软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(3):985−1004 [doi: 10.13328/j.cnki.jos.006447]

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

面向 Flink 迭代作业的动态资源分配策略*

岳晓飞[1], 史 岚[1], 赵宇海[1], 季航旭[1], 王国仁[2]

[1](东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

[2](北京理工大学 计算机学院, 北京 100081)

通信作者: 赵宇海, E-mail: zhaoyuhai@mail.neu.edu.cn

摘 要: 新兴分布式计算框架 Apache Flink 支持在集群上执行大规模的迭代程序, 但其默认的静态资源分配机制导致无法进行合理的资源配置来使迭代作业按时完成. 针对这一问题, 应该依靠用户来主动表达性能约束而不是被动地进行资源保留, 故提出了一种基于运行时间预测的动态资源分配策略 RABORP (resource allocation based on runtime prediction), 来为具有明确运行时限的 Flink 迭代作业制定动态资源分配计划并实施. 其主要思想是: 通过预测各个迭代超步的运行时间, 然后根据预测结果在迭代作业提交时和超步间的同步屏障处分别进行资源的初始分配和动态调整, 以保证可使用最小资源集, 使迭代作业在用户规定的运行时限内完成. 通过在不同数据集下执行多种典型的 Flink 迭代作业进行了相关对比实验, 实验结果表明, 所建立的运行时间预测模型能够对各个超步的运行时间进行准确预测, 而且在单作业和多作业场景下, 采用所提出的动态资源分配策略相比于目前最先进算法在各项性能指标上都有所提升.

关键词: 迭代作业; 运行时间预测; 资源分配; 运行时限; Apache Flink

中图法分类号: TP311

中文引用格式: 岳晓飞, 史岚, 赵宇海, 季航旭, 王国仁. 面向 Flink 迭代作业的动态资源分配策略. 软件学报, 2022, 33(3): 985–1004. http://www.jos.org.cn/1000-9825/6447.htm

英文引用格式: Yue XF, Shi L, Zhao YH, Ji HX, Wang GR. Dynamic Resource Allocation Strategy for Flink Iterative Jobs. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 985−1004 (in Chinese). http://www.jos.org.cn/1000-9825/6447.htm

Dynamic Resource Allocation Strategy for Flink Iterative Jobs

YUE Xiao-Fei[1], SHI Lan[1], ZHAO Yu-Hai[1], JI Hang-Xu[1], WANG Guo-Ren[2]

[1](School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

[2](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Apache Flink, an emerging distributed computing framework, supports the execution of large-scale iterative programs on the cluster, but its default static resource allocation mechanism makes it impossible to carry out reasonable resource allocation to make iterative jobs complete on time. In response to this problem, that users should be relied on to actively express performance constraints rather than passively retain resources. RABORP, a dynamic resource allocation strategy based on runtime prediction is proposed to develop and implement a dynamic resource allocation plan for Flink iterative jobs with clear runtime limits. The main idea is to predict the runtime of each iteration superstep, and then the initial allocation and dynamic adjustment of resources are performed at the time of the iterative job submission and the synchronization barrier between the supersteps according to the predicted results, to ensure that the minimum set of resources can be used to complete the iterative job within the runtime limit specified by the user. A variety of typical Flink iterative jobs were executed under the dataset to carry out relevant comparative experiments. Experimental results show that the established runtime prediction model can accurately predict the runtime of each superstep, and compared with the current state-of-the-art algorithms, the proposed dynamic resource allocation strategy used in single-job and multi-job scenarios has improved various

* 基金项目: 国家重点研发计划(2018YFB1004402); 国家自然科学基金(61772124)
本文由“数据库系统新型技术”专题特约编辑李国良教授、于戈教授、杨俊教授和范举教授推荐.
收稿时间: 2021-06-30; 修改时间: 2021-07-31; 采用时间: 2021-09-13; jos 在线出版时间: 2021-10-21

performance indicators.

Key words: iterative job; runtime prediction; resource allocation; runtime limit; Apache Flink

迭代计算通常是机器学习和图处理算法的核心部分[1,2], 即一个或多个步骤被重复执行, 直到满足收敛标准, 在数据挖掘、社交网络等各类应用中起着举足轻重的作用. 随着互联网技术的高速发展, 各种场景下的数据集规模日益增多, 使得在工业和科学领域上使用 Apache Spark[3]和 Flink[4]等分布式计算框架进行分布式迭代计算已经变得非常普遍[5]. 分布式迭代作业在真实生产中通常占据很大比重, 而且往往需要反复执行[6], 它们大都需要遵守服务级目标(service level goal, SLO)[7], 需要在严格规定的运行时限内完成, 因为外部应用通常需要及时反馈的分析结果, 例如, Twitter 公司明确要求在 10 分钟内完成对 TB 级日志数据索引的搜索和查询[8], 时延太高会违反与用户签订的服务水平协议(service level agreement, SLA)[9]. 在大多数情况下, 用户往往会过度地为作业进行资源供应, 这导致集群整体的资源利用率较低, 而且产生了庞大的运维成本[10], 这种资源分配不均衡现象对于需要循环执行的迭代作业的影响更大.

Apahce Flink 是一种低延迟、高吞吐的新型分布式计算框架, 相对于其他主流框架, 具有极高的迭代计算效率[11−13], 已经受到学术界和工业界的广泛关注. 虽然 Flink 在迭代计算方面有一定的优越性, 但它并不会将迭代作业的时限需求作为资源分配的依据, 而是简单地通过用户请求来进行初始资源配置, 该配置在作业开始后就不再改变[14]. 同时, 近年来还没有针对 Flink 迭代作业资源分配优化的有效解决方案, 而且大部分工作[15−17]都专注于对资源调度的优化, 仍具有无法解决目前普遍存在的集群资源浪费现象的局限性. 所以, 如何在 Flink 环境下及时优化集群资源供应来使迭代作业按时完成, 是目前亟需解决的问题.

分布式作业的执行行为通常具有可预测性[18], 因为除了循环执行的迭代作业外, 其他批处理作业也通常长时间周期性执行, 它们的特点是每次执行使用相同的算法逻辑来处理不同规模的数据, 而且容易获取执行过程中的各项统计信息, 所以目前对集群资源分配优化的研究[19−28]主要依赖于建立机器学习模型进行性能预测, 来为作业选择最佳资源配置, 已取得了一系列有效的研究成果. 但是这些工作普遍存在模型复杂、训练开销大以及对迭代作业的优化效果较差等问题. 不同于以往的工作, 我们认为, 应该依靠用户来主动表达性能约束而不是被动地进行资源保留, 因此提出了面向 Flink 迭代作业的动态资源分配策略 RABORP, 该策略的核心目标是: 使用最少的资源来使 Flink 迭代作业在用户期望的时间内完成. 同时, 该策略并不局限于 Flink 系统, 它能够很轻易地移植在使用整体同步并行模型(bulk synchronous parallel, BSP)构建迭代计算程序的其他框架上. 本文的主要贡献如下:

- 提出了面向迭代超步的轻量级运行时间预测模型, 可针对不同的预测目标在参数回归模型和非参数回归模型之间进行选择.
- 提出了基于运行时间预测的动态资源分配策略 RABORP, 可根据当前迭代作业剩余超步运行时间的预测值和用户提出的作业时限要求制定动态资源分配计划并实施.
- 通过在 Flink on Yarn 框架的基础架构上添加一些必要的组件, 实现了 RABORP 策略的原型系统.
- 通过在不同数据集上执行多个典型的 Flink 迭代作业来进行对比实验, 说明了所提出的运行时间预测模型和 RABORP 策略的正确性和性能提升效果.

本文第 1 节对目前与本文研究密切相关的工作进行介绍和总结. 第 2 节主要介绍 Flink 中的迭代计算模型以及通信模式等背景知识和与所研究问题相关的定义. 第 3 节对面向超步的运行时间预测模型的建立和拓展过程进行说明. 第 4 节基于运行时间预测来对当前迭代作业的资源进行动态分配来满足用户需求. 第 5 节在多种数据集上进行相关对比实验, 对所提出的方法进行评估. 第 6 节对全文整体的工作进行总结, 并对未来值得继续深入研究的方向进行展望.

1 相关工作

本节从 2 个方面介绍与研究内容相关的现有工作: (1) 运行时间预测方法; (2) 自动资源分配技术.

1.1 运行时间预测方法

运行时间预测不仅对于解决作业可行性问题是一个非常有用的机制, 而且可用于对集群资源的分配和部署成本的优化. 在过去几年里, 有许多在大规模数据处理平台上观察、分析和预测大规模作业的运行时间的工作, 这些工作被广泛应用于各种分布式集群和云环境下.

- 运行时间预测最早大量出现在 Hadoop 系统上且开始被人们广泛熟知. Starfish[29]在 Hadoop 上引入了一个自调整框架, 该框架通过在部分数据上运行 MapReduce 作业来观察和分析作业的运行时间指标, 并通过调整其配置选项来优化系统性能; ParaTimer[30]通过识别 MapReduce 作业 DAG 结构中的关键路径, 并估计该路径的运行时间来达到对作业整体运行时间的预测.
- 针对目前应用最广泛的 Spark 系统, 文献[31]介绍了采用模拟驱动的模型, 通过模拟 Spark 作业在部分数据上的执行并收集其执行指标, 如内存消耗、I/O 成本和访存行为等, 来预测 Spark 作业的执行时间; 文献[32]中提出了一种对 Spark 作业运行时间进行预测的灰盒建模方法, 其主要通过白盒模型来预测弹性分布式数据集(resilient distributed datasets, RDD)的大小, 然后通过预测的 RDD 大小建立黑盒模型, 来对任务整体的运行时间进行估计.
- 运行时间预测还可应用于云资源的部署成本优化, Ernest[33]通过在少量云实例下捕获高级分析作业的计算和通信模式来建立性能预测模型, 然后该模型被用于模拟更多实例时该作业的运行时间.

