doi:10.11887/j.cn.202405024 http://journal.nudt.edu.cn

执行时间预测驱动的工作流作业调度

胡亚红*,邱圆圆,毛家发
(浙江工业大学 计算机科学与技术学院,浙江 杭州 310023)

摘 要:针对工作流作业调度问题,提出使用关键路径法进行工作流的执行时间预测和资源分配。工作流执行时间预测算法使用并行应用有向无环图描述工作流中子作业的执行顺序。基于此顺序,为子作业进行系统资源的逻辑分配。根据子作业的特征和资源分配信息,使用梯度提升决策树进行子作业执行时间预测,并计算工作流的关键路径。关键路径上所有子作业的完成时间之和即为工作流的执行时间。若预测的工作流执行时间满足用户要求,则根据子作业执行顺序和资源分配方案进行作业调度,执行工作流。对比实验表明,两个工作流的执行时间预测误差分别为5.72%和1.57%。与Spark默认调度算法相比,工作流调度算法将两个工作流的完成时间分别缩短了15.71%和15.44%。

关键词:工作流;时间预测;关键路径;调度算法;梯度提升决策树
中图分类号:TP393 **文献标志码**:A **文章编号**:1001-2486(2024)05-228-11

论文拓展

Workflows scheduling powered by execution time prediction model

HU Yahong, QIU Yuanyuan, MAO Jiafa*
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: For the problem of workflow job scheduling, the critical path method was proposed to predict the execution time of the workflow and allocate resources. The parallel application directed acyclic graph was used to describe the relationships among the sub-jobs of a workflow in the workflow execution time prediction algorithm. Based on this order, the system resources were logically allocated to the sub-jobs. According to the characteristics and resource allocation information of sub-jobs, the gradient boosting decision tree-based algorithm was used to predict the execution time of sub-jobs, and the critical path of workflow was calculated. The sum of the completion time of all sub-jobs on the critical path is the execution time of the workflow. If the predicted workflow execution time satisfies the user's requirements, job scheduling was executed according to the sub-job execution sequence and resource allocation scheme, and the workflow was executed. Comparative experiments show that the prediction errors of the execution time of two workflows are 5.72% and 1.57%, respectively. Compared with the default scheduling algorithm of Spark, the workflow scheduling algorithm reduces the completion time of the two workflows by 15.71% and 15.44%, respectively.

Keywords: workflow; time prediction; critical path; scheduling algorithm; gradient boosting decision tree

工作流作业由多个具有控制约束或数据约束的子作业构成,广泛存在于工业生产和科学计算领域。对工作流子作业执行约束的分析,有助于对子作业进行合理调度,从而缩短工作流的执行时间。由于分布式系统能够同时执行多个作业,因而被认为是处理工作流作业的理想选择[1]。

作业调度属于NP-Complete问题,其最主要的目标是缩短作业的完成时间。分布式异构系统中的作业调度要完成的任务有两种,其一是为作业安排最优的执行顺序,其二是为每个作业分配最适合的系统资源以进行任务的执行[2]。由于作业调度的特点,目前提出的调度算法多是启发式算法。为了提高异构系统性能,降低调度算法的复杂度,Topcuoglu等提出了异构最早完成时间(heterogeneous earliest finish time, HEFT)算法,该算法在每一步为向上等级最高的作业分配处理器[3]。Zhou等提出了一种基于列表的改进预测最早完成时间的静态作业调度算法,算法使用悲观费用表对作业的优先级进行评价[4]。为了能够缩短用户作业的完成时间,同时最小化能量消

收稿日期:2022-05-21
基金项目:国家重点研发计划资助项目(2018YFB0204003)
*第一作者:胡亚红(1971—),女,陕西西安人,副教授,博士,硕士生导师,E-mail:huyahong@zjut.edu.cn
引用格式:胡亚红,邱圆圆,毛家发. 执行时间预测驱动的工作流作业调度[J]. 国防科技大学学报, 2024, 46(5): 228-238.
Citation: HU Y H, QIU Y Y, MAO J F. Workflows scheduling powered by execution time prediction model [J]. Journal of National University of Defense Technology, 2024, 46(5): 228-238.

耗,Zhang 等提出了基于模因算法的工作流调度算法[5]。该算法与同类算法相比,能够缩短 4.9% 的作业执行时间,能量则节省了 24.3%。Kumar 等提出使用混合模型的遗传算法作业调度方法以减少作业完成时间、提高分布式实时系统的可靠性[6]。在分析了基于队列的作业调度方法的不足后,Zheng 等提出了基于计划的模拟退火算法以进行作业调度[7]。与基于队列的调度方法相比,该方法能够缩短 40% 的作业等待时间,减少 30% 的作业响应时间。Li 等采用博弈论研究了并行环境下简单线性退化任务的调度问题[8]。对于 Spark 应用,Fu 等采用二部图模型完成了位置感知的任务调度算法,有效减少了数据的网络传输和访问延迟[9]。高性能计算领域发展极快,高性能计算的应用已不仅为计算密集型,数据密集型应用也越来越多。Fan 等提出了多目标优化遗传算法,综合考虑系统 CPU、burstbuffer 等多种资源的利用情况[10]。随着人工智能技术的发展,越来越多的机器学习算法被应用于任务的调度。Fan 等搭建了一个层次化的神经网络,使用深度强化学习的方法让任务调度代理完成对目标环境的学习,从而做出满足优化目标的调度方案。实验表明,论文中提出的算法比传统的启发式优化算法性能提升 45%[11]。

工作流一般使用有向无环图(directed acyclic graph, DAG)进行描述和分析[12]。Adhikari 等详细介绍了使用 DAG 对工作流进行建模的方法[13]。Ilavarasan 等提出的高效能任务调度算法(performance effective task scheduling, PETS)使用 DAG 对作业进行描述[14]。文中进行分析和实验使用的 DAG 一部分是随机生成的,另一部分则是根据实际应用产生的。Sulaiman 等提出的使用复制方案的基于混合列表的任务调度(hybrid list-based task scheduling using duplication scheme, HLTSD)算法同样使用 DAG 作为作业表示的方法,算法能够得到最小费用的调度方案[15]。Martinez 等使用 DAG 描述作业之间的串并行关系。提出使用特征矩阵以描述作业的信息,包括 DAG 中路径的数目、DAG 的层数及每层中节点的数目、相邻层之间的边的条数等。DAG 的第一条路径为关键路径,对它优先进行资源分配[16]。

