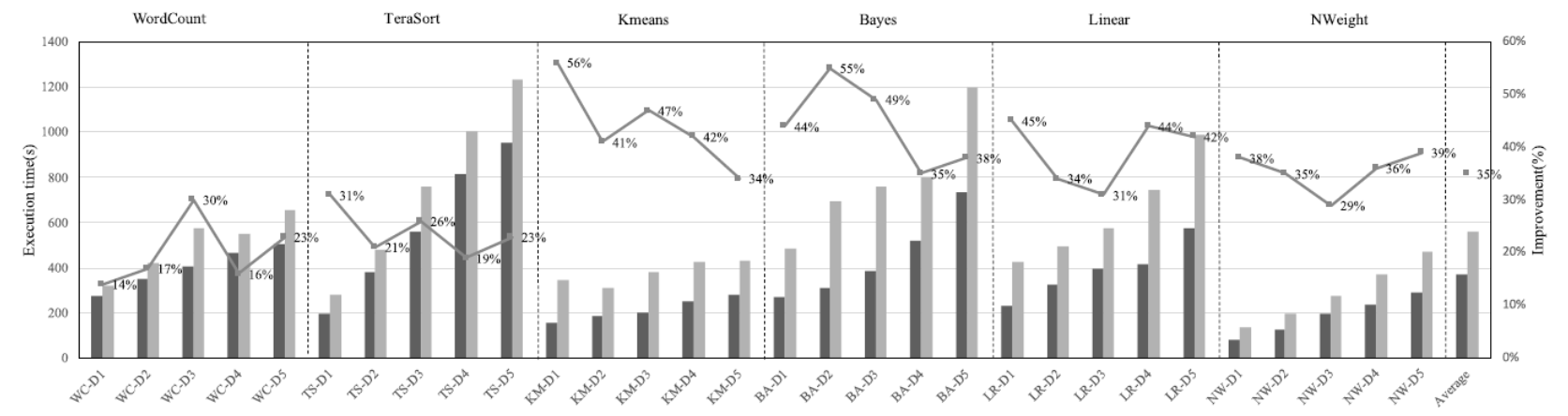


图2-10 AB-MOEA/D与BestConfig 再执行时间改进上的比较和改进百分比

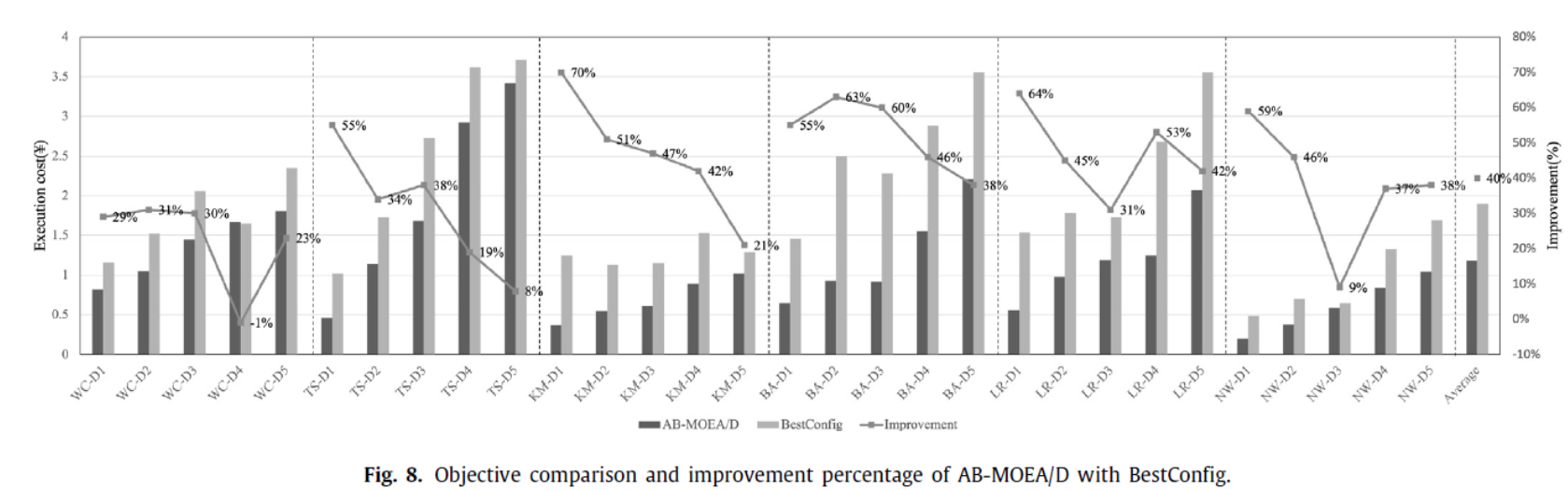


图2-11 AB-MOEA/D与BestConfig 再执行成本改进上的比较和改进百分比

如图2-10所示AB-MOEA/D在**执行时间方面的改进幅度为14% - 56%，平均为35%**。如图2-11所示AB-MOEA/D**执行成本的提高幅度从- 1%到70%，平均为40%。**