上述研究基本都依赖于特定的分布式框架, 而且不能对迭代作业进行很好的支持. 目前仍然有一些工作是面向迭代作业的: PREDIcT[34]是一种用于预测在 Giraph 上实现的图迭代算法的运行时间的方法, 它通过样本运行来捕捉每次迭代的关键输入特征, 然后建立成本模型来预测迭代次数和每次迭代的运行时间; SMiPE[35]通过捕捉已完成的迭代的硬件利用率等特性来将运行中的作业与之前的执行进行匹配, 并基于所匹配的历史执行来估计迭代数据流的进度.

1.2 自动资源分配技术

在传统的 MapReduce、Spark 和新兴的 Flink 系统中, 均存在对迭代作业资源分配不均衡、算法难以在用户期望的时间内收敛的问题. 针对上述问题, 许多研究通过在前期对作业的执行行为进行预测, 来设计并实现资源动态分配或调整的端对端系统. 这些研究成果大致可分为以下两类.

(1) 静态资源分配

目前已有多个系统使用静态资源分配技术进行资源的初始分配,包括 AROMA[36]、GML[37]、Q-Flink[38]等与特定框架集成的系统和 PerfOrator[39]、Bell[40]、DECEF[41]等支持不同分布式框架的通用解决方案.

- 在针对特定框架的系统中, 经典的 AROMA 根据资源利用率对历史 MapReduce 作业进行聚类, 然后按类别分别进行性能模型的训练, 该模型可用于从异构资源中选择类别匹配后的作业, 这也是早期许多工作[36,42,43]的核心思想; GML 首次引入生成对抗网络来生成部分训练数据, 并建立机器学习模型来为 Fink 系统设置最佳资源配置参数, 以提高作业执行效率; Q-Flink 是一种适用于高流量负载下的 Flink 资源分配控制器, 它监测共享资源在共建应用中的干扰, 并根据底层资源使用的性能状态的连续反馈来分配资源, 以减少作业的服务质量(quality of service, QoS)违反率; 文献[44]利用蒙特卡洛仿真框架来实现低成本的训练, 从而能够使用少量的样本数据和资源来对较大的数据集和 Spark 集群下的虚拟内核和内存的总量进行可靠的预测和推荐.
- 在通用解决方案中, PerfOrator 利用非线性回归对作业的性能进行建模, 来实现资源的自动分配, 然而 PerfOrator 需要框架模型作为输入来准确捕捉并行性, 因此需要进行特定配置才能直接适用于各种分布式系统; Bell 通过建立机器学习模型捕捉作业的 scale-out 行为来对虚拟化的集群工作节点规模进行弹性伸缩来提供合理的资源供应; DECEF 通过将对预期响应时间和服务器利用率不平衡等优化目标投射到博弈论环境中, 并基于差分进化来解决该博弈问题来优化异构集群下的资源分配.

(2) 动态资源分配

同样地，与本文相似的使用动态资源分配技术来对运行中的作业进行动态资源调整的研究包括 Jockey[19], Quasar[20]和 Ellis[21]等通用解决方案和其他针对特定框架的工作.

- 在通用解决方案中，Jockey 和 Quasar 均使用详细的作业统计数据来建立表示作业的经济价值与总执行时间的关系的效益函数来代替运行时限，然后监控运行中的作业并使其效益最大化，以动态地进行资源分配; Ellis 提出了对迭代作业的运行时间分阶段进行建模和预测的思想，并通过限制运行时间的偏离幅度来进行动态的资源分配和调整，并实现了与 Spark 系统进行集成的原型系统；文献[22]提出了基于共享和学习分布式作业的历史运行时间数据来寻找最佳资源配置方法，它通过持续的细粒度集群监控来学习工作负载特性，然后协作共享预测结果并改进作业调度方式来自适应地调整资源分配.
- 与特定框架集成的研究工作主要集中在 Spark 和 Flink 等系统以及云环境下.
  1) 针对 Spark 系统，文献[23]提出了一种将运行时间预测与资源分配相结合的两阶段闭环算法来实现应用程序级的动态资源分配，旨在根据运行期间工作负载和资源需求的变化来调整计算资源；文献[24]首次建立了包含故障变量的广义数学模型，以根据集群和网络条件的故障概率计算出所需的最佳资源规模，并阶段性调整作业的资源分布，使其在目标时间内完成.
  2) 针对 Flink 系统，文献[25]提出的解决方案借鉴了闭环控制理论中的原理，通过建立 CPU 和内存调整机制来满足提交的 Flink 应用程序所要求的不同 QoS 水平；文献[26]则通过在 Flink 迭代作业运行过程中持续监测 CPU 利用率，并在 CPU 利用率下降或者上升到某一阈值时，动态调整后续迭代任务的并行度来进行资源的弹性分配.
  3) 在云环境下，文献[27]提出了一种改进的任务调度和最优功率最小化的方法来实现高效的动态资源分配过程，通过使用预测机制和动态资源表更新算法，提高了任务完成和响应时间等方面的资源分配效率；文献[28]通过在一组由 RAM 和 CPU 资源组成的可用虚拟机上安排独立任务，来模拟资源提供者在云中所面临的容错性和能源消耗等关键问题，并通过解决与成本之间的冲突来提高提供给客户的 QoS.

本文提出的 RABORP 策略与上述动态资源分配技术的不同之处在于: (1) RABORP 策略通过在资源分配过程中充分考虑迭代作业的执行特征，一定程度上解决了目前研究工作无法对迭代作业进行有效支持的问题; (2) RABORP 策略被实现为一个轻量级、可拔插的系统，能够很轻易地移植在使用 BSP 模型构建迭代计算程序的其他框架上; (3) RABORP 策略通过与 Yarn 系统协同工作来进行容器级的资源分配，可以较低的时间开销来进行全面且细粒度的资源调整; (4) RABORP 策略首次将用户对作业运行时间的约束引入资源分配算法中，并将其作为资源分配以及动态调整的主要依据.

2 预备知识

2.1 Flink中的迭代计算

在 MapReduce 被推出后，一些支持迭代计算的分布式计算框架就陆续出现，例如 Pregel[45]等基于 BSP 模型的框架和 PowerGraph[46]等非 BSP 的迭代框架. 目前, Flink 是对迭代计算支持最完善且效率最高的新兴分布式计算框架. 如图 1 所示，Flink 主要通过定义步骤函数并将其嵌入到一个包含执行图的特殊迭代算子中来支持迭代计算. 同时，为了维护基于 DAG 的运行时和调度器，它通过建立迭代“头”和“尾”任务以及互相连接的反馈边来构建反馈通道，并通过控制事件为反馈通道中的数据记录提供协调功能[47].

(1) Flink 对图迭代作业的支持

Flink-Gelly 库利用 Flink 迭代算子来支持大规模迭代图处理，其中包括以顶点为中心的 vertex-centric 模型[48]，也即基于 BSP 模型的“Pregel”，它支持每个顶点的任意计算和消息传递，是 Flink-Gelly 库中提供的最通用的迭代模型. 如图 2 所示，基于 BSP 模型构建的迭代作业是以连续的超步来运行的，每个超步由以下 3 个

阶段组成.

- 本地计算. 集群中的每个工作节点对存储在本地内存中的数据进行互相独立的平行计算.
- 消息传递. 在本地计算结束后, 不同工作节点之间通过通信技术进行数据交换.
- 全局同步. 所有工作节点在障碍处进行信息同步, 以确保所有节点都已完成本次计算.

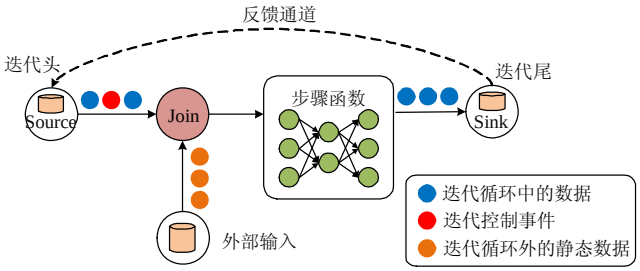


图 1 Flink 中迭代计算模型的基本结构

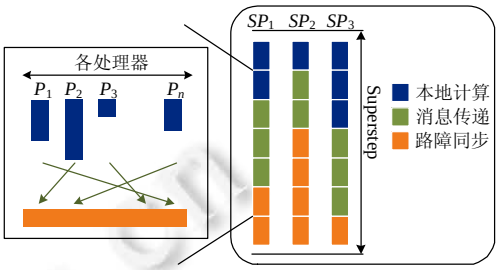


图 2 基于 BSP 模型的迭代作业执行阶段

(2) Flink 对非图迭代作业的支持

Flink 对于 *K*-Means 等非图迭代作业虽然无法通过 Flink-Gelly 中的迭代模型来进行构建, 但是它同样借鉴了 BSP 模型中的相关概念来进行实现. 如图 3 所示, Flink 将所有并行实例上步骤函数的一次执行称为超步, 也即同步的粒度. 相邻的超步间存在着同步屏障, 即当前超步的所有并行任务都必须在下一个超步初始化之前完成.