准确进行作业执行时间预测能够为作业调度和资源分配提供依据。目前已有一些作业执行时间预测的研究工作。Yu 等详细分析了 Map/Reduce 模式下,Map 阶段和 Reduce 阶段具体任务的执行时间划分[17]。在每个核仅能执行特定类型任务的情况下,Han 等分析了多核异构环境中并行 DAG 任务的最差响应时间(worst-case response time, WCRT),给出了两种 WCRT 的分析方法和有效的实现算法,提高了 WCRT 的计算精度[18]。针对包含有加速元件,如现场可编程门阵列(field programmable gate array, FPGA)、图形处理器(graphic processing unit, GPU)的并行异构硬件结构,Serrano 等研究了 DAG 任务的响应时间分析[19]。为了提高在加速部件上所运行任务和在通用多核上所运行任务的并行程度,文献[19]提出了 DAG 的转换算法。基于同构系统中作业响应时间分析和 DAG 转换算法,进行了包含加速元件的异构系统中 DAG 任务的响应时间分析。Gao 等设计了一个具有较高预测精度的两步预测框架[20]。预测模型采用背景梯度提升回归。具有相似性能表现的作业使用相同的性能预测模型。模型通过匹配资源利用特征和历史应用来进行执行时间的预测。如何在众多的时间预测方法中选择一种有效可行的预测方法,成为作业执行时间预测中首先要解决的问题。Kumbhare 等建立了线性回归模型进行作业运行时间预测,并使用随机森林提供模型在不同输入数据时预测时间的转化系数[21]。很多研究工作通过减少高估作业运行时间来提高运行时间预测的准确性,Fan 等则认为过低估计作业执行时间同样会引起很多问题。为解决低估作业执行时间的问题,他们提出了在线运行时间调整框架 Tobit 运行时间预测(Tobit runtime prediction, TRIP)[22]。

目前有关工作流作业执行时间预测领域的研究基本假定子作业的执行时间是已知的。本文的创新工作体现在将子作业执行时间预测与工作流执行时间预测及子作业调度有机结合在一起。即通过分析工作流中各子作业的依赖关系,确定子作业的执行顺序,并依此进行系统资源分配。在此基础上,完成子作业的执行时间预测。根据各子作业的执行时间,计算出工作流中的关键路径,从而预测出工作流作业的执行时间。

考虑这样一个应用场景:一家企业拥有自己的小型集群,并经常需要处理复杂的工作流作业。为了确保每一个用户的工作流能够在指定截止时间前完成,需要预测出工作流作业的执行时间。如果预测出的执行时间不能满足用户的要求,则需要及时将预测结果告知用户,方便用户选择是将作业转移到其他集群去执行,还是延长作业的截止时间。这样及时有效的信息沟通,能够提升

用户的满意度。若预测出集群可以在用户期望的时间内完成作业,则按照进行时间预测时生成的最优资源分配方案进行资源的分配、作业的调度和执行。最优资源分配方案的使用可以有效缩短工作流的完成时间。

本文提出了基于子作业执行时间预测的工作流作业调度算法。在分析各个子作业间的执行约束后,为子作业进行资源的逻辑分配。根据资源分配方案使用梯度提升决策树(gradient boosting decision tree, GBDT)算法预测出子作业的执行时间后,计算工作流执行的关键路径,进而得到工作流作业的执行时间。对比正向和逆向资源分配方案下预测的工作流执行时间,按照时间较短的方案进行作业流中各子作业的资源物理分配,并运行工作流。

1 基于关键路径分析的工作流作业时间预测与调度算法

为了预测工作流作业的运行时间并进行工作流中各个子作业的合理调度,提出了基于关键路径分析的工作流作业时间预测与调度算法。算法流程如图1所示。

此算法的基本思想是分析工作流作业中各个子作业的约束关系,生成对应的有向无环图。为了强调本算法对子作业执行并行度的提升目标,工作流作业对应的有向无环图称为并行应用有向无环图(parallel application DAG,PAD)。算法通过广度优先搜索(breadth first search, BFS)找到子作业执行的串并行关系。将图中不同顶点作为广度优先搜索的起始点,会得到不同的搜索结果。结合PAD的特点,本算法采用了正向和逆向的两种搜索,即从第一个子任务或最后一个子任务开始进行广度优先搜索,得到两组子作业执行顺序的串并行关系。

根据子作业之间的关系,为各子作业进行集群资源的分配。将子作业的信息和资源分配信息作为单作业时间预测模型的输入,预测出各子作业的执行时间。根据子作业的执行时间和相互关系,找到PAD中的关键路径。此关键路径上各个子作业完成时间之和即为工作流作业的预计完成时间。

可以看出,进行单作业的执行时间预测是算法的核心,因此首先介绍这个算法。

2 单作业的时间预测算法

机器学习中常用的回归预测算法包括梯度提升(gradient boost)、线性回归(linear regression)[23]、决策树(decision tree)[24]、支持向量机(support vector machine, SVM)[25]、随机森林(random forest, RF)[26]和梯度提升决策树等。