#### 2.2.5 Towards Automatic Tuning of Apache Spark Configuration

该论文发表在CLOUD2018，研究基于机器学习的方法来构建特定负载的性能模型，并使用它来调优spark平台上的应用程序性能。

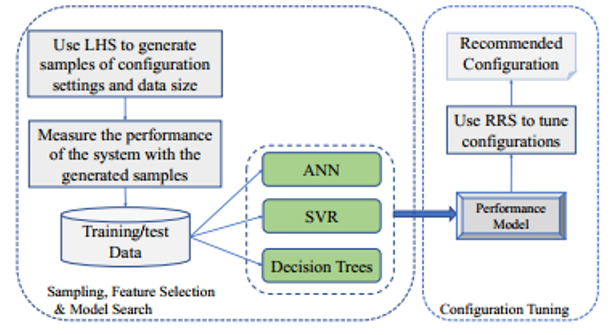


图2-12 方法流程图

**构建性能预测模型**。通过**LHS拉丁超立方抽样**来减少搜索空间，收集训练样本和测试样本。然后使用**神经网络构建性能模型预测特定配置参数对应的执行时间。**

**调优配置**。递**归随机搜索(RRS)算法来识别配置设置的候选组合**。搜索过程中不通过实际运行负载来获得运行时间，而是使用前面步骤中确定的性能模型来预测系统的运行时间。

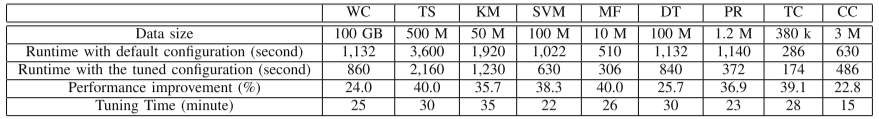


表2-1 方法流程图

如表2-1所示该调优方法与默认配置相比可以将Spark的**性能提高22.8%到40.0%**。

### 2.3技术方案对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 方案一 | 方案二 | 方案三**…** | 方案四 |
| **技术方案介绍** | 决策树模型 | 随机森林 | GAN | Adaboost |
| **当前最新现状** | 使用较少，并且实验的模型精度章节不会与集成学习算法对比效果。 | 属于集成学习类别，使用较广泛 | 主要用于减少建模所需的训练样本的收集时间 | 属于集成学习类别，使用较广泛 |
| **主要适用场景** | 数据维度低时使用较广，复杂时（spark调优）使用得很少 | 数据纬度高时使用较广泛 | 近两年才开始把GAN使用在大数据配置调优领域 | 数据纬度高时使用较广泛，但不如随机森林使用的广 |
| **实时性** | 可以在建模过程中输出重要参数 | 可以在建模过程中输出重要参数 | 不能边建模边进行参数排序 | 可以在建模过程中输出重要参数 |
| **成本** | 预测精度和数据量成正比 | 预测精度和数据量成正比 | 可以构造数据达到足够数据量 | 预测精度和数据量成正比 |
| **方案优势** | 速度快，单一模型较简单 | 采用Bagging集成学习算法，精度比大多数单个算法好，精确度更高 | 能够减少收集训练数据的成本同时保证精确度 | 相对Random Forest算法AdaBoost充分考虑的每个分类器的权重 |
| **方案劣势** | 对于复杂问题，单个模型的精度往往比集成模型低 | 当决策树个数较多时，训练需要的时间和空间比较大 | 训练过程中两个模型要保持一致，否则可能到最后GAN网络跑不起来 | 训练比较耗时，每次重新选择当前分类器最好切分点 |

表2-2 技术方案对比

## 三 减少训练样本的方法

### 3.1 背景

复杂的软件系统通常提供大量的参数，供用户在特定的应用场景配置它们。然而，配置调优需要对软件系统有深入的了解。为了解决这个问题，许多现有方法都侧重于探索和学习特定配置下的性能来构建性能评估模型。但是当训练样本的数量很少时，模型的准确性往往会受到很大影响，尤其是在工业领域更注重成本，长时间占用集群收集训练样本往往会带来巨大的费用，目前这是一个棘手的问题。

目前我们调研关于减少训练样本的方法主要包括使用GAN生成部分训练数据、使用投影抽样估算预测模型需要的最佳样本量以及构建测试平台用小数据集上的配置调优替代大数据的配置调优。

### 3.2 学术界最新进展

#### 3.2.1 ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks

该论文发表在ASE2019，作者为解决收集训练样本耗时长的问题提出了一个假设，**假设好的配置通常都有某些隐藏的结构**。因此该文章不是试图改进给定配置的性能模型，而是关注于捕获好的配置的隐藏结构，并**使用GAN学习这种结构生成潜在的更好的配置。**

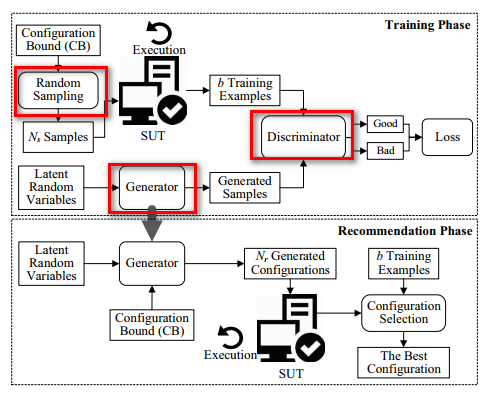


图3-1 ACTGAN方法流程图

**配置调优**。首先通过**随机抽样Ns个样本**，在集群上运行这些样本并取出性能较好的b个子集作为GAN的训练集。使用训练样本训练GAN的生成器G和判别器D，获得GAN网络后，**利用生成器G生成Nr个新的配置**，运行这些配置得到实际的执行时间，**从Ns和Nr个配置中选择性能最好的配置作为最终的调优配置。**

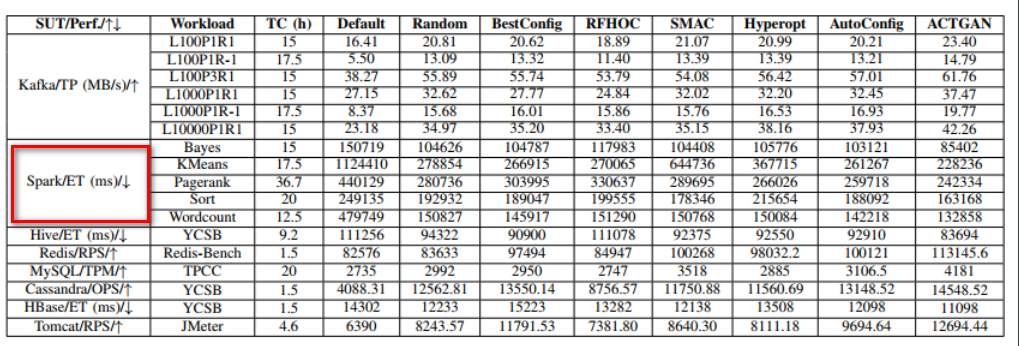


表3-1 不同算法在固定时间约束下的性能结果

如表3-2所示ACTGAN比**默认配置**平均提高了**76.22%**，且优于所有其他六种算法:比**Random**提高**9.28%-53.99%**，比**BestConfig**提高**7.21%-27.45%**，比**RFHOC**提高**9.52%-38.44%**，比**SMAC**提高**10.41%-49.58%**，比**Hyperopt**提高**9.66%-64.56%**，比**AutoConfig**提高**6.58% - 34.59%。**

#### 3.2.2 GML: Efficiently Auto-Tuning Flink's Configurations Via Guided Machine Learning

该论文发表在TPDS2021，提出了一种GML引导机器学习方法来调整Flink的配置，与传统的机器学习方法相比，该方法收集训练数据的时间明显更短。该方法使**用生成式对抗网络(Generative Adversarial Networks, GAN)生成性能高于随机配置平均性能的训练数据，减少收集训练数据所需的时间。随后，使用改进的遗传算法搜索最优配置。**