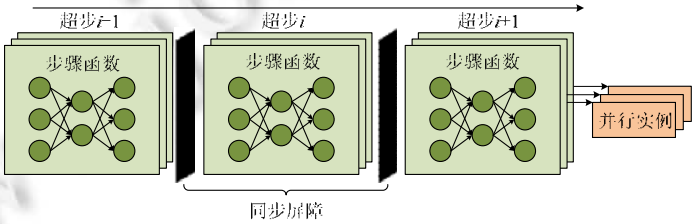
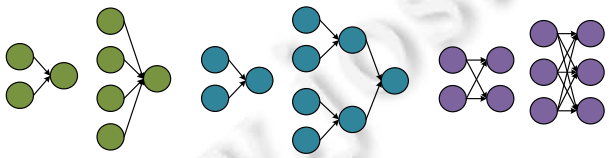


图 3 Flink 迭代计算的执行过程

2.2 分布式迭代作业的通信模式

Chowdhury 等人[49]发现: 在大规模分布式作业中, 仅有部分通信模式会反复地出现, 如图 4 所示. 这些通信模式包括: (1) collect 模式, 即所有分区的数据被发送到某个节点上; (2) Tree Aggregation 模式, 即使用树状结构来进行数据聚合; (3) Shuffle 模式, 即来自不同源节点的数据被传递到不同的目的节点. 这些通信模式并不是分布式迭代作业特有的, 但是由于迭代作业重复执行的特点, 造成这些通信模型出现的占比更高.



(a) Collect (b) Tree Aggregation (c) Shuffle

图 4 分布式迭代作业中常见的通信模式

图 5 显示了前期对不同迭代作业的通信模式占比的统计结果, 意味着可以尝试自动推断出迭代作业执行中的通信成本是如何随着计算资源规模的增加而变化的. 例如, 假设每个计算节点所处理的数据是恒定的, 那么所传递的数据量会随着计算节点数量的增加而增多, 从而使得在 collect 模式下所花费的通信时间以 O(*machine*)的速度增长. 同样地, 在树状聚合模式下, 所花费的时间也以 O(log(*machine*))的速度增长.

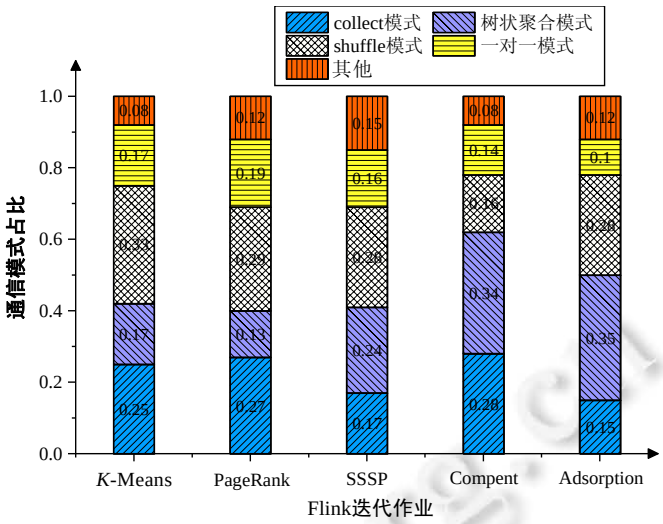


图 5 迭代作业中各项通信模式的占比

2.3 问题定义

本节将对所研究的关键问题和其挑战性进行梳理，对解决方案进行概括，并对相关基本概念做出定义.

本文的主题是研究 Flink 环境下迭代作业的动态资源分配，即“给定 Flink 迭代作业的运行时限，如何自动且持续为该作业分配最小的必要资源集?”，而在该研究目标下，所要解决的问题是双重的.

- 运行时间预测. 迭代作业的运行时间取决于输入数据集、资源利用率、算法参数等多种因素，捕捉上述特征对作业运行时间的影响，需要详细的统计数据并且建立高度复杂的性能模型，但是在作业执行前难以获得完整的统计数据，在作业执行时还会造成极大的时间开销. 因此，问题的第 1 部分是: 如何针对 Flink 迭代作业建立高精度、低延迟的轻量级运行时间预测模型?
- 运行时间差异. 由于迭代间依赖以及集群环境不同程度的干扰，迭代作业中不同超步的运行时间存在相当大的差异. 运行时间差异可简单地通过按最坏情况进行配置来解决，但很容易造成资源过度供应和集群利用率降低. 因此，问题的第 2 部分是: 如何解决在基于运行时间预测为迭代作业分配最小的资源集时存在的运行时间差异?

如图 6 所示，本文的解决方案是: 让用户明确说明他们期望的作业运行时间，然后以超步为对象建立运行时间预测模型，并在重复执行的历史作业上进行模型训练，最终根据所预测的运行时间来分配最小资源集以满足用户需求; 同时，持续地通过监控和动态调整来解决运行过程中集群的性能差异问题.

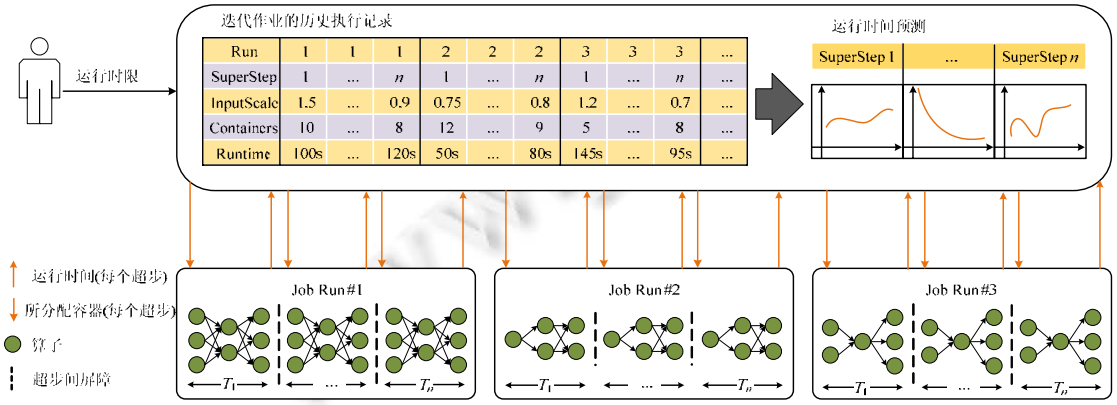


图 6 基于超步间的运行时间预测来动态分配资源以满足运行时限

定义 1(超步). 在 Flink 迭代计算中, 超步被定义为所有并行任务中步骤函数的一次执行. 相邻超步间存在同步屏障, 即在下一个超步被初始化之前, 该次迭代中的所有并行任务都需要完成当前超步, 超步也即同步的粒度. 超步的运行时间被定义为下一次超步与当前超步的开始时间之差.

定义 2(容器). 在 YARN 等资源管理系统中, 容器作为资源分配的基本单位, 被不同的分布式框架请求, 并应用于相应的分布式作业[50]. 容器中封装了 CPU、内存、磁盘、网络带宽等基础计算资源, 封装规则由用户配置文件来定义. 本文中, 对于系统资源的分配均以容器为对象.

定义 3(运行时间预测模型). 在分布式迭代作业中, 令 $x^{(i)}$为第 i 个超步使用的集群资源和输入特征, $y^{(i)}$为该超步的运行时间, 运行时间预测模型就是要建立如公式(1)所示的函数关系.

$$y^{(i)}=\omega x^{(i)}+\mu+\varepsilon_i \tag{1}$$

其中, ω是模型中特征对应的参数, μ是常数项, 上述参数均可通过最小二乘法等最优化算法计算得出; ε_i 是预测过程中的随机误差.

定义 4(动态资源分配). 本文并不要求迭代作业必须在隔离状态下运行以及有完全可预测的性能, 而且也不对并发作业和容器放置的调度方式进行限制. 我们通过专注于在运行时为当前迭代作业进行容器数量的动态扩展, 来尽可能地使该作业满足其运行时限, 称为动态资源分配.

3 面向超步的运行时间预测

本节将介绍以超步为对象建立的运行时间预测模型. 该模型主要用于预测 Flink 迭代超步的运行时间. 模型建立的过程主要包括训练数据收集、模型定义以及模型拓展这 3 部分.

3.1 训练数据收集

在真实世界中, 分布式迭代作业通常在集群中进行周期性运行来处理相似规模的数据集, 所以可以合理地假设历史作业与当前迭代作业有相同或者相近的迭代次数, 故本文采用历史迭代作业在执行过程中的相关统计数据作为运行时间预测模型的训练数据; 对于历史数据较匮乏的情况, 则采用抽样执行的方式快速生成数据点, 即将不同的迭代作业运行在以不同的抽样比生成的样本数据集上. 同时, 采用 Popescu 等人[34]提出的偏向随机跳转方法(biased random jump, BRJ)与参数缩放相结合的方式保证抽样执行的迭代次数能够和当前迭代作业保持一致. 由于抽样执行仅仅用于收集运行时间预测模型训练的数据点, 并不需要考虑迭代作业执行结果的准确性, 所以可通过该方法快速地进行数据点的收集.