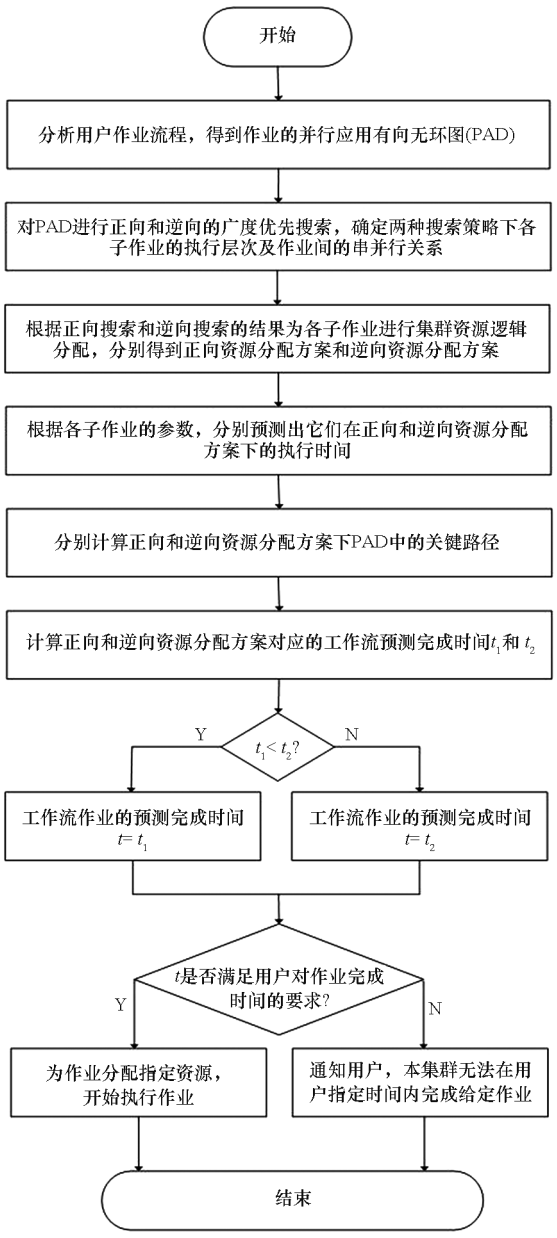

图1 基于关键路径分析的工作流作业时间预测与调度算法

Fig. 1 Critical path analysis-based workflow execution time prediction and scheduling algorithm

GBDT可以用于回归、分类、排序等各种任务,具有模型训练要求的数据量小、预测精度高、调参时间短、可解释性强、能够有效处理低维数据和非线性数据等优点,广泛应用于各个领域的预测,例如心脏疾病的预测[27]、短期电量负载的预测[28]、焊点质量预测[29]、遥控操作中的力反馈和对象位置[30]预测、金融指数数据预测[31]、道路交

通拥堵情况预测[32-33]、作业执行时间预测等。文献[29]通过实验对比了 GBDT 与其他预测算法的性能。实验结果表明,与 K 最近邻算法(K-nearest neighbors, KNN)、SVM、决策树、朴素贝叶斯和 RF 等算法相比,GBDT 的 F1 值最高。由于数据量的限制,文献[31]无法使用神经网络等需要大量训练集的预测方式。在实验对比中,文献[31]提出的基于 GBDT 的金融指数预测模型的预测效果优于 SVM 和 RF。文献[32]指出使用 GBDT 预测道路交通拥堵的准确率能够达到 94.46%。文献[33]得到的每日交通指数的预测准确度也超过 90%。

本文的训练集较小,同时对作业运行时间进行预测的实时性要求高,因此选择 GBDT 作为时间预测算法。

2.1　GBDT 算法描述

GBDT 是 gradient boost 和 decision tree 两种算法的融合,它通过多轮迭代不断产生弱学习器,每个学习器在上一轮学习器的残差基础上进行训练,通过减低残差不断提高学习器的精度。

GBDT 使用的损失函数 $L(y_i,f(x_i))$ 是平方损失,使用式(1)计算得到。

$$L(y_i,f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2 \tag{1}$$

式中,y_i 表示样本的输出,$f(x_i)$ 表示学习器。

$$y_i - f(x_i) = -\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)} \tag{2}$$

式中,$y_i - f(x_i)$ 表示残差。

GBDT 算法的描述见算法 1。将使用 GBDT 算法预测工作流中各个子作业的执行时间,进而计算出工作流的执行时间。

2.2　基于 GBDT 的时间预测模型

进行作业执行时间预测的目标是提高集群的资源使用率并最小化作业执行时间,从而提高集群性能。图 2 是基于 GBDT 的时间预测模型的总体架构。

构建用户作业执行时间预测模型包括以下步骤:

1)数据收集。分析作业的整体执行流程,结合已有的研究成果确定对作业性能影响较大的参数。根据这些作业参数,生成符合要求的作业并在集群中执行,收集作业执行时间等数据。

2)模型训练。使用收集的数据样本进行 GBDT 模型训练,得到作业执行时间的预测模型。

3)作业执行时间预测。当有新的作业进入集群时,利用作业执行时间预测模型得到相应的

算法 1　GBDT 算法描述

Alg. 1　Description of GBDT

输入:损失函数 L,最大迭代次数 M,样本总数 N,训练集样本 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$。

输出:强学习器 $f(x)$。

Step1:初始化弱学习器 $f_0(x)$。

$$f_0(x) = \underset{c}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i,c)$$

其中,c 是使损失函数极小化的参数值。

Step2:对迭代轮数 $m=1,2,\cdots,M$ 执行:

(a)对每个样本 $i=1,2,\cdots,N$,使用下式计算负梯度。

$$r_{mi} = -\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

其中,r_{mi} 表示第 m 轮的第 i 个样本的损失函数的负梯度,$f_{m-1}(x)$ 是第 $m-1$ 轮得到的回归树。

(b)将上步得到的残差作为样本新的真实值,并将数据 $(x_i,r_{mi})(i=1,2,\cdots,N)$ 作为下棵树的训练数据,得到第 m 棵回归树,即弱学习器 $f_m(x)$。其对应的叶子节点区域为 $R_{mj}(j=1,2,\cdots,J)$,J 为每棵回归树的叶子节点的个数。

(c)对叶子节点区域计算最佳拟合值。

$$c_{mj} = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} L(y_i,f_{m-1}(x_i) + c)$$