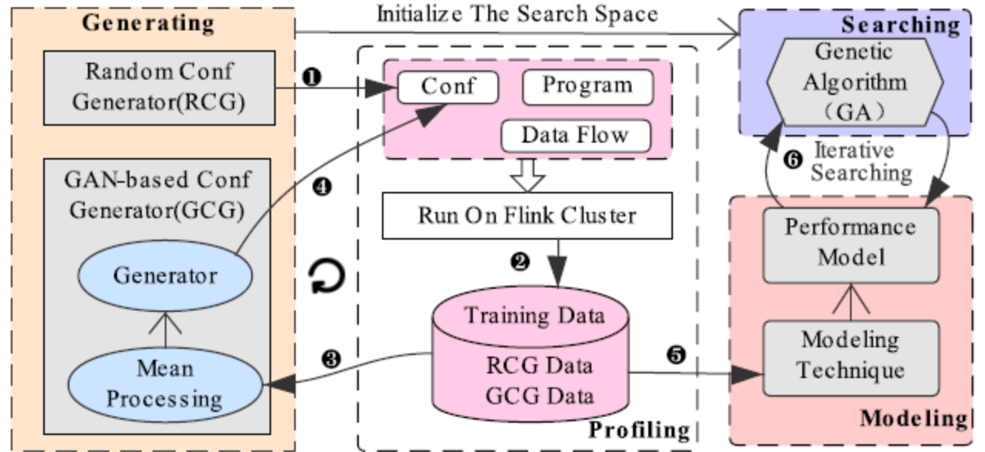


图3-2 GML方法流程图

**配置生成器**。**随机生成k个配置**，在Flink集群上运行配置记录性能（吞吐量/延迟），**计算这k个配置的平均性能，将性能高于平均性能的配置作为GAN的输入，训练GAN网络，**GAN学习性能较好的配置从而**生成更好的p个配置**，将这p个配置放入集群实际运行，记录性能，重复以上过程直到收集足够的训练数据。

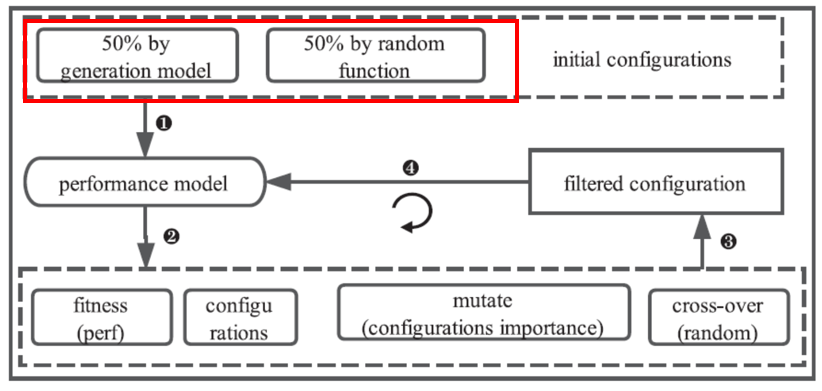


图3-3 遗传算法流程图

**构建性能预测模型（SGBRT模型）**。使用SGBRT构建性能预测模型，训练数据的组成为**50%随机生成，50%由GAN生成**。将配置输入性能模型，输出该配置的预测执行时间。在构建模型同时量化配置参数对性能的重要性。

**配置调优**。如图3-3所示**遗传算法的初始种群由50%GAN生成，另外50%随机生成。**在遗传算法搜索过程中，利用配置参数的重要性来指导遗传算法的变异操作。**将配置参数的重要性作为变异概率**，引导遗传算法只对重要的配置参数进行变异。避免了对性能影响不大的参数进行变异操作，缩短了寻找最优配置参数值的时间。

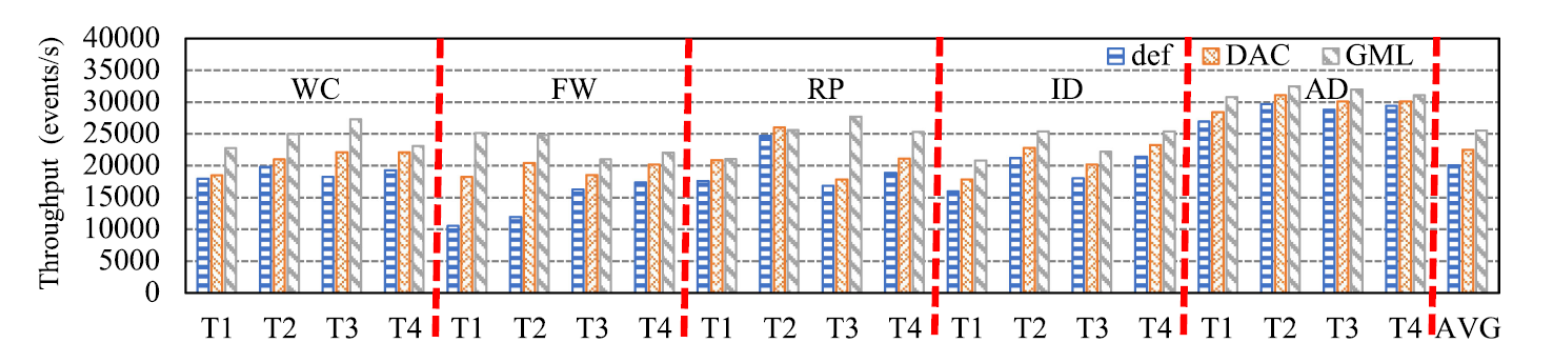


图3-4 GML与默认配置、DAC方法对比，性能（吞吐量）提升

如图3-4所示对于性能1（**吞吐量**），与默认配置相比GML可以提高**吞吐量，平均提高1.3倍** ，最高提高2.4倍。GML的平均表现**比DAC高出1.13倍**。

****

图3-5 GML与默认配置、DAC方法对比，性能（延迟）提升

如图3-5所示对于性能2（**延迟**），与默认配置相比**GML的平均和最大改进分别是1.7倍 和2.3 倍** 。GML的平均表现**比DAC高出1.3倍**。

#### 3.2.3 Efficient Performance Prediction for Apache Spark

该论文发表在J. Parallel Distrib. Comput.2021，提出了一种**以投影抽样为基础的迭代建模方法，采用的模型为Adaboost。**它是一种细粒度的方法，将于task,stage相关的指标考虑进来协助建立预测模型。