3.2 运行时间预测模型

本节所介绍的以超步为对象的运行时间预测模型是一种混合模型, 包括参数回归和非参数回归这两部分.每个超步下的回归模型可由一个函数 $f:X\rightarrow Y$ 来表示, X 代表当前超步下所使用的容器数量和数据规模, Y 代表该超步的运行时间.

(1) 参数回归

在综合考虑基于 BSP 构建的 Flink 迭代计算模型后, 使用如公式(2)所示的基于 Ernest[33]改进的参数回归模型来捕捉 Flink 迭代作业中各个超步消耗在计算和通信两方面的时间, 而同步实质上是不同节点间的信息交互, 故将其隐式地涵盖在通信过程内, 不进行显式表达.

$$time=\omega_0+\omega_1\times\left(scale\times\frac{1}{containers}\right)+\omega_2\times\sqrt{containers}+\omega_3\times\log(containers)+\omega_4\times containers \tag{2}$$

其中, $containers$ 表示在本次超步中使用的 Yarn 容器的数量, $scale$ 代表本次超步经过归一化后的数据规模, $time$ 为本次超步执行所花费的时间, $\omega=(\omega_0,\ldots,\omega_4)$为每个特征项的权值.

公式(2)是适用于 Flink 迭代作业的分布式计算和通信模型, 它由一系列非线性特征经过线性组合得到. 这些非线性特征代表在容器协同处理数据过程中花费在本地计算和远程通信上的时间, 按照公式(2)中的顺序依次为: (1) 固定成本项, 代表串行计算的时间; (2) 数据规模和容器数量倒数的乘积项, 代表并行计算的时

间; (3) 根号项, 代表多对一通信模式的时间; (4) 对数项, 代表树聚合通信模式的时间; (5) 线性项, 代表一对一通信模式和容器的固定开销. 本文采用非负最小二乘法(non-negative least squares, NNLS)来进行模型参数的训练, 因为 NNLS 相比于梯度下降等算法可以保证模型中的权重向量ω为非负数, 尤其对于运行时间预测模型来说, 所有参数项均代表作业花费在各方面的时间, 故非负性是避免过拟合的一个重要机制.

(2) 非参数回归

基于给定模型的参数回归模型是在前期实验和分析的先验基础上建立的, 可能无法很好地捕捉影响迭代作业执行的所有特征. 但非参数回归恰恰相反, 它可以通过假设训练数据中局部定义的行为来自动推导模型. 本文使用局部线性回归(local linear regression, LLR)和高斯核来对回归函数进行估计, 所建立的非参数回归模型如公式(3)所示.

$$time=g(containers,scale)+\varepsilon \tag{3}$$

则根据核函数估计的思想, 给定容器数量 C 和数据规模 S 对应的 $time$ 值可按式(4)进行估计.

$$g(C,S)=\frac{\sum_{i=1}^{n}K\left(\frac{C-containers_i}{h}\right)\times K\left(\frac{S-scale_i}{h}\right)\times time_i}{\sum_{i=1}^{n}K\left(\frac{C-containers_i}{h}\right)\times K\left(\frac{S-scale_i}{h}\right)} \tag{4}$$

其中, $K(u)=(2\pi)^{-\frac{1}{2}}\exp\left\{-\frac{1}{2}u^2\right\}$, $K(\cdot)$为高斯核函数, 用于确定数据点在估计 $g(containers,scale)$中的权重; n 为用于训练的数据点的数量; h 为控制局部领域大小的窗宽.

但是由于 LLR 的局部优化准则, 导致其只能用于插值, 而不能用于外推. 也就是说, 采用非参数回归的运行时间预测模型只能对容器数在训练集中最小和最大容器数之间的情况下进行预测.

3.3 模型拓展

如第 3.2 节所述, 超步间的运行时间预测模型可通过参数回归和非参数回归两种方式来针对 Flink 迭代作业的运行时间进行建模. 在可用的训练数据较密集的情况下, 非参数回归可以准确地拟合迭代作业的运行特征. 然而, 当只有少数训练数据点时, 参数回归可能提供更好的结果. 所以, 为了充分利用所收集的数据点并提高拟合精度, 在训练前还需要进行模型选择来拓展运行时间预测模型, 以提高其灵活性. 特别地, 由于非参数回归模型无法用于外推, 故模型选择仅在插值情况下进行.

本文利用交叉验证在两个回归模型之间进行选择, 通过对两种模型分别进行 K 折交叉验证来选择平均误差最小的模型. 具体来说, 即在所收集的训练数据点中剔除提前终止迭代的运行和极端的异常值后, 假设剩余的数据点分别来自 K+2 类迭代作业的执行结果, 那么按迭代作业类型来将数据集分为 K 份来进行交叉验证. 由于模型选择仅考虑插值情况, 所以平均容器数量最小和最大的 2 折无法作为验证集, 故将其忽略.

4 基于运行时间预测的动态资源分配策略

本节将介绍基于运行时间预测的动态资源分配策略 RABORP, 它主要包括两个阶段: (1) 在迭代作业提交时进行资源的初始分配; (2) 在迭代作业执行过程中自适应地对已分配资源进行调整.

4.1 动态资源分配算法

RABORP 策略主要根据迭代作业每个超步运行时间的预测结果来进行资源分配, 它的核心是一种贪心的思想. 图 7 展示了 RABORP 策略进行资源分配的基本过程, 其分配原则是: 在资源边界之间, 搜索能够让当前迭代作业的运行时间预测值小于运行时限的最小容器数.

在迭代作业提交时, RABORP 策略在系统可用资源允许的范围内, 搜索能够使所有超步运行时间的预测值之和小于迭代作业时限的最小容器数. 如果没有, 那么就选择能够使所有超步运行时间的预测值之和最小的容器数进行初始分配. 若已知某超步使用的容器数量和运行时间, 那么可通过建立第 3.2 节介绍的运行时

间预测模型来拟合面向该超步的回归函数 f_i.

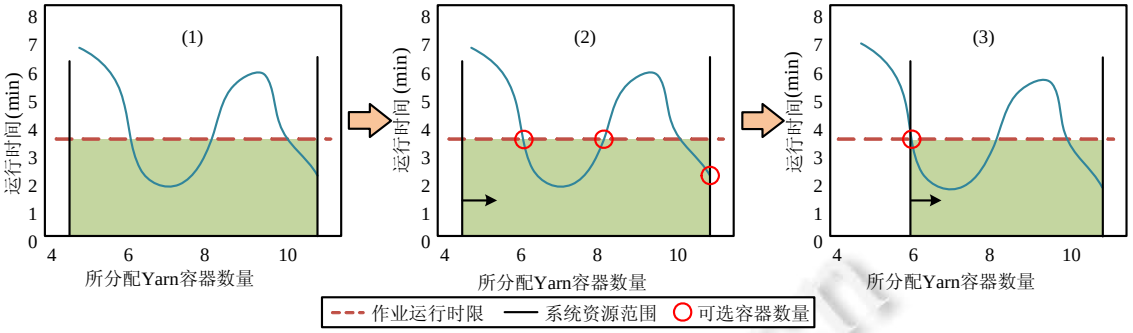


图 7 RABORP 策略进行资源分配的基本过程

算法 1 描述了在迭代作业提交时进行初始资源分配的过程, 该过程首先计算出在所提供的最小和最大容器范围内能够满足预测的运行时间的有效容器集, 如果这个解集是非空, 程序就使用解集中最小的容器数; 否则, 程序就使用预测的运行时间最小时的容器数.

算法 1. 迭代作业提交时的初始资源分配算法.

输入: 当前提交的迭代作业 *Job*, 运行时限 *C*, 初始数据规模 *scale*, 最小容器数 $x_{\min}$, 最大容器数 $x_{\max}$.

输出: 初始分配的容器数 x_{init}.

1. f_i=*runtimePredictModel*(*Job*,*i*) //建立各个超步的运行时间预测模型
2. $f_{total}=\sum_i f_i$ //f_{total}: 所有超步的运行时间预测值的算术和
3. $X=\{x_{\min},\ldots,x_{\max}\}$ //*X*: 系统及配置文件中允许的初始容器集
4. *availSet*=$\{x\in X|f_{total}(x,scale)<C\}$ //筛选 *X* 中符合运行时限 *C* 的元素组成有效容器集 *availSet*
5. **If** *availSet*≠∅ **Then**
6. x_{init}=min(*availSet*) //若 *availSet* 不为空, 则选择其中最小的容器数
7. **Else**
8. $x_{init}=\text{argmin}_{x\in X}(f_{total}(x,scale))$ //否则, 选择 *X* 中使 f_{total} 最小的容器数
9. **End If**
10. *setInitResource*(*Job*,x_{init})
11. **Return** x_{init}

当迭代作业以初始分配的资源开始执行后, Flink 的迭代数据流会在超步间的同步屏障处为某些特定的算子(例如 Join 和 Reduce 等)进行数据同步. 在这些同步屏障处, 前一个超步的任务管道已结束, 新的任务管道尚未开始, 所以没有算子的状态需要在此处迁移, 仅需对产生的中间结果数据进行 shuffle 处理, 因此可在此处调整迭代作业的资源分配[26], 来让 RABORP 策略适应迭代过程中因集群状态等原因造成的变化.