(d)更新强学习器。

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I$$

其中,I 表示学习率(learning rate),$x \in R_{mj}$。

Step3:得到最终的强学习器 $f(x)$。

$$f(x) = f_M(x) = f_0(x) + \sum_{m=1}^{M}\sum_{j=1}^{J} c_{mj} I$$

其中,$x \in R_{mj}$。

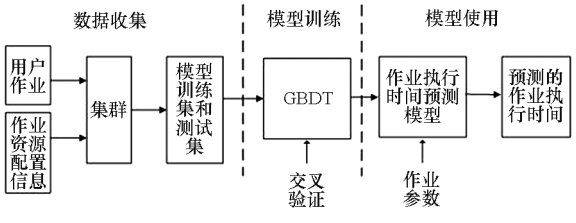

图 2　基于 GBDT 的时间预测模型

Fig. 2　GBDT based time prediction model

预测执行时间。在此过程中,将模型给出的配置作为作业的新配置参数,在集群中运行作业,记录结果,并将此次结果放入数据集中,以便进一步优化时间预测模型。

2.3　GBDT 模型的数据收集和训练

GBDT 模型能够根据当前集群状态及作业的信息完成作业执行时间预测。下面介绍模型输入

参数的选择、数据收集及模型训练。

模型参数的选择。Spark 共有 180 多个配置参数,但对作业处理性能影响较大的参数为数不多。本文共选取了 6 个参数作为 GBDT 模型的输入参数,如表 1 所示。其中,前 4 个参数是 Spark 的配置参数,inputsize 表示作业的输入数据大小,nodes 表示作业使用的集群节点数。

表 1 Spark 作业参数

Tab. 1 Spark job parameters

参数	描述	取值建议	实例
Spark. total. executor. cores	集群中所有 Executor 的 CPU 核数		10
Spark. cores. max	一个作业最多可以申请的 CPU 核数		4
Spark. executor. memory	每个 Executor 可用的内存	4 ~ 8 GB	4 GB
Spark. executor. cores	每个 Executor 的 CPU 核数,决定了每个 Executor 能够并行执行的 task 个数	2 ~ 4	2
inputsize	作业的数据量		10 GB
nodes	作业使用的节点个数		3

GBDT 模型的输入参数为表 1 中的 6 个参数,输出是作业在 Spark 集群中的运行时间。数据采集所用的作业为 BigDataBench[34] 的 WordCount 和 Sort 负载。数据样本的输入采用排列组合的方式确定,表 1 给出了一个输入参数组合。

确定输入参数后,将符合参数条件的作业提交到 Spark 集群,集群按输入参数的要求进行配置。每个输入组合在集群中运行 10 次,如果 10 次运行结果的误差在合理的范围内,则保留这组数据,否则将该数据舍弃。经过实验,最终确定了 1 000组有效输入参数,结合对应的作业运行时间,共得到 1 000 条样本数据。这 1 000 组数据用于 GBDT 模型的训练和测试,其中 80% 为训练集,20% 为测试集。训练完成后,使用测试集进行 GBDT 模型测试。

3 工作流作业执行时间预测算法

在预测得到工作流中每一个子作业的运行时间后,需要对工作流中子作业的相互关系进行分析,进而完成工作流的执行时间预测。建立图的模型来表示工作流的内部结构。

3.1 图模型的建立

为了处理含有内部数据依赖关系的工作流作业,建立了 PAD 模型。图中的节点 a_i 表示子作业 i 完成的事件,节点 a_0 是定义的一个空作业节点,表示整个工作流作业的开始事件,它没有前趋。图中的最后一个节点表示最后一个子作业的完成事件,它没有后继节点。PAD 中的有向边 $<a_i, a_j>$ 表示子作业 i 是子作业 j 的前趋作业,仅当子作业 i 执行结束,子作业 j 才可以开始运行。有向边上的权值 T_j 表示子作业 j 的执行时间,此时间由 GBDT 时间预测模型预测得到。

以“学生学业等级分析评价”工作流为例。此工作流可以分解为 6 个子作业,分别为:子作业 1“学生学业数据预处理”,子作业 2“根据课程考试成绩计算绩点”,子作业 3“计算学生课外科技竞赛加分”,子作业 4“计算学生参加志愿者等活动加分”,子作业 5“学生学习成绩汇总”和子作业 6“学生学业情况分析,给出学生学业等级”。各子作业间存在的数据依赖关系如表 2 所示。其中的子作业 0 是为了构建图模型增加的空作业,没有执行时间。此工作流对应的 PAD 如图 3 所示。

表 2 工作流作业中各子作业的依赖关系

Tab. 2 Dependency among the sub jobs of a workflow

子作业	前趋作业	预测的任务执行时间
0	无	0
1	0	T_1
2	1	T_2
3	1	T_3
4	1	T_4
5	2,3	T_5
6	4,5	T_6

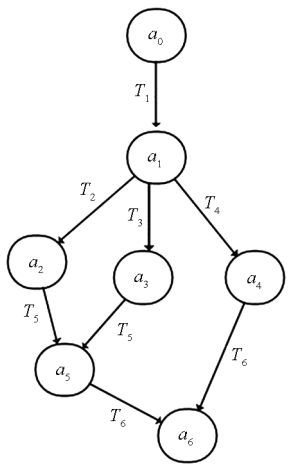

图 3 工作流作业 PAD

Fig. 3 PAD of the workflow

3.2　子作业执行顺序的搜索

进行作业执行时间预测首先需要知道集群系统为作业分配的资源数量。因为工作流作业中各个子作业有执行顺序的约束,所以必须先查找出子作业可能的执行顺序,尤其是确定可以并行执行的子作业。本文使用 BFS 完成对工作流作业 PAD 的搜索。针对工作流作业的特点,进行了正向搜索和逆向搜索。

1)正向搜索:将 PAD 中节点 a_0 作为起始点,运行 BFS,确定每一个子作业 i 所在的层次,即从 a_0 到 a_i 的路径的长度。

2)逆向搜索:将 PAD 中最后一个节点作为起始点,运行 BFS,确定每一个子作业 i 所在的层次。

表 3 给出了对图 3 中 PAD 分别进行正向搜索和逆向搜索得到的子作业层次关系。同一层次的子作业之间没有数据依赖关系,可以并行执行。因此根据正向和逆向搜索的结果,可以得到正向和逆向两组子作业串并行执行关系。

表 3　图 3 中工作流各子作业的正向和逆向串并行执行关系

Tab. 3　Forward and reverse series-parallel execution relationship of each sub-job in the workflow in Fig. 3