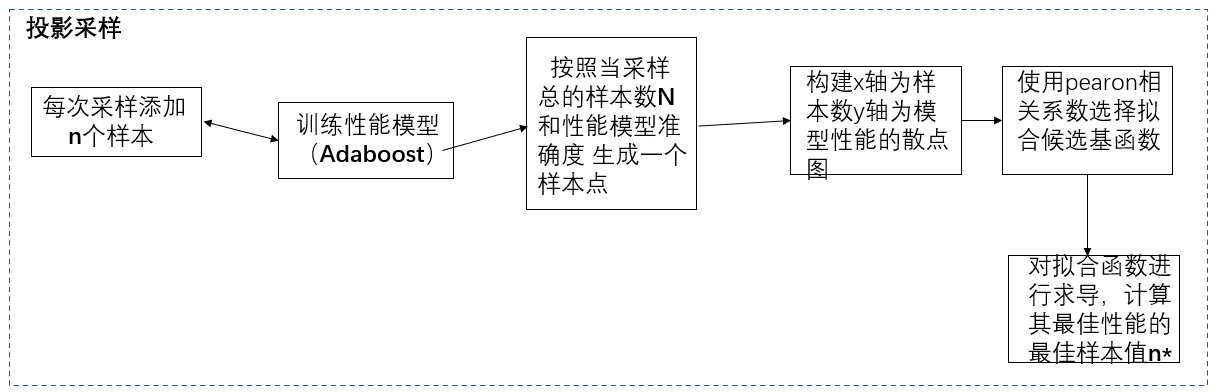


图3-6 方法流程图

**构建性能预测模型（Adaboost）**。采用一种迭代建模的思想，每轮采用**随机抽样的方法采N个样本**，并在样本的基础上**构建Adaboost预测模型，统计模型精度**，绘制样本数和模型精度的图像，并**用一些基函数进行拟合，将MSE误差最小的基函数作为图像的拟合曲线。利用最优的拟合曲线来计算最优抽样数目。**

与传统的全随机抽样方法相比，它可以用少量的训练样本达到更高的样本精度。与渐进抽样相比，增加了一个拟合基函数预测最优抽样数目。渐进抽样需要多执行几轮抽样确定最优模型所需的样本数，而渐进抽样可以进行早停，避免了多余的抽样样本的浪费。这无疑减少了更多的抽样成本，在大数据平台配置调优上无疑是很有价值的。

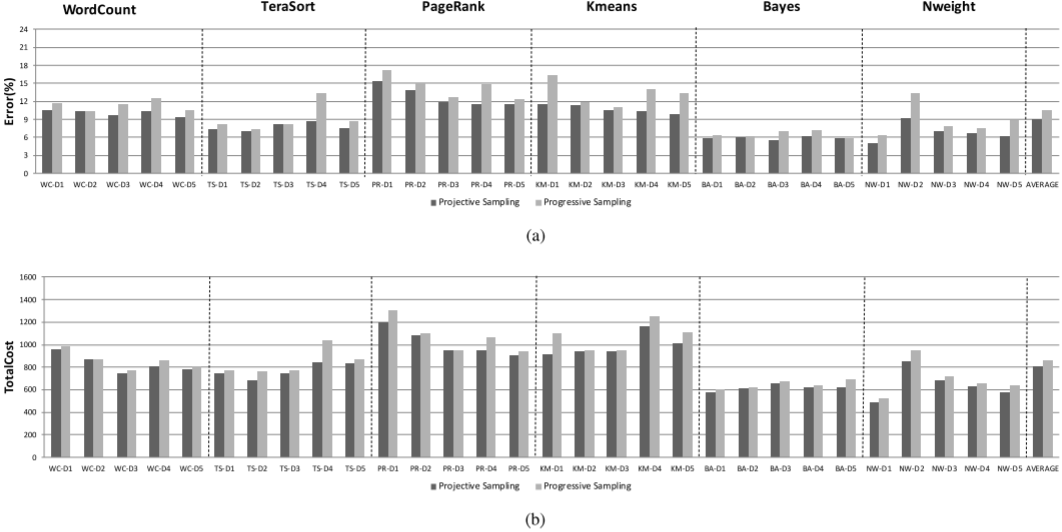


图3-7 使用投影抽样和渐近抽样构建模型的平均预测误差(a)和成本(b)

如图3-7所示不管是在**模型准确度**还是**模型花费**上，投影抽样都要比分层抽样在一定程度上优秀，如图3-7（a）所示投影抽样和分层抽样的**平均误差分别为9.02%和10.6%**，如图3-7（b）所示投影抽样和分层抽样的平**均消耗分别为815和867**。

#### 3.2.4 Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks

该论文发表在Big Data2018，提出一个自动参数调节系统。该系统首先同时**采用投影抽样的方法，将实际数据量、集群规模和缩减规模作为输入构建一个成本预测模型，来预测测试平台的测试数据量和测试集群规模。**

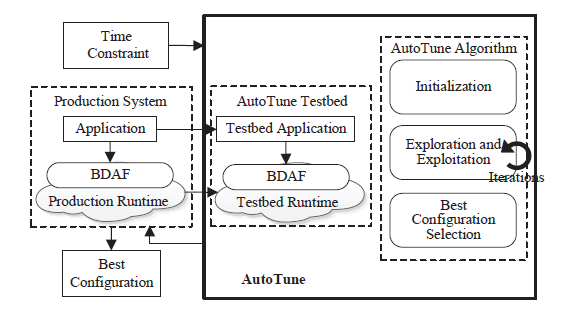


图3-8 AutoTune方法流程图

**构建性能预测模型**。在成本模型预测结果的基础上，该系统构建一个**小规模的测试平台**，减少应用的数据量和集群规模，**在同样的时间成本下产生更多的样本 ，构建出性能预测模型**。构建性能模型采用一种拉丁超立方抽样的方法，相比随机抽样能够在减少抽样样本数目的基础上，更好的覆盖样本的分布。

**配置调优**。在得到预测模型后，**该方法与GA搜索算法进行结合**，在该方法可以在牺牲一定精度的基础上，大幅度的减少抽样的成本。

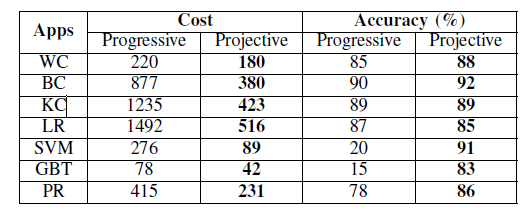


表3-2 渐近抽样和投影抽样的成本和精度

如表3-2所示**在成本和模型精确度方面，投影抽样都比分层抽样好**，同时不容易陷入到局部最优。‘

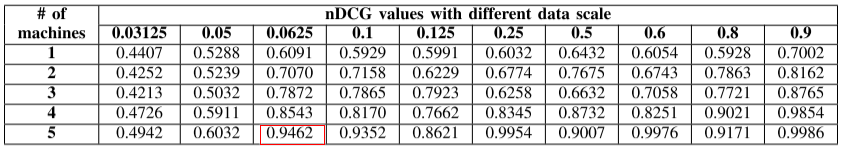


表3-3 不同试验台设置下K-MEANS应用的NDCG值(越大越好)