在超步间的同步屏障处, RABORP 策略通过比较剩余超步的运行时间预测值之和与时限要求的剩余运行时间(即总时限减去迭代作业已运行的时间)来进行动态调整, 如果当前的资源分配情况已无法满足运行时限, 就向上搜索能够满足要求的最小容器数; 如果剩余超步运行时间预测值之和大大低于运行时限, 则向下搜索更小的容器数来进行资源释放; 如果以上两种情况下均找不到合适的容器数, 那么就维持现状继续执行.

算法 2 描述了在同步屏障处调整资源分配的过程. 相比于算法 1, 算法 2 将历史超步的真实数据加入运行时间预测模型的训练数据集中, 而且采用相对松弛度 sk_{rt} 和绝对松弛度 sk_{at} 来对运行时限进行拓展, 从而尽可能地降低因不必要的调整或者搜索造成的巨大时间开销, 只有预测的剩余运行时间在触发资源动态调整的时间范围内, 即满足 $time>(C-T)\times(1+sk_{rt})+sk_{at}$ 或 $time<(C-T)\times(1-sk_{rt})-sk_{at}$ 时, 才在可用容器范围内进行类似于算法 1 的搜索来满足拓展后的运行时限.

算法 2. 迭代作业在同步屏障处的资源分配调整算法.

输入: 运行中的迭代作业 *Job*, 作业运行时限 C, 最小容器数 $x_{\min}$, 最大容器数 $x_{\max}$, 最近的超步α_*, 当前超步的数据规模 *scale*, 当前容器数 x_{now}, 已运行时间 T, 相对松弛度 sk_{rt}, 绝对松弛度 sk_{at}.

输出: 调整后的容器数 x_{adapt}.

1. f_i=*runtimePredictModel*(*Job*,*i*,α_*) //建立α_*后各个超步的运行时间预测模型
2. $f_{remain}=\sum_{i>\alpha_*} f_i$ //f_{remain}: 剩余超步的运行时间预测值的算数和
3. *time*=$f_{remain}(x_{now})$ //*time*: 该作业在容器数 x_{now} 下的剩余时间的预测值
4. **If** *time*>(C−T)×(1+sk_{rt})+sk_{at} **Then**
5. $X_{>now}=\{x_{now}+1,\ldots,x_{\max}\}$ //$X_{>now}$: 初始容器集 X 中大于当前容器数的子集
6. //筛选 $X_{>now}$ 中符合拓宽后的运行时限的元素组成有效集 *availSet*
7. *availSet*=$\{x\in X_{>now}|f_{remain}(x,scale)<(C-T)\times(1+sk_{rt})+sk_{at}\}$
8. **If** *availSet*≠∅ **Then**
9. x_{adapt}=min(*availSet*) //若 *availSet* 不为空, 则选择其中最小的容器数
10. **Else**
11. $x_{adapt}=\arg\min_{x\in X_{>now}}(f_{remain}(x,scale))$ //否则, 选择 $X_{>now}$ 中使 f_{remain} 最小的容器数
12. **End If**
13. **Else If** *time*<(C−T)×(1−sk_{rt})-sk_{at} **Then**
14. $X_{<now}=\{x_{\min},\ldots,x_{now}-1\}$ //$X_{<now}$: 初始容器集 X 中小于当前容器数的子集
15. //筛选 $X_{<now}$ 中符合拓宽后时限的元素组成有效集 *availSet*
16. *availSet*=$\{x\in X_{<now}|f_{remain}(x,scale)>(C-T)\times(1-sk_{rt})-sk_{at}\}$
17. **If** *availSet*≠∅ **Then**
18. x_{adapt}=min(*availSet*) //若 *availSet* 不为空, 则选择其中最小的容器数
19. **Else**
20. $x_{adapt}=x_{now}$ //否则, 维持当前状态不变
21. **End If**
22. **End If**
23. *setTempResource*(*Job*,x_{adapt})
24. **Return** x_{adapt}

4.2 算法局限性

由于第 3 节介绍的运行时间预测模型的训练数据点主要来自历史迭代作业的执行结果, 因此无法保证一定有足够的数据来进行训练, 而且对于收敛参数会随着数据集规模的变化而进行改变的迭代算法, 例如 PageRank, 使用抽样执行的前提是需要找到合适的参数缩放规则, 但是在短时间内针对不同的抽样比进行缩放是很困难的, 如果消耗时间太长又会失去进行运行时间预测的意义. 而对于回归模型来说, 至少要有 2 个数据点才能够拟合出一条曲线, 所以可从以下两种情况来考虑: (1) 训练数据为 0; (2) 有一个训练数据. 算法 3 显示了在训练数据不足时, 如何进行合理地初始资源分配.

算法 3. 训练数据不足时的初始资源分配算法.

输入: 当前提交的迭代作业 *Job*, 运行时限 C, 最小容器数 $x_{\min}$, 最大容器数 $x_{\max}$.

输出: 初始分配的容器数 x_{init}.

1. *num*=*collecData*(*Job*).*count*(·) //获取训练集数据量
2. $X=\{x_{\min},\ldots,x_{\max}\}$
3. **If** *num*==0 **Then** //出现冷启动现象

4. x_{init}=max(*availSet*) //默认向作业提供最大可用容器数
5. **Else If** *num*==1 **Then** //根据唯一的历史执行消耗的时间来判定
6. **If** *collecData*(*Job*).*runtime*(·)<*C* **Then**
7. $x_{init}=(x_{min}+x_{mean})/2$ //若上次该作业满足运行时限 *C*
8. **Else**
9. $x_{init}=(x_{max}+x_{mean})/2$ //若上次该作业未满足运行时限 *C*
10. **End If**
11. **End If**
12. *setInitResource*(*Job*,x_{init})

4.3 算法实现与部署

在 Flink on Yarn 框架中，标准的 Flink 迭代作业经过客户端提交，并将相关 Jar 包和配置上传至 HDFS，并向 Yarn 资源管理器注册并申请容器来启动 JobManager 节点上的 APP Master，然后，APP Master 根据存储在 HDFS 中的配置信息来为每个 TaskManager 节点分配容器，容器分配后就不再改变，最终多个 TaskManager 协同工作来完成每轮迭代任务直到算法收敛，每轮迭代的中间结果和状态通过 CheckPoint 机制存储在状态后端中，最终结果则输出到 HDFS 等 Data Sink 中.

为了实现 RABORP 策略，系统需要对 Yarn 的容器分配情况和超步的运行时间进行监控，并对各个超步的运行时间进行预测，然后根据预测结果制定并实施资源分配计划. 通过在 Flink on Yarn 框架的基础架构上添加 4 个必要组件，来进行 RABORP 策略的端到端实现. 图 8 显示了部署在 3 个节点下的 RABORP 原型系统的结构以及节点间的信息传递，这些组件分别运行在 Flink 集群不同的节点上来协同工作，具体功能如下.

(1) 时间预测组件. 根据 HDFS 上存储的历史执行信息，建立并训练运行时间预测模型，并在作业提交和同步时，对后续超步的运行时间进行预测.

(2) 资源协调组件. 综合运行时间预测结果、系统配置以及用户期望的作业时限，然后运行动态资源分配算法来制定实时的资源分配计划，并传递给 APP Master 进行实施.

(3) 时间监控组件. 在数据同步完成后，监控当前超步的端对端运行时间并发送至 TaskManager，然后通过“心跳”机制将该信息传递给 JobManager.

(4) 资源监控组件. 监控各个超步执行时 YARN 分配的容器数以及所处理的数据规模，并将统计信息发送至 TaskManager，然后通过“心跳”机制将该信息传递给 JobManager.

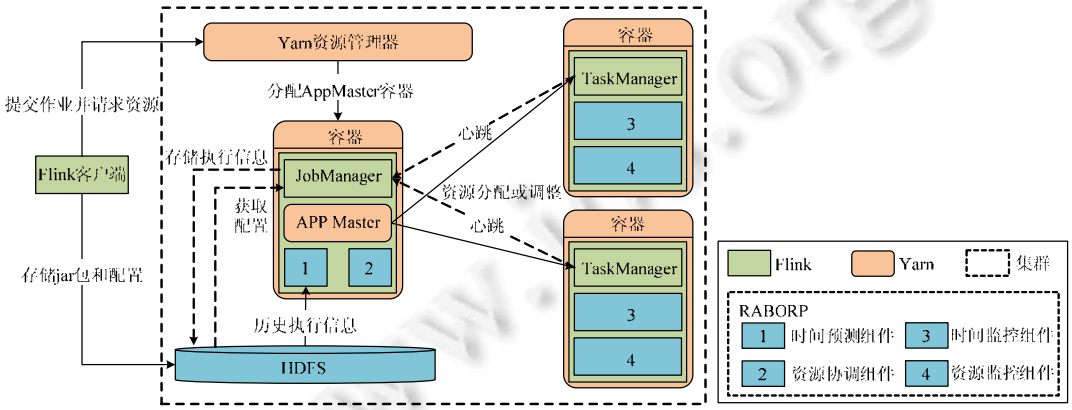


图 8 基于 Flink on Yarn 框架的 RABORP 原型系统的组成结构

5 实验与分析

本节首先介绍实验的环境和方法, 然后进行对比实验来对所提出的方法和其他方法进行评估和比较.