节点位置	正向搜索	逆向搜索
第 0 层	a_0	a_6
第 1 层	a_1	a_4, a_5
第 2 层	a_2, a_3, a_4	a_2, a_3
第 3 层	a_5	a_1
第 4 层	a_6	a_0

3.3　子作业逻辑资源的分配

当子作业的执行顺序确定后,就可以为它们进行集群资源的逻辑分配,方法见式(3)和式(4)。针对目前设定的应用场景,一个工作流可以使用集群中全部的资源。

$$C_{\mathrm{pu}i} = floor\left(\frac{T_{\mathrm{otalCPU}}}{W_i}\right) \tag{3}$$

$$M_{\mathrm{emory}i} = floor\left(\frac{T_{\mathrm{otalMemory}}}{W_i}\right) \tag{4}$$

为每个子作业分配的 CPU 核数 $C_{\mathrm{pu}i}$ 和内存数目 $M_{\mathrm{emory}i}$ 是进行作业执行时间预测重要的输入参数。式(3)和式(4)中:T_{otalCPU} 和 $T_{\mathrm{otalMemory}}$ 是集群中 CPU 的总核数和总的内存容量;W_i 是子作业 i 所在层次的子作业的数目,即可以并行执行的子作业的数目,同一层次中的子作业平分集群中的 CPU 和内存资源;$floor(\ )$ 为向下取整函数。

针对正向和逆向两组子作业串并行执行关系,进行资源逻辑分配后,可以得到正向资源分配方案和逆向资源分配方案。

3.4　工作流作业关键路径的确定

得到子作业的资源分配方案后,使用时间预测算法可以得到每一个子作业的执行时间。基于工作流的 PAD,使用关键路径算法,就可以找到 PAD 中的关键路径。工作流中关键路径上各子作业是工作流中最为重要的作业,任一关键作业的延期都会导致整个工作流无法按时完成。关键路径上所有子作业完成时间之和即为工作流的执行时间。

$$est(a_i, R_{\mathrm{s}}) = \max\{est(a_k, R_{\mathrm{s}}) + T_k\} \tag{5}$$

$$lst(a_i, R_{\mathrm{s}}) = \min\{lst(a_h, R_{\mathrm{s}}) - T_h\} \tag{6}$$

其中:$est(a_i, R_{\mathrm{s}})$ 是资源分配方案 R_{s} 下事件 a_i 的最早发生时间,作业 k 是作业 i 的前趋作业,T_k 是子作业 k 在 R_{s} 下的预测执行时间,可以知道 $est(a_0, R_{\mathrm{s}}) = 0$。$lst(a_i, R_{\mathrm{s}})$ 是 R_{s} 下事件 a_i 的最晚发生时间,作业 h 是作业 i 的后继作业,PAD 中最后一个事件的最晚发生时间与最早发生时间一致,即 $lst(a_{\mathrm{final}}, R_{\mathrm{s}}) = est(a_{\mathrm{final}}, R_{\mathrm{s}})$。

根据式(5)和式(6)可以计算出工作流 PAD 中每一个事件的最早发生时间 est 和最晚发生时间 lst。对于事件 a_i,若有 $lst(a_i, R_{\mathrm{s}}) = est(a_i, R_{\mathrm{s}})$,则事件 a_i 为关键事件,子作业 i 为关键作业。所有的关键事件,构成 PAD 的关键路径。一个 PAD 中可能存在多条关键路径,但是关键路径上所有事件对应的子作业的执行时间之和是一样的。这个时间即为工作流作业在此资源分配方案 R_{s} 下的预测执行时间。

3.5　工作流作业的执行

在正向资源分配方案和逆向资源分配方案下可以分别预测得到工作流作业的执行时间,较小的预测时间作为工作流的预测执行时间。判断工作流的预测执行时间是否能够满足用户作业截止时间要求。若可以,则按照对应的资源分配方案进行资源的物理分配,开始工作流作业的执行。

3.6　算法复杂度分析

当作业时间预测模型 GBDT 训练完成后,输入作业参数就可以得到预测的作业运行时间。因此基于关键路径分析的工作流作业执行时间预测算法的复杂度决定于广度优先搜索算法和关键路径的求取算法。

假设工作流作业中包含 n 个子作业,其对应的 PAD 中有 m 条边时,确定各子作业所处层次的广度优先搜索算法的时间复杂度为 $O(n+m)$,寻找关键路径的算法的时间复杂度也为 $O(n+m)$。因此本文算法的时间复杂度为 $O(n+m)$。通常一个工作流中包含的子作业数量不会很多,子作业间相互具有的关系数量也不会很高,即 n 和 m 的数量级都不太大。因此相对于作业的运行时间,本文算法的执行时间数量级较低,执行代价很低。

4 实验及结果分析

本节通过实验进一步说明基于关键路径分析的工作流作业执行时间预测与调度算法的使用和性能。

4.1 实验环境

本实验构建了一个基于 Spark Standalone 模式的异构集群,集群包含 6 个节点(1 个主节点、5 个从节点),集群节点的系统配置如表 4 所示。

表 4 集群节点硬件配置

Tab. 4 Hardware configuration for the cluster nodes

节点	CPU 核数	内存/GB	磁盘容量/GB
主节点	2	5	60
从节点 1	2	4	40
从节点 2	1	2	40
从节点 3	1	2	40
从节点 4	2	1	40
从节点 5	2	2	50

4.2 实验作业介绍

本小节进行了两个工作流的实验。工作流 1 包含 6 个 WordCount 子作业,子作业的执行顺序依赖关系如表 2 和图 3 所描述,其数据量见表 5,其中子作业 0 是为了算法实现而添加的空作业,它的数据量为 0,因而没有执行时间。工作流 2 包含 10 个 Sort 子作业,子作业的执行顺序依赖关系见图 4,其数据量见表 6。