如表3-3所示对于不同的数据缩放系数。首先，如果实际环境和测试平台使用**同样数量的机器**，效果是最好的。其次，**中等规模**的数据大小要远好于小规模的数据大小。随着数据大小的增加，排名准确度并不会显著增加。因此**在保证同样的环境下，适当选用中等规模数据代替原大数据下的配置调优**。本实验便是采用1/16(0.0625)，和nm=5

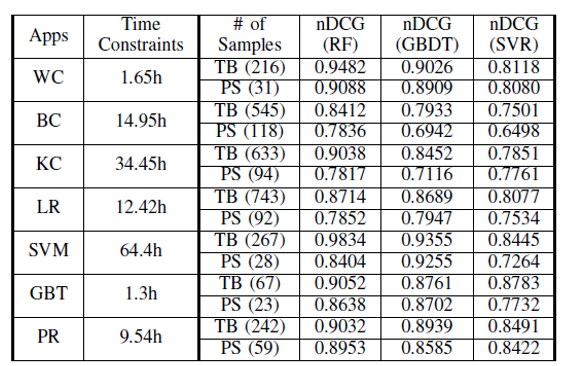


表3-4 三种不同的学习模型的NDCG值

如表3-4所示通过测试平台与原系统相比，由于时间的限制可以获得更多的抽样，**显著提高模型性能，平均提高7.11%**。同时随机森林要比其他两个模型要好。

### 3.3 技术方案对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | 方案一 | 方案二 | 方案三 |
| **技术方案介绍** | GAN | 投影抽样 | 小数据平台 |
| **主要适用场景** | 通过学习已有数据的隐藏结构来构造数据 | 搭配模型使用 | 用小数据量调优过程代替大数据下的调优 |
| **扩展性** | 可以搭配各种抽样来构建数据 | 可以搭配各种模型使用来估计最佳数据量 | 针对不同的大数据负载需构建不同的测试平台 |
| **成本** | 较低，构造的数据执行时间短且不容易产生错误数据 | 相比渐近抽样构建模型而言，成本更低 | 比直接使用模型和搜索算法的方法要低 |
| **方案优势** | 能够通过已有的样本生成更多样本，在保证性能模型精确度的同时减少收集训练样本的时间 | 相比目前的渐近抽样技术，建模时需要的样本量更少 | 可以根据小数量的调优来实现大数据下的调优 |
| **方案劣势** | 训练GAN网络的时候容易出现崩溃和跑不通的问题 | 如果学习曲线不准确，预测的最佳数据量就不准 | 需要提前有一定量的大数据平台下的数据 |

表3-5 技术方案对比

## 四 拟提出的技术方案

### 4.1 我们提出的技术方案

本场景下，我们将**采用GAN生成部分训练数据，迭代构建集成学习模型预测特定配置下的执行时间，使用贝叶斯优化算法搜索最优配置参数。**整体技术分为三部分，一是收集数据模块，二是探索重要参数和构建性能模型模块，三是调优配置模块。

图4-1为基本流程示意图。

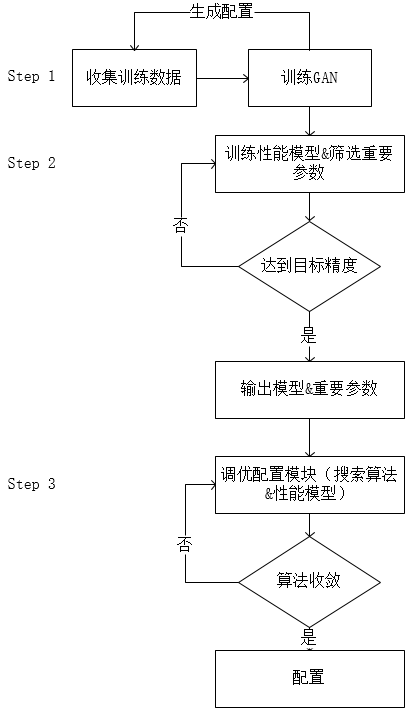


图4-1 基于GAN和贝叶斯优化方法的流程图

#### 4.1.1收集数据模块

##### 4.1.1.1主要功能

在配置的参数空间内采集样本供性能模型训练。

##### 4.1.1.2实现方案

以下是收集数据模块的实现步骤：

1. 使用拉丁超立方LHS抽样在参数空间内均匀的抽样样本点
2. 在集群上运行这些样本，记录样本的执行时间
3. 将已有的配置和执行时间传递给GAN做训练，利用GAN生成部分样本
4. 在集群上运行GAN生成的样本，记录样本的执行时间
5. 重复步骤4、5，直到收集到足够的训练数据。

#### 4.1.2选择重要参数和构建性能模型模块

##### 4.1.2.1主要功能

Spark有180多个配置参数，然而真正对系统性能产生影响的敏感参数只有十几个。需要调优的配置参数维度过大，会给模型训练和参数搜索过程带来巨大的成本消耗，因此通过保留对性能影响最大的配置参数，剔除对性能无影响的参数来加快调优过程。

1. 选出对性能最敏感的重要参数
2. 构建性能模型，输入一组配置参数，能够预测该配置的执行时间
3. 调优配置的过程中搜索产生的新配置不放入集群中实际运行，而是将配置传给性能模型，由模型输出执行时间

##### 4.1.2.2 实现方案

以下是构建性能模型模块的实现步骤：

1. 在配置参数值范围内，随机生成k个配置
2. 运行这些配置，记录相应的执行时间
3. 将每一组的配置和对应的执行时间制成一张表格
4. 将表格作为训练集训练GAN网络，并使用GAN网络生成p个配置，实际运行这些配置，并将配置和执行时间追加到表中
5. 选择5个集成学习模型，将表格输入模型做训练，取80%数据作为训练集，20%数据作为测试集，并计算该模型的预测精度，同时对参数进行排序，每次建模剔除2个最不重要的参数（即不将它们的值输入性能模型做训练，而是直接采用默认值）
6. 如果模型精度到达20%以下则将该模型选定为最终使用的性能预测模型，该模型留下的参数作为性能敏感参数，进行后续调优
7. 否则继续生成新的配置，重新迭代训练模型，直到精度达到要求