5.1 实验设置

(1) 实验环境

实验所描述的相关技术细节均在 Flink 1.8.0 中进行实现, 采用 Java 进行编码. 所使用的 Flink 系统部署在由 21 个物理节点构成的大规模分布式集群上, 包括 1 个部署 JobMananger 的主节点和 20 个部署 TaskManager 的从节点, 节点之间通过万兆以太网连接. 每个节点采用的硬件配置和相关软件配置参数分别见表 1 和表 2.

表 1 节点硬件配置参数

配置项	配置参数
CPU	Intel Xeon Gold 5118@2.30 GHz*2
内存	256 GB DDR3 2 400 MHz
硬盘	400 GB SSD
网络	10 Gb/s

表 2 节点软件配置参数

配置项	配置参数
操作系统	Centos 7.1
JDK 版本	Jdk1.8.0-181-Linux-x64
Flink 版本	1.8.0
Hadoop&Yarn 版本	2.7.3

我们通过在主节点上建立多个内存和磁盘共享的系统用户进行作业提交来模拟多任务实验环境, 除此之外, 实验中所有的图和非图迭代作业分别采用 Flink-Gelly 库和 Flink 基础迭代算子来进行实现, 实验默认资源调整算法中的相对松弛度 sk_{rt}=0.05, 绝对松弛度 sk_{at}=15s.

(2) 对比算法

在综合大量相关工作的基础上, 选择以下 2 种动态资源分配算法与所提出的 RABORP 策略进行性能对比, 包括通用算法 IBuml[21]和与 Flink 系统集成的算法 Flink-QoS[25].

- IBuml 与本文的解决方法密切相关, 均通过捕捉历史作业的执行特征来对资源供应进行优化, 但是它采用更加细粒度的资源消耗信息建立多项式回归模型来进行在线训练, 而且改进作业调度方式来达到自适应资源分配的目的.
- Flink-QoS 和本文的解决方法均将用户对 Flink 作业性能的约束纳入资源调整的过程中, 不同于本文引入的对作业运行时间的明确限制, 该算法是通过将用户在不同作业下对 QoS 的要求为第一目标, 来对所分配的 CPU 和内存资源进行调整.

(3) 评价指标

对于运行时间预测方法的相关实验, 使用相对平均误差(relative average error, RAE)作为其评价指标, 并与其他方法进行对比. 令 n 为预测的总次数, 其计算方法如公式(5)所示.

$$\text{相对平均误差(RAE)}=\frac{1}{n}\sum_{i=1}^{n}\frac{|\text{实际值}_i-\text{预测值}_i|}{\text{实际值}_i}\times 100\% \tag{5}$$

对于动态资源分配策略的相关实验, 分别使用 3 个通用评价指标: 运行时间、吞吐量、CPU 利用率和计算方法如第 5.4 节中表 6 所示的 3 个自定义评价指标: 资源使用量、时限违反次数、时限违反幅度.

5.2 数据集

针对以 3 种典型迭代算法为核心的 Flink 作业, 使用了多种数据集, 包括 5 种真实数据集和 1 种模拟数据集, 具体信息见表 3.

- 真实数据集

LiveJournal 数据集是来自 LiveJournal 在线社区的用户之间的友谊关系数据; UK-2006 数据集是人工标识过的网页是否含有垃圾信息特征的大型垃圾网站数据集合, 其中包含页面内容和跳转链接信息; WikiPedia 数据集是英语维基百科的百科全书页面图. 以上 3 个数据集来自 KONECT 图数据集[51]. Friendster 数据集中包含一个真实社交网站中用户间的友谊和多于 3 人的群组关系; RoadNet-CA 数据集来自加利福尼亚州的一个道路网络数据, 包括交叉口和道路起始点信息. 以上 2 个数据集来自斯坦福大型网络数据集[52].

- 模拟数据集

Points 数据集是通过从 5 个高斯混合模型中进行抽样，来生成的一系列不同规模的二维数据集，具有随机聚类中心和相等的变异概率. 由于 *K*-means 算法所处理的数据关系比较简单，收敛速度较快，所以采用模拟数据集来时其保持较长时间运行；而对于其他两种迭代作业，则使用真实数据集来保证实验效果的一般性.

表 3 Benchmark 作业及其数据集

迭代作业	简称	数据集	大小(GB)	参数
PageRank	PR	WikiPedia Friendster LiveJournal UK-2006	5.74 31.16 10 4.7	*d*=0.85, 150 iterations
ConnectedComponent	CC	LiveJournal WikiPedia Friendster RoadNet-CA	10 5.74 31.16 6.9	150 iterations
K-Means	–	Points-1 Points-2 Points-3	10 15 20	5 clusters, 150 iterations

5.3 运行时间预测的性能评估

- 正确性评估

为了评估在不同工作负载和训练数据下运行时间预测模型的正确性，通过对 3 种迭代作业运行时间预测模型的训练数据集进行随机抽样，构造了多种训练情况. 特别地，对于每个作业和特定训练数据点数量构成的子任务，分别使用参数回归、非参数回归和混合模型这 3 种方法进行预测，混合模型即采用交叉验证，在两种回归模型之间进行选择. 同时，对于每个子任务都进行了 50 轮抽样和预测过程，并使用 RAE 作为最终的测试指标.

图 9 中显示了随着训练数据点数量的增加，在不同迭代作业上非参数回归的效果都要逐渐优于参数回归；但是对于小规模的数据集来说，参数回归则优于非参数回归. 因此，为了综合参数回归的稳定性和非参数回归的灵活性，添加模型切换的优化机制至关重要. 混合模型通过使用交叉验证来进行模型选择，使得在不同的工作负载下的预测效果都能随着训练数据点数量的增加而提升.

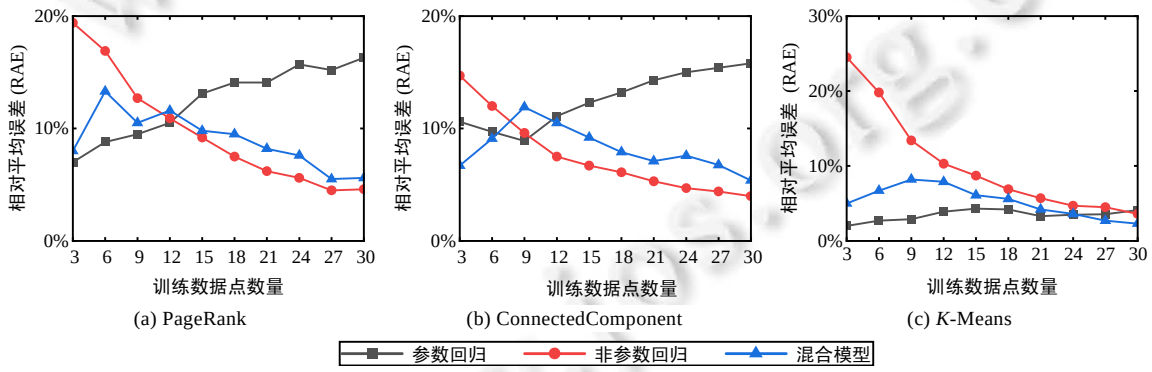
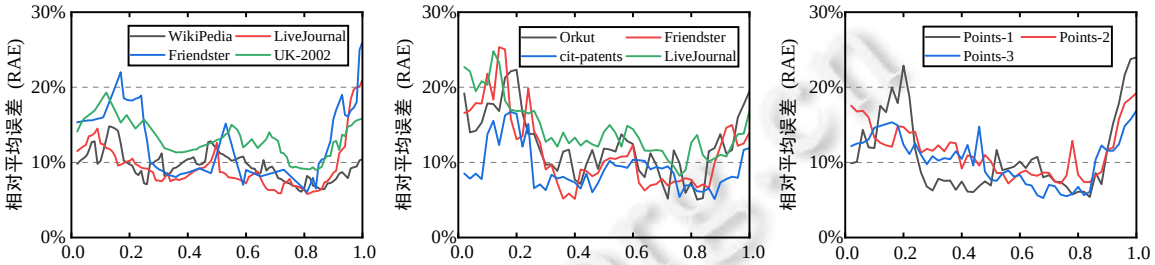


图 9 3 种运行时间预测方式在不同数量的训练点下的预测误差对比

- 准确性评估

本次实验采用相对迭代进度(relative iterative progress, RIP)来表示迭代作业的执行进程，RIP 是当前迭代次数与总迭代次数之比. 为了充分对所提出的运行时间预测模型的预测准确度进行评估，通过在作业进行到不同迭代次数时，对每个在不同数据集上执行的迭代作业构成的子任务都进行 50 次预测，来捕捉不同的工作负载和迭代次数对模型的影响，最终计算其 RAE 作为测试指标.

图 10 中显示了随着迭代任务的执行，不同工作负载下运行时间预测的 RAE 变化趋势. 可见，不同工作负载的 RAE 在 RIP 为 0.3%–0.7%之间时达到稳定，保持在 7.57%–13.21%之间，但是在迭代作业刚开始或者即将结束时的 RAE 明显高于此范围. 这种现象是由于迭代开始时，模型的训练数据基本为历史信息，然后能够反映该工作负载实际情况的真实数据开始用于模型训练，故 RAE 开始下降；在迭代即将结束时，不同超步间处理的数据规模下降很快，导致其运行时间差异较大所造成 RAE 上升，但迭代后期超步的绝对运行时间都很短，所以对于判断是否达到运行时限的影响不大. 除此之外，使用不同的数据集并没有对其造成很大的影响，因为除了由于数据类型和规模造成的 RAE 不同以外，3 种迭代作业的 RAE 变化趋势基本保持一致.