表 5 工作流 1 各子作业的数据量

Tab. 5 Data size of each sub-job in workflow 1

子作业	0	1	2	3	4	5	6
数据量/GB	0	4	2	2	1	2	6

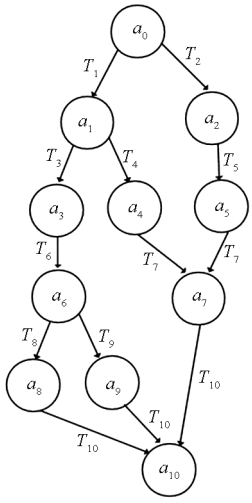

图 4 工作流作业 2 的 PAD

Fig. 4 PAD of workflow 2

表 6 工作流 2 各子作业的数据量

Tab. 6 Data size of each sub-job in workflow 2

子作业	0	1	2	3	4	5	6	7	8	9	10
数据量/GB	0	4	2	2	1	2	4	2	1	4	6

4.3 工作流作业 1 的预测运行时间和实际运行时间

使用 BFS 从正向和逆向对图 3 所示的 PAD 进行搜索,各子作业所处的层次关系见表 3。可以得到正向和逆向的工作流作业执行顺序。

正向搜索:子作业 1 →子作业 2、3、4 并行执行→子作业 5 →子作业 6。

逆向搜索:子作业 1 →子作业 2 和 3 并行执行→子作业 4 和 5 并行执行→子作业 6。

基于上述搜索结果对子作业进行系统资源分配,并使用基于 GBDT 的时间预测算法进行各子作业在对应资源分配方案下的执行时间预测,结果如表 7 和表 8 所示。

表 7 工作流作业 1 的正向资源分配方案及子作业的预测执行时间

Tab. 7 Resource allocation scheme under forward searching for workflow 1 and the predicted execution time of sub-jobs

子作业	资源分配量		子作业预测执行时间/s
	CPU 核数	内存/GB	
1	10	10	69
2	3	4	49
3	3	4	49
4	3	4	32
5	10	10	51
6	10	10	92

表 8　工作流作业 1 的逆向资源分配方案及子作业预测的执行时间

Tab. 8　Resource allocation scheme under backward searching for workflow 1 and the predicted execution time of sub-jobs

子作业	资源分配量		子作业预测执行时间/s
	CPU 核数	内存/GB	
1	10	10	69
2	5	6	39
3	5	6	39
4	5	6	23
5	5	6	39
6	10	10	92

根据各子作业的执行时间预测结果,进行关键路径的计算,可以得到如表 9 所示的关键路径和工作流作业的预测执行时间。

表 9　工作流作业 1 的关键路径和预测执行时间

Tab. 9　Critical path and the predicted execution time of workflow 1

搜索方向	关键路径	预测执行时间/s
正向	$a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_5 \rightarrow a_6$ $a_0 \rightarrow a_1 \rightarrow a_3 \rightarrow a_5 \rightarrow a_6$	261
逆向	$a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_5 \rightarrow a_6$ $a_0 \rightarrow a_1 \rightarrow a_3 \rightarrow a_5 \rightarrow a_6$	239

从表 9 可看出,逆向搜索出的子作业执行顺序可以获得较短的工作流作业执行时间,因此这个工作流的预测执行时间为 239 s。

表 10 列出了在逆向资源分配方案下工作流作业 1 中各个子作业的预测执行时间和实际运行时间,以方便进行对比。

表 10　工作流作业 1 的预测执行时间和实际运行时间对比

Tab. 10　Comparison of the predicted running time and actual running time of workflow 1

子作业	预测执行时间/s	实际执行实际/s	误差/%
1	69	62.01	11.27
2	39	45.31	-13.93
3	39	45.87	-14.98
4	23	19.89	15.64
5	39	35.06	11.24
6	92	83.13	10.67
总工作流	239	226.07	5.72

从表 10 可以看出,各子作业的预测执行时间误差都在 16% 以内。造成误差的主要原因是目前用于 GBDT 模型训练的数据量不是很大,未能将模型的超参数调整到最优的状态。另外一个原因是当前的 GBDT 算法实现采用了传统的方式,因此限制了预测的精度,后续将采用 XGBoost 实现 GBDT 的算法。XGBoost 对 GBDT 实现进行了深度优化,例如 XGBoost 支持多种类型的基分类器;能够对损失函数进行二阶泰勒展开,可以同时使用一阶和二阶导数;XGBoost 的目标函数多了正则项,使得学习得到的模型不容易过拟合。

由于子作业执行时间预测有的高于实际运行时间,有的低于实际运行时间,总的工作流的运行时间预测误差为 5.72%。

4.4　工作流作业 2 的预测运行时间和实际运行时间

使用广度优先搜索算法从正向和逆向对图 4 所示的 PAD 进行搜索,各子作业所处的层次关系见表 11。

表 11　工作流作业 2 的子作业的正向和逆向串并行执行关系

Tab. 11　Forward and reverse series-parallel execution relationship of the sub-jobs of workflow 2

节点位置	正向搜索	逆向搜索
第 0 层	a_0	a_{10}
第 1 层	a_1, a_2	a_7, a_8, a_9
第 2 层	a_3, a_4, a_5	a_4, a_5, a_6
第 3 层	a_6, a_7	a_2, a_3
第 4 层	a_8, a_9	a_1
第 5 层	a_{10}	a_0

可以得到正向和逆向的工作流作业 2 执行顺序。

正向搜索:子作业 1 和 2 并行执行→子作业 3、4、5 并行执行→子作业 6 和 7 并行执行→子作业 8 和 9 并行执行→子作业 10。

逆向搜索:子作业 1 →子作业 2 和 3 并行执行→子作业 4、5、6 并行执行→子作业 7、8、9 并行执行→子作业 10。

基于上述搜索结果对子作业进行系统资源分配和执行时间预测,得到的结果如表 12 和表 13 所示。

表 12 工作流作业 2 的正向资源分配方案及子作业的预测执行时间

Tab. 12 Resource allocation scheme under forward searching for workflow 2 and the predicted execution time of sub-jobs

子作业	资源分配量		子作业预测执行时间/s
	CPU 核数	内存/GB	
1	5	6	129
2	5	6	71
3	3	4	89
4	3	4	47
5	3	4	89
6	5	6	129
7	5	6	71
8	5	6	38
9	5	6	129
10	10	10	143