#### 4.1.3调优配置模块

##### 4.1.3.1主要功能

1. 使用搜索算法对配置参数进行调优
2. 直到算法收敛或超出时间限制时停止，当前运行时间最短的配置作为调优后的最优配置

##### 4.1.3.2 实现方案

以下是调优配置模块的实现步骤：

1. 在配置参数值范围内，随机生成k个配置
2. 运行这些配置，记录相应的执行时间
3. 将每一组的配置和对应的执行时间制成一张表格
4. 将表格作为训练集训练GAN网络，并使用GAN网络生成p个配置，实际运行这些配置，并将配置和执行时间追加到表中
5. 将以上样本作为贝叶斯优化算法的初始样本，通过贝叶斯优化找出下一个最有可能对性能有更大提升的样本
6. 算法收敛（执行时间不再大幅度减少）时，停止算法；否则重复第5步
7. 搜索的样本中执行时间最短的样本作为最终的调优样本

### 4.2 我们提出的技术方案

本场景下，针对目前最常用的渐近构建模型方法，我们将**采用投影抽样法计算模型最低点需要的样本数量，构建小数据测试平台评估实际生成环境的应用程序性能，从而减少收集训练样本的时间。**步骤包括构建测试平台和调优配置过程。

图4-2为基本流程示意图。

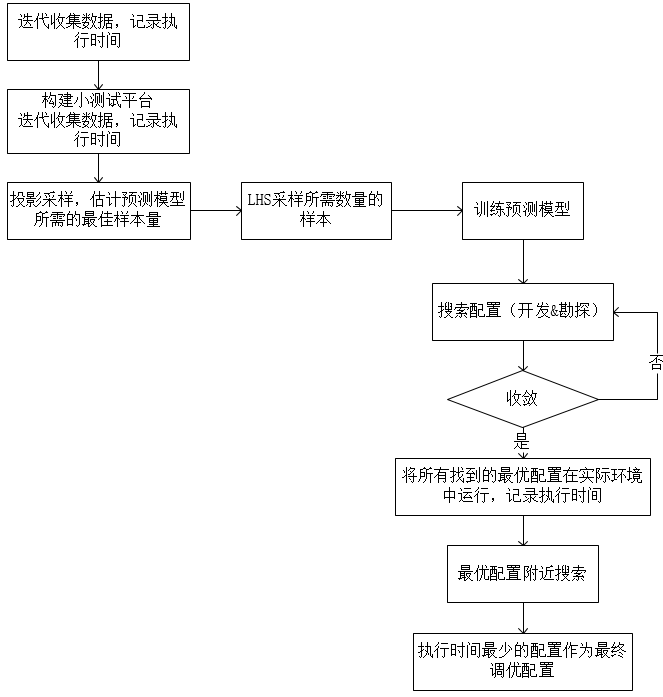


图4-2 构建测试平台方法的流程图

#### 4.2.1构建测试平台模块

##### 4.2.1.1主要功能

基于学习的调优中最关键的挑战是获取训练样本非常耗时。为了应对这一挑战，提出了一种基于测试平台的方法。目标是以准确且最快的速度评估实际环境中应用程序在不同配置上的性能。关键是减少处理的数据集的大小，使不同配置在测试平台上的相对性能尽可能接近生产系统上的性能。

##### 4.2.1.2实现方案

以下是构建测试平台模块的实现步骤：

1. 从空训练集开始，每次迭代增加k个样本，运行样本并获得性能值
2. 捕获应用程序的计算属性和应用程序中表达内部通信模式来构建预测模型（根据应用程序的数据规模和机器数量预测执行时间）
3. 估计学习曲线相关性，使用投影抽样来估计构建预测模型所需的样本数量
4. 一旦学习曲线的相关性为负数且最小，则确定该学习曲线作为最佳拟合函数，即可以计算构建预测模型所需的最佳样本数量

#### 4.2.2调优配置模块

##### 4.2.2.1主要功能

在时间限制下集成测试平台和生产系统来搜索更好的配置

##### 4.2.2.2实现方案

以下是调优配置模块的实现步骤：

1. 根据构建测试平台模块计算的最佳训练集大小，使用拉丁超立方抽样LHS生成初始训练集
2. 训练预测模型。尝试几种集成学习算法来训练模型，选择精度最低的模型
3. 勘探样本，使用LHS随机生成b个配置，通过遗传算法和预测模型在b个配置中搜索最优配置，并将最优配置加入到集合B中
4. 开发样本，在最优配置区间内开发另一组h个样本，在这个区间内寻找更优的样本，并将最优配置加入到集合B中
5. 将得到的b+h个样本传入模型中改进性能预测模型
6. 迭代进行步骤3~5直到收敛，即得到最配配置集合B
7. 将B集合的配置放到实际生产环境中运行，记录运行时间，选择时间最短的配置作为最优配置
8. 在该最优配置附近进行搜索，直到找到执行时间更短的配置

### 4.3 目前使用方法

目前，我们采用迭代建模的方法构建性能预测模型并筛选重要参数，将模型与遗传算法结合搜索最优配置。具体方案请参见《容器中大数据业务性能优化技术报告》。

图4-3为基本流程示意图。

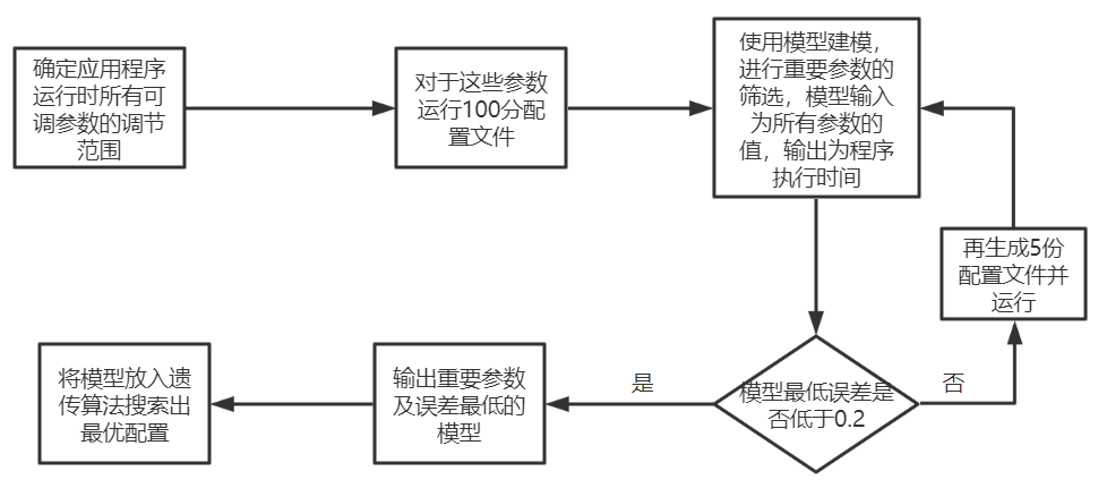


图4-3 迭代建模方法的流程图

#### 4.3.1 实现方案

1. 确定所有需要调节的参数的合理的取值范围，运行100组随机生成的配置文件
2. 模型建模，建模的过程中逐步筛选重要参数。输出误差最低点的模型，根据误差判断是否要增加模型训练的数据量，如果误差高于0.2则再增加5份数据后重新训练，如果误差低于0.2则直接输出误差最低点的模型和重要参数。
3. 将模型作为遗传算法的适应度函数，用遗传算法搜索出最优配置。