(a) PageRank (b) ConnectedComponent (c) *K*-Means

图 10 不同迭代进度(RIP)下各个迭代作业运行时间的预测误差对比

5.4 动态资源分配的性能评估

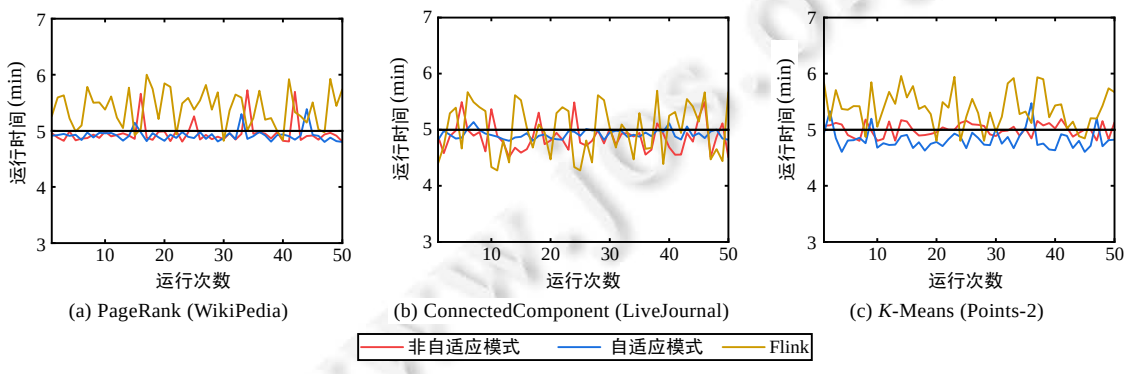
- 有效性评估

为了对 RABORP 策略整体和动态调整等方面的有效性进行全面评估，实验在单作业场景下分别评测了 RABORP 策略的两种模式.

(1) 自适应模式：在超步间的同步屏障处对初始资源分配进行调整.

(2) 非自适应模式：仅在作业提交时进行初始资源分配而不进行后续调整.

实验将 3 种迭代作业分别在上述两种模式和 Flink 原系统下执行 50 次来进行效果对比，图 11 显示了它们的运行时间的对比，图中水平黑线为运行时限. 可见，采用非自适应模式的稳定性较差，在部分情况下的运行效果甚至要低于 Flink 原系统. 在非自适应模式出现了对资源进行过度供应的现象，即作业运行时间多次出现远低于运行时限的现象. 而在自适应模式下，迭代作业高于运行时限的峰值次数则远少于 Flink 原系统，而且能够维持在运行时限附近，说明在迭代作业执行过程中进行资源分配的动态调整是有效且十分有必要的.



(a) PageRank (WikiPedia) (b) ConnectedComponent (LiveJournal) (c) *K*-Means (Points-2)

图 11 不同资源分配模式在单作业场景下的运行时间对比

- 性能提升对比

实验在 RABORP 系统、Flink 原系统和两个 baseline 系统下，采用相同的集群配置和其默认参数来分别在

单作业和多作业场景下执行作业，并对比 3 种 benchmark 作业在不同场景下各项评价指标的平均值.

PageRank 程序主要用于计算网页排名，是迭代计算领域应用最广泛的 benchmark 作业. 在单作业场景下，PageRank 作业的平均运行时间和吞吐量对比如图 12 所示，使用 RABORP 策略时作业的运行时间均低于其他 3 种方法，而且除了 *RIP*=0.4 时出现了吞吐量的明显上升，稳定性比两种 baseline 方法要差以外，但 RABORP 策略的吞吐量在整个迭代过程中均高于两种 baseline 方法. 图 13 显示了在不同迭代次数下，CPU 利用率的变化. RABORP 原型系统的 CPU 利用率基本高于其他 3 种系统，保持在 60%–80%之间，但是在 *RIP*=0.4 时，CPU 利用率明显降低. 结合图 12 中的信息可以判断，出现了资源过度供应现象. 但在整体迭代过程中均高于 Flink 原系统，说明 RABORP 策略已经有效地解决了 Flink 中存在的因资源过度供应导致的资源利用率低的现象.

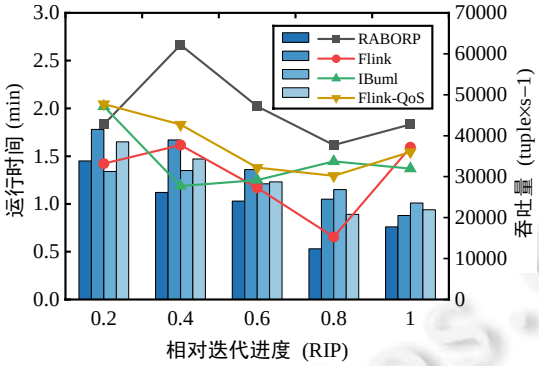


图 12 不同资源分配技术下 PageRank 作业的运行时间和吞吐量对比

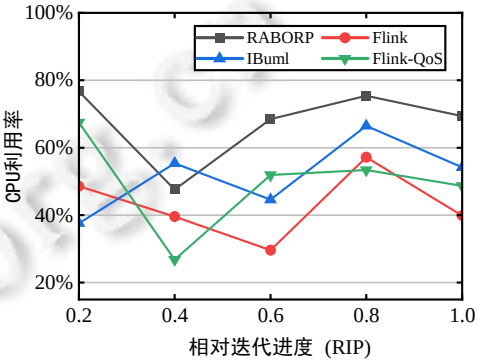


图 13 不同资源分配技术下 PageRank 作业的 CPU 利用率对比

为了评估 RABORP 策略在多个并发作业互相干扰下的性能，我们构造了如表 4 所示的多种多作业实验场景，同一场景下的迭代作业具有不同的开始时间 *T* 和运行时限执行 *C*，其中，*T* 是满足[0,100]的随机整数，*C* 是满足[5,15]的随机整数，以确保跨作业的干扰是不可预测的.

表 4 多作业执行场景

场景 ID	作业	开始时间(s)	数据集	运行时限(min)
Multi#1	PR	2	WikiPedia	7
	CC	35	LiveJournal	12
	K-Means	87	Points-1	8
Multi#2	CC	3	WikiPedia	14
	K-Means	11	Points-2	6
	PR	40	UK-2006	8
Multi#3	*K*-Means	55	Friendster	5
	PR	7	RoadNet-CA	13
	CC	43	Points-3	7

表 5 为 benchmark 作业在不同场景下各项性能评估指标的平均值对比结果，可见，虽然 RABORP 策略在多作业场景下受并发作业间资源抢占的干扰，使得在 3 种指标下的效果低于单作业场景，但是相比于 Flink 原系统和其他两种 baseline 算法，在两种场景下均能够有效降低迭代作业的执行时间，并提高迭代作业执行时的吞吐量和 CPU 利用率. 具体来看，相比于 RABORP 策略，虽然 Ibuml 算法能够大幅度降低作业执行时间，但在多作业场景下受作业执行顺序影响较大，性能不稳定；而 Flink-QoS 虽然在两种场景下性能较为稳定，但由于其资源调整粒度较粗且时间开销较大，使得性能优化幅度均低于 RABORP 策略.

除此之外，为了对 RABORP 策略针对 Flink 原系统的性能提升进行全面评估，使用了 3 个自定义的评价指标. 令 x_i 表示第 i 个超步使用的容器数量，y_i 表示相应的运行时间，C 代表迭代作业的运行时限，这 3 个指标的计算公式见表 6.

表 5 单/多作业场景下, 不同 benchmark 作业的平均性能评估指标对比

作业	场景	Baselines	执行时间(min)	吞吐量(*w*×TPS)	CPU 利用率(%)
PR	Single-Avg	RABORP	5.83	3.76	73.57
		Flink	12.37	2.53	53.42
		IBuml	**4.79**	4.17	92.74
		Flink-QoS	8.52	4.36	74.57
	Multi-Avg (#1/2/3)	RABORP	**6.17**	2.93	84.94
		Flink	18.32	1.75	77.31
		IBuml	7.13	3.87	90.24
		Flink-QoS	9.54	3.52	80.62
CC	Single-Avg	RABORP	**12.79**	2.97	82.52
		Flink	19.47	2.35	42.54
		IBuml	13.23	2.15	85.03
		Flink-QoS	17.65	2.72	69.37
	Multi-Avg (#1/2/3)	RABORP	**14.51**	2.57	79.17
		Flink	25.74	1.95	57.42
		IBuml	27.45	1.85	63.82
		Flink-QoS	19.21	2.52	72.65
K-Means	Single-Avg	RABORP	**7.35**	5.27	83.45
		Flink	15.27	3.23	53.57
		IBuml	13.45	4.53	85.74
		Flink-QoS	9.27	4.27	79.35
	Multi-Avg (#1/2/3)	RABORP	**7.91**	4.14	85.87
		Flink	18.93	2.15	60.92
		IBuml	15.73	3.42	79.65
		Flink-QoS	11.35	3.77	74.92

表 6 自定义评价指标及其计算公式

指标	计算公式	符号
资源使用量	$\sum_i (x_i \times y_i)$	RS
时限违反次数	$\sum_{i\in\{i\mid y_i>C\}} 1$	TVC
时限违反幅度	$\sum_{i\in\{i\mid y_i>C\}} (y_i - C)$	TVM