表 13 工作流作业 2 的逆向资源分配方案及子作业预测的执行时间

Tab. 13 Resource allocation scheme under backward searching for workflow 2 and the predicted execution time of sub-jobs

子作业	资源分配量		子作业预测执行时间/s
	CPU 核数	内存/GB	
1	10	10	101
2	5	6	71
3	5	6	71
4	3	4	47
5	3	4	89
6	3	4	186
7	3	4	89
8	3	4	47
9	3	4	186
10	10	10	143

根据各子作业的执行时间预测结果,进行关键路径的计算,可以得到如表 14 所示的关键路径和工作流作业的预测执行时间。

表 14 工作流作业 2 的关键路径和预测执行时间

Tab. 14 Critical path and the predicted execution time of workflow 2

搜索方向	关键路径	预测执行时间/s
正向	$a_0 \rightarrow a_1 \rightarrow a_3 \rightarrow a_6 \rightarrow a_9 \rightarrow a_{10}$	619
逆向	$a_0 \rightarrow a_1 \rightarrow a_3 \rightarrow a_6 \rightarrow a_9 \rightarrow a_{10}$	687

从表 14 可看出,对于工作流作业 2 而言,正向搜索得到的资源分配方案较好,因此工作流作业 2 的预测执行时间为 619 s。

表 15 列出了在正向资源分配方案下工作流作业 2 中各个子作业的预测执行时间和实际运行时间。可以看出,各子作业的预测执行时间误差都在 15% 以内,工作流的预测时间误差较小,为 1.57%。

表 15 工作流作业 2 的预测执行时间和实际运行时间对比

Tab. 15 Comparison of the predicted running time and actual running time of workflow 2

子作业	预测执行时间/s	实际执行实际/s	误差/%
1	129	112.86	14.30
2	71	80.56	-11.87
3	89	78.99	12.67
4	47	51.97	-9.56
5	89	96.92	-8.17
6	129	116.49	10.74
7	71	79.12	-10.26
8	38	33.08	14.87
9	129	142.61	-9.54
10	143	158.47	-9.76
总工作流	619	609.42	1.57

对包含 6 个子作业和 10 个子作业的工作流进行实验的结果表明,当工作流中子作业数量增加时,工作流执行时间预测算法能够保持一定的预测准确度,具有可扩展性。本文算法的单个作业运行时间预测的精度可以达到 84%。误差产生的基本原因是 GBDT 模型中特征和正则化参数的选择没有达到最优,后续的研究工作将进行更多的实验,优化模型参数配置。

4.5 与 Spark 默认作业调度算法的对比

为了验证基于关键路径分析的工作流作业调度算法的有效性,将工作流作业 1 和 2 分别使用本文算法和 Spark 默认作业调度算法进行运行。为了减少实验误差,每组实验分别执行 5 次,取 5 次结果的平均值作为工作流作业的执行时间。实验结果如图 5 所示。

对比工作流作业的预测执行时间和实际运行时间,可以看到对于工作流作业 1,采用基于关键路径分析的资源分配和作业调度算法,作业运行时间缩短了 15.71%;工作流作业 2 的作业运行

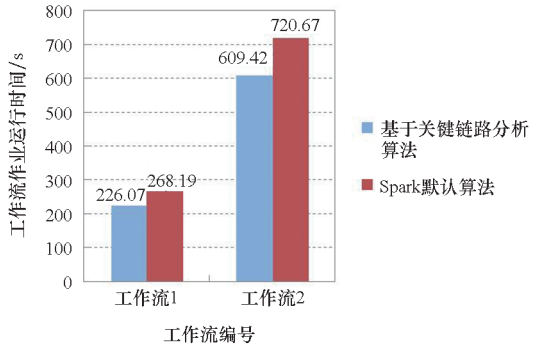

图 5　工作流作业执行时间对比

Fig. 5　Workflow job execution time comparison

时间缩短了 15.44%。可见,本文提出的算法能够根据工作流中各子作业的特点为其分配合适的资源,并进行有效的作业执行顺序调度,从而能够缩短工作流作业的完成时间。

5　结论与展望

为了提高服务质量和用户的满意度,数据中心、超算中心等需要在规定的时间范围内完成用户作业。本文提出一个针对小规模集群的基于关键路径分析的工作流作业运行时间预测和作业调度算法。该算法通过预测工作流中每一个子作业的执行时间,结合关键路径的计算,预测出工作流作业的执行时间。使用本算法给出的资源分配和作业调度方案,能够缩短工作流的执行时间,提高系统性能。

下一步的研究工作将在以下几个方面开展:

1)目前的工作主要针对小规模集群进行,即认为每个工作流独占系统的全部资源。后面的工作将取消这个假设,考虑多个工作流并发和系统资源的争用问题。

2)缩短关键路径上作业的执行时间能够压缩工作流作业的完成时间。当前在进行资源分配时,同层次的作业均分系统资源,没有考虑如何减少关键作业的执行时间。下一步,将为关键作业进行优先资源分配,在可能的范围内缩短其运行时间。

3)作业的执行时间与作业的类型关系密切,需要作为时间预测算法的输入。将选取不同类型的作业进行更多的实验,确定更为合适的 GBDT 模型参数配置,提高时间预测算法对不同类型作业的预测精度。

4)目前的实验较为简单,仅以二个实例说明本文提出的算法是可行的。下一步将扩大实验数据的来源。一方面,采用随机算法生成 DAG;另一方面寻找更多的应用实例。更多的实验将进一步说明算法的有效性和可扩展性。

参考文献(References)

[1] LEE J. Time-reversibility for real-time scheduling on multiprocessor systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(1): 230 – 243.

[2] DJIGAL H, FENG J, LU J M, et al. IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(5): 1057 – 1071.

[3] TOPCUOGLU H, HARIRI S, WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260 – 274.

[4] ZHOU N Q, QI D Y, WANG X Y, et al. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table[J]. Concurrency and Computation: Practice and Experience, 2017, 29(5): e3944.

[5] ZHANG Y J, TONG F, LI C Y, et al. Bi-objective workflow scheduling on heterogeneous computing systems using a memetic algorithm[J]. Electronics, 2021, 10(2): 209.