#### 4.3.2 挑选对性能敏感的query

TPCDS共有104个query，有的query对性能不敏感（使用不同的配置在负载上运行这些query，执行时间变化不大），有的query对性能敏感（配置改变时这类query的执行时间变化很大）。我们认为不敏感的query没有调优的必要，因为调优这类query对整体性能改进不大，我们选择调优性能敏感query，在有限的搜索时间内达到性能的大幅提升，特别是对于性能不敏感并且执行时间长的query，我们不把它放入考虑范围内，可以大大减少调优时间。

以下是挑选性能敏感query的步骤：

1. 随机生成30组配置放入104个query中，实际运行它们并记录执行时间
2. 使用统计量标准差衡量query的执行时间变化
3. 执行时间变化超过阈值则归为性能敏感的query；执行时间变化低于阈值则归为性能不敏感的query

#### 4.3.3 实验结果

在TPCDS-21G负载上，我们的方法相比默认配置而言，能够将运行时间提升为原来的2倍。



## 附件A

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 文献 | 获取网址 | 引用量 | 年份 | 等级 | 简称 |
| 1 | Ernest: Efficient performance prediction for large-scale advanced analytics | https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman | 415 | 2016 | A | NSDI |
| 2 | An End-to-End Automatic Cloud Database Tuning Systyem Using Deep Reinforcement Learning | https://dl.acm.org/doi/abs/10.1145/3299869.3300085?casa\_token=tOAXc6b\_UucAAAAA:QDnSyEx3Ov42G5Jw\_cxx9-J0RwknXKdxAcT9XiZ3XG9mJIr84REUgMLuHjUTLOzIHhy8Oa6q2f89 | 114 | 2019 | A | SIGMOD |
| 3 | RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop’s Configuration | https://ieeexplore.ieee.org/abstract/document/7132754/ | 66 | 2016 | A | TPDS |
| 4 | Distance-Based Sampling of Software Configuration Spaces | https://ieeexplore.ieee.org/abstract/document/8812049/ | 50 | 2019 | A | ICSE |
| 5 | Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing | https://dl.acm.org/doi/abs/10.1145/3173162.3173187?casa\_token=bjLIeOOR0MQAAAAA:pJqb8gEdASRUtDmGU0gjnR\_gjUAEikZ3kHw24mZihXCYL0hvBoIJK12PDX3HBOMWA-5yhASVI7PI | 46 | 2018 | A | ASPLOS |
| 6 | Black or White How to Develop an AutoTuner forMemory-based Analytics | https://dl.acm.org/doi/abs/10.1145/3318464.3380591 | 14 | 2020 | A | TPDS |
| 7 | CounterMiner: Mining Big Performance Data from Hardware Counters | https://ieeexplore.ieee.org/abstract/document/8574573/ | 10 | 2018 | A | MICRO |
| 8 | MIA: Metric Importance Analysis for Big Data Workload Characterization | https://ieeexplore.ieee.org/abstract/document/8057859/ | 6 | 2017 | A | TPDS |
| 9 | To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics | https://dl.acm.org/doi/abs/10.1145/3394486.3403299 | 4 | 2020 | A | SIGKDD |
| 10 | CTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks | https://ieeexplore.ieee.org/abstract/document/8952456/ | 4 | 2019 | A | ASE |
| 11 | GML Efficiently Auto-Tuning Flink’s Configurations Via Guided Machine Learning | https://ieeexplore.ieee.org/abstract/document/9435010/ | 1 | 2021 | A | TPDS |
| 12 | BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning\* | https://dl.acm.org/doi/abs/10.1145/3127479.3128605?casa\_token=StM1fEYdK7kAAAAA:Auo\_asdkHpYSzEzRMT9dighLV7gI-Bxo2-w5HtOM8WzXpQwFV0CschVOc\_cYsmfzDNpqhgshEVj\_ | 103 | 2017 | B | SOCC |
| 13 | Auto-tuning Spark Big Data Workloads on POWER8:Prediction-Based Dynamic SMT Threading | https://ieeexplore.ieee.org/abstract/document/7756772/ | 23 | 2016 | B | PACT |
| 14 | Machines Tuning Machines Configuring Distributed Stream Processors with Bayesian Optimization | https://ieeexplore.ieee.org/abstract/document/7307560/ | 19 | 2015 | B | CLUSTER |
| 15 | Cost-effective resource provi- sioning for Spark workloads | https://dl-acm-org.yorku.80599.net/doi/abs/10.1145/3357384.3358090 | 6 | 2019 | B | CIKM |
| 16 | Efficient Performance Prediction for Apache Spark | https://www.sciencedirect.com/science/article/pii/S0743731520303993 | 3 | 2021 | B | J. Parallel Distrib. Comput. |
| 17 | Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model | https://www.sciencedirect.com/science/article/pii/S0164121221001254 |  | 2021 | B | J Syst Softw |
| 18 | Performance Analysis and Auto-tuning for SPARK in memory analytics | https://ieeexplore.ieee.org/abstract/document/9474122/ |  | 2021 | B | DATE |
| 19 | A Frequency Scaling based Performance Indicator Framework for Big Data Systems | https://link.springer.com/chapter/10.1007/978-3-030-18576-3\_2 |  | 2019 | B | DASFAA |
| 20 | A Top-Down Method for Performance Analysis and Counters Architecture | https://ieeexplore-ieee-org.washington.80599.net/abstract/document/6844459/ | 235 | 2014 | C | ISPASS |
| 21 | A Novel Method for Tuning Configuration Parameters of Spark Based on Machine Learning | https://ieeexplore.ieee.org/abstract/document/7828429/ | 82 | 2016 | C | HPCC/SmartCity/DSS |
| 22 | Using machine learning to optimize parallelism in big data applications | https://www.sciencedirect.com/science/article/pii/S0167739X17314668 | 46 | 2018 | C | FGCS |
| 23 | Stage Aware Performance Modeling of DAG Based in Memory Analytic Platforms | https://ieeexplore.ieee.org/abstract/document/7820271/ | 39 | 2016 | C | CLOUD |
| 24 | Configuring In-memory Cluster Computing Using Random Forest | https://www.sciencedirect.com/science/article/pii/S0167739X16305167 | 24 | 2018 | C | FGCS |
| 25 | Learning-based automatic parameter tuning for big data analytics frameworks | https://ieeexplore.ieee.org/abstract/document/8622018/ | 17 | 2018 | C | IEEE Big Data |