[6] KUMAR H, TYAGI I. Hybrid model for tasks scheduling in distributed real time system[J]. Journal of Ambient Intelligence and Humanized Computing, 2021, 12(1): 2881 – 2903.

[7] ZHENG X W, ZHOU Z, YANG X, et al. Exploring plan-based scheduling for large-scale computing systems[C]// Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), 2016: 259 – 268.

[8] LI K L, LIU C B, LI K Q. An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs[J]. Theoretical Computer Science, 2014, 543: 46 – 51.

[9] FU Z M, TANG Z, YANG L, et al. An optimal locality-aware task scheduling algorithm based on bipartite graph modelling for spark applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(10): 2406 – 2420.

[10] FAN Y P, LAN Z L, RICH P, et al. Scheduling beyond CPUs for HPC[C]//Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, 2019: 97 – 108.

[11] FAN Y P, LAN Z L, CHILDERS T, et al. Deep reinforcement agent for scheduling in HPC[C]//Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021: 807 – 816.

[12] SUKHOROSLOV O V, NAZARENKO A, ALEKSANDROV R. An experimental study of scheduling algorithms for many-task applications[J]. The Journal of Supercomputing, 2019, 75(12): 7857 – 7871.

[13] ADHIKARI M, AMGOTH T, SRIRAMA S N. A survey on scheduling strategies for workflows in cloud environment and emerging trends[J]. ACM Computing Surveys, 2020, 52(4): 1 – 36.

[14] ILAVARASAN E, THAMBIDURA P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments[J]. Journal of Computer Science, 2007, 3(2): 94 – 103.

[15] SULAIMAN M, HALIM Z, WAQAS M, et al. A hybrid list-based task scheduling scheme for heterogeneous computing[J]. The Journal of Supercomputing, 2021,

77(9): 10252 – 10288.

[16] MARTINEZ A V. Scheduling in heterogeneous distributed computing systems based on internal structure of parallel tasks graphs with meta-heuristics[J]. Applied Sciences, 2020, 10(18): 6611.

[17] YU L, TENG F, NING S M, et al. A two steps method of resources utilization predication for large Hadoop data center[J]. Concurrency and Computation: Practice and Experience, 2020, 32(15): e5634.

[18] HAN M L, GUAN N, SUN J H, et al. Response time bounds for typed DAG parallel tasks on heterogeneous multi-cores[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(11): 2567 – 2581.

[19] SERRANO M A, QUIÑONES E. Response-time analysis of DAG tasks supporting heterogeneous computing[C]// Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018: 1 – 6.

[20] GAO Z P, WANG T, WANG Q, et al. Execution time prediction for apache spark[C]// Proceedings of the International Conference on Computing and Big Data, 2018: 47 – 51.

[21] KUMBHARE N, MARATHE A, AKOGLU A, et al. A value-oriented job scheduling approach for power-constrained and oversubscribed HPC systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(6): 1419 – 1433.

[22] FAN Y P, RICH P, ALLCOCK W E, et al. Trade-off between prediction accuracy and underestimation rate in job runtime estimates[C]//Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), 2017: 530 – 540.

[23] SIKAROUDI A E, PARK C. A mixture of linear-linear regression models for a linear-circular regression[J]. Statistical Modelling, 2021, 21(3): 220 – 243.

[24] WANG J T, QIAN Y H, LI F J, et al. Fusing fuzzy monotonic decision trees[J]. IEEE Transactions on Fuzzy Systems, 2020, 28(5): 887 – 900.

[25] ZENG J Y, TAN Z H, MATSUNAGA T, et al. Generalization of parameter selection of SVM and LS-SVM for regression[J]. Machine Learning and Knowledge Extraction, 2019, 1(2): 745 – 755.

[26] WANG Y, XIA S T, TANG Q T, et al. A novel consistent random forest framework: bernoulli random forests[J]. IEEE Transactions on Neural Networks and Learning Systems, 2018, 29(8): 3510 – 3523.

[27] YUAN X M, WANG X, HAN J C, et al. A high accuracy integrated bagging-fuzzy-GBDT prediction algorithm for heart disease diagnosis[C]//Proceedings of the 2019 IEEE/CIC International Conference on Communications in China (ICCC), 2019: 467 – 471.

[28] LV S W, LIU G, BAI X. Multifeature pool importance fusion based GBDT (MPIF-GBDT) for short-term electricity load prediction[J]. IOP Conference Series: Earth and Environmental Science, 2021, 702(1): 012012.

[29] PENG S C. A GBDT based quality prediction method for the resistance spot welding[C]//Proceedings of the International Conference on Security, Pattern Analysis, and Cybernetics (SPAC), 2021: 451 – 455.

[30] DENG Q F, FAGHANIMAKRANI T, AGHVAMI A H. GBDT-based modules for force prediction in a model-mediated teleoperation system[C]//Proceedings of the 27th International Conference on Telecommunications (ICT), 2020: 1 – 6.

[31] LEI H. Financial index data prediction based on improved GBDT model[C]//Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2021: 697 – 702.

[32] BAI X M, FENG Y X, LI L X, et al. Research on prediction of urban road congestion based on spark-GBDT[C]// Proceedings of the IEEE 5th International Conference on Intelligent Transportation Engineering (ICITE), 2020: 102 – 106.

[33] 翁剑成, 付宇, 林鹏飞, 等. 基于梯度推进决策树的日维度交通指数预测模型[J]. 交通运输系统工程与信息, 2019, 19(2): 80 – 85, 93.
WENG J C, FU Y, LIN P F, et al. GBDT method based on prediction model of daily dimension traffic index[J]. Journal of Transportation Systems Engineering and Information Technology, 2019, 19(2): 80 – 85, 93. (in Chinese)

[34] 詹剑锋, 高婉铃, 王磊, 等. BigDataBench: 开源的大数据系统评测基准[J]. 计算机学报, 2016, 39(1): 196 – 211.
ZHAN J F, GAO W L, WANG L, et al. BigDataBench: an open-source big data benchmark suite[J]. Chinese Journal of Computers, 2016, 39(1): 196 – 211. (in Chinese)