| 26 | Towards Automatic Tuning of Apache Spark Configuration | https://ieeexplore.ieee.org/abstract/document/8457827/ | 17 | 2018 | C | CLOUD |
| 27 | DocLite: A Docker-Based Lightweight Cloud Benchmarking Tool | https://ieeexplore.ieee.org/abstract/document/7515691/ | 16 | 2016 | C | CCGrid |
| 28 | Understanding the Influence of Configuration Settings: An Execution Model-driven Framework for Apache Spark Platform | https://ieeexplore.ieee.org/abstract/document/8030677/ | 13 | 2017 | C | CLOUD |
| 29 | mrMoulder: A recommendation-based adaptive parameter tuning approach for big data | https://www.sciencedirect.com/science/article/pii/S0167739X17318526 | 11 | 2019 | C | Future Generation Computer Systems |
| 30 | Auto-tuning Spark Configurations Based on Neural Network | https://ieeexplore.ieee.org/abstract/document/8422658/ | 5 | 2018 | C | ICC |
| 31 | d-Simplexed: Adaptive Delaunay Triangulation for Performance Modeling and Prediction on Big Data Analytics | https://researchportal.helsinki.fi/files/134280616/Delaunay\_Triangulation\_Paper.pdf | 5 | 2019 | C | TBD |
| 32 | 基于运行数据分析的Spark任务参数优化 | http://www.cqvip.com/qk/94293x/201601/667685878.html | 3 | 2016 | C | 计算机工程与科学 |
| 33 | A Presentation and Retrieval Hash Scheme of Images Based On Principal Component Analysis | https://link.springer.80599.net/article/10.1007/s00371-020-01973-8 | 1 | 2021 | C | The Visual Computer |
| 34 | You Only Run Once: Spark Auto-Tuning From a Single Run | https://ieeexplore.ieee.org/abstract/document/9244226/ |  | 2020 | C | TNSM |
| 35 | Application-specifc feature selection and clustering approach with HPC system profling data | https://link.springer.com/article/10.1007/s11227-020-03533-2 |  | 2021 | C | TJSC |
| 36 | Optimal design of anti-interrupted sampling repeater jamming waveform for missile-borne | https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sil2.12066 |  | 2017 | C | IJESM |
| 37 | Genetic Learning Particle Swarm Optimization | https://ieeexplore.ieee.org/abstract/document/7271066/ | 304 | 2016 | SCI 1 | IEEE Transactions on Cybernetics |
| 38 | A Methodology for Spark Parameter Tuning | https://www.sciencedirect.com/science/article/pii/S2214579617300114 | 57 | 2018 | SCI 1 | Big data research |
| 39 | A Survey on Automatic Parameter Tuning for Big Data Processing Systems | https://dl.acm.org/doi/abs/10.1145/3381027 | 21 | 2020 | SCI 1 | ACM Computing Surveys |
| 40 | A Cluster-Principal-Component-Analysis-Based Indoor Positioning Algorithm | https://ieeexplore-ieee-org.washington.80599.net/abstract/document/9113271/ | 6 | 2020 | SCI 1 | IEEE Internet Things J. |
| 41 | Daily Activity Feature Selection in Smart Homes Based on Pearson Correlation Coefficient | https://link.springer.80599.net/content/pdf/10.1007/s11063-019-10185-8.pdf | 49 | 2020 | SCI 2 | Int. J. Electr. Power Energy Syst. |
| 42 | Initialization Methods for Large Scale Global Optimization | https://ieeexplore.ieee.org/abstract/document/6557902/ | 58 | 2013 | SCI 3 | IEEE CEC |
| 43 | ATCS: Auto-Tuning Configurations of Big Data Frameworks Based on Generative Adversarial Nets | https://ieeexplore.ieee.org/abstract/document/9031432/ | 6 | 2020` | SCI 3 | IEEE Access |
| 44 | Dione: Profiling spark applications exploiting graph similarity | https://ieeexplore-ieee-org.washington.80599.net/abstract/document/8257950/ | 4 | 2017 | SCI 3 | big data |
| 45 | MonkeyKing Adaptive Parameter Tuning on Big Data Platforms with Deep Reinforcement Learning | https://www.liebertpub.com/doi/abs/10.1089/big.2019.0123 | 1 | 2019 | SCI 3 | Big data |
| 46 | Gunther: Search-Based Auto-Tuning of MapReduce | https://link.springer.com/chapter/10.1007/978-3-642-40047-6\_42 | 88 | 2013 |  | Euro-Par |
| 47 | Measuring Docker Performance: What a Mess!! | https://dl.acm.org/doi/abs/10.1145/3053600.3053605?casa\_token=UwBeO0wr-GAAAAAA:9N8M6kI-ZhWc8JIWthLpcMtoC\_256HkoA4Crh9hD-a4TSpa\_zLzMeKvRsz6927z65vYIXkfaeRho | 71 | 2017 |  | ICPE |
| 48 | Spark Parameter Tuning via Trial-and-Error | https://link.springer.com/chapter/10.1007/978-3-319-47898-2\_24 | 44 | 2016 |  | INNS Conference on Big Data |
| 49 | Extending the roofline model: Bottleneck analysis with microarchitectural constraints | https://ieeexplore-ieee-org.washington.80599.net/abstract/document/6983061/ | 29 | 2014 |  | ISWC |
| 50 | SMBSP: A Self-Tuning Approach using Machine Learning to Improve Performance of Spark in Big Data Processing | https://ieeexplore.ieee.org/abstract/document/8539328/ | 4 | 2018 |  | ICCCE |
| 51 | Towards machine learning-based self-tuning of hadoop-spark system | https://www.researchgate.net/publication/339806427\_Towards\_machine\_learning-based\_self-tuning\_of\_Hadoop-Spark\_system | 2 | 2019 |  | IndonesiJECS |
| 52 | A Method to Identify Spark Important Parameters Based on Machine Learning | https://link.springer.com/chapter/10.1007/978-981-13-2203-7\_42 | 1 | 2018 |  | ICPCSEE |