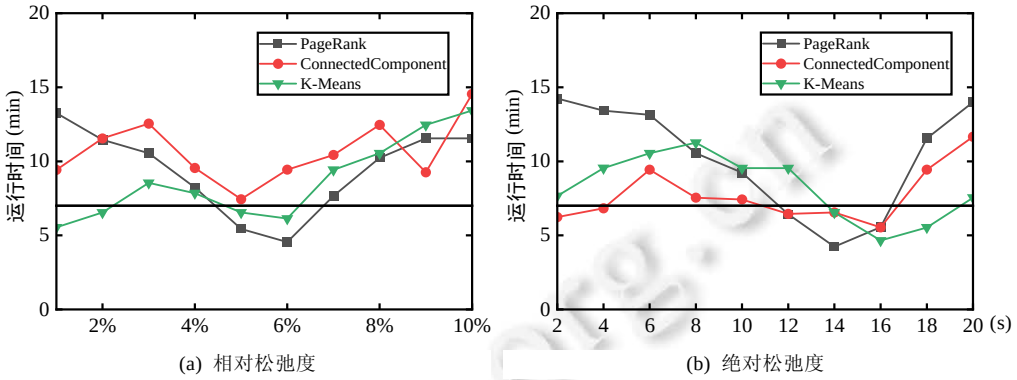
通过获取上述实验中的统计信息, 分别计算了 3 种迭代作业在 RABORP 系统和 Flink 原系统下的自定义指标, 并通过 RABORP 系统下的指标除以在 Flink 原系统下的相应指标, 然后进行算术平均得到了自定义指标比率. 表 7 总结了 3 种 benchmark 作业在单作业和多作业场景下的 3 种评价指标的比率, 在单作业和多作业场景下使用 RABORP 策略后, 每个作业的时限违反数量和幅度均得到了有效减少, 而且对于使用数据集规模最大的 *K*-Means 作业, 也能够有效降低其资源使用量, 但是对于 PageRank 作业来说, 时限违反数量和幅度的降低是以高资源使用量为代价的. 总体来看, 除了 PageRank 以外的其他 2 种作业的资源使用量, 相对于 Flink 原系统都有所降低, 而且在单作业场景下, RABORP 策略相比于 Flink 原系统减少了 55.2%–70.3%的时限违反次数和 72.3%–90.2%的时限违反幅度; 在多作业场景下, 则减少了 27.7%–74.9%的时限违反次数和 64.9%–80.5%的时限违反幅度. 可以说明, RABORP 策略有效地解决了 Flink 原系统对迭代作业的资源分配不均衡的问题.

表 7 不同作业的自定义评价指标比率 (%)

场景	迭代作业	RS ratio	TVC ratio	TVM ratio
Single-job	PageRank	125.37	44.76	15.43
	CC	79.38	37.32	27.69
	K-Means	**57.39**	**29.62**	**9.85**
Multi-job	PageRank	109.21	72.34	35.15
	CC	89.29	**25.12**	**19.51**
	K-Means	**69.27**	45.93	21.32

5.5 算法参数设置

RABORP 策略在超步间同步屏障处判断当前资源配置是否需要进行调整时，采用相对松弛度 sk_{rt} 和绝对松弛度 sk_{at} 来对剩余运行时限进行拓展，合理的参数配置能够避免不必要的调整造成的时间开销，从而增强 RABORP 策略的稳定性. 通过分析不同松弛度的 RABORP 策略对各种迭代作业运行时间的影响来确定最优松弛度，实验结果如图 14 所示，图中水平直线代表运行时限为 7 分钟.



(a) 相对松弛度 (b) 绝对松弛度

图 14 不同松弛度参数下各个迭代作业的运行时间对比

从图 14 中可以很明显地看到，不同松弛度参数下的 RABORP 策略对迭代作业的运行时间影响很大. 最终选择能够使作业运行时间保持在运行时限附近的参数值作为最优参数值，故确定最优相对松弛度 sk_{rt} 为 0.05，最优绝对松弛度 sk_{at} 为 15s.

6 总结与展望

针对 Flink 难以按合理的资源配置使迭代算法按时收敛的问题，提出了面向 Flink 迭代作业的动态资源分配策略 RABORP，并基于 Flink on Yarn 框架实现了该策略的原型系统. 该策略的核心是一个混合的面向迭代超步的运行时间预测模型，通过以预测的运行时间为导向，分别在迭代作业提交和超步间的每个同步屏障处进行资源的初始分配和调整，来达到对 Flink 迭代作业资源的动态分配，从而在用户期望的时间内完成作业执行. 经过大量的对比实验，证实所提出的运行时间预测模型和动态资源分配策略是十分有效的，而且对于 Flink 原系统的性能也有很大提升，有效解决了目前 Flink 系统针对迭代作业的资源分配不均衡的问题.

虽然 RABORP 策略已通过设置松弛度参数等方法来缓解因资源动态缩放引入的额外时间开销，但未来在动态调整资源分配时，使用更详细的效益模型来优化调整的时机和幅度，也是值得深入研究的内容.

References:

[1] Kang U, Tsourakakis CE, Faloutsos C. PEGASUS: A peta-scale graph mining system implementation and observations. In: Proc. of the 9th IEEE Int'l Conf. on Data Mining. IEEE, 2009. 229–238.

[2] Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012, 5(8): 716–727.

[3] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Nahum EM, Xu DY, eds. Proc. of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2010). USENIX Association, 2010.

[4] Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K. Apache Flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 38(4): 28–38.

[5] Chen CLP, Zhang CY. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Information Sciences, 2014, 275: 314–347.

[6] Jyothi SA, Curino C, Menache I, Narayanamurthy SM, Tumanov A, Yaniv J, Mavlyutov R, Goiri I, Krishnan S, Rao S, Kulkarni J. Morpheus: Towards automated slos for enterprise clusters. In: Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI). USENIX, 2016. 117−134.

[7] Tumanov A, Zhu T, Park JW, Kozuch MA, Harchol-Balter M, Ganger GR. TetriSched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Proc. of the European Conf. on Computer Systems. ACM, 2016. 1−16.

[8] Mishne G, Dalton J, Li Z, Sharma A, Lin J. Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2013. 1147−1158.

[9] Wolf J, Rajan D, Hildrum K, Khandekar R, Kumar V, Parekh S, Wu KL, Balmin A. FLEX: A slot allocation scheduling optimizer for MapReduce workloads. In: Proc. of the CM/IFIP/USENIX Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing. Springer, 2010. 1−20.

[10] Morton K, Friesen A, Balazinska M, Grossman D. Estimating the progress of MapReduce pipelines. In: Proc. of the 26th IEEE Int'l Conf. on Data Engineering (ICDE). IEEE, 2010. 681−684.

[11] Dai MZ, Gao SF. Framework performance evaluation based on Hadoop, Spark and Flink large-scale data analysis. Journal of China Academy of Electronics and Information Technology, 2018, 13(2): 149−155 (in Chinese with English abstract).

[12] Kamburugamuve S, Wickramasinghe P, Ekanayake S, Fox GC. Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink. The Int'l Journal of High Performance Computing Applications, 2018, 32(1): 61−73.

[13] Akil B, Zhou Y, Rohm U. On the usability of Hadoop MapReduce, Apache Spark & Apache Flink for data science. In: Proc. of the IEEE Int'l Conf. on Big Data (Big Data). IEEE, 2017. 303−310.

[14] Ye F, Jing Z, Huang Q, Hu C, Chen Y. The research and implementation of a distributed crawler system based on Apache Flink. In: Proc. of the Int'l Conf. on Algorithms and Architectures for Parallel Processing. Springer, 2018. 11338: 90−98.

[15] Sun D, He H, Yan H, Gao S, Liu X, Zheng X. Lr-Stream: Using latency and resource aware scheduling to improve latency and throughput for streaming applications. Future Generation Computer Systems, 2021, 114: 243−258.

[16] Mortazavi-Dehkordi M, Zamanifar K. Efficient deadline-aware scheduling for the analysis of big data streams in public cloud. Cluster Computing, 2020, 23(1): 241−263.

[17] Li ZY, Yu J, Bian C, Pu YL, Wang YF, Zhang YT, Guo BL. Flink-ER: An elastic resource-scheduling strategy for processing fluctuating mobile stream data on Flink. Mobile Information Systems, 2020(4): No.5351824. [doi: 10.1155/2020/5351824]

[18] Agarwal S, Kandula S, Bruno N, Wu MC, Stoica I, Zhou J. Reoptimizing data parallel computing. In: Proc. of the 9th USENIX Symp. on Networked Systems Design and Implementation (NSDI). USENIX, 2012. 281−294.

[19] Ferguson AD, Bodik P, Kandula S, Boutin E, Fonseca R. Jockey: Guaranteed job latency in data parallel clusters. In: Proc. of the 7th ACM European Conf. on Computer Systems. ACM, 2012. 99−112.

[20] Delimitrou C, Kozyrakis C. Quasar: Resource-efficient and QoS-aware cluster management. ACM SIGPLAN Notices, 2014, 49(4): 127−144.

[21] Thamsen L, Verbitskiy I, Beilharz J, Renner T, Polze A, Kao O. Ellis: Dynamically scaling distributed dataflows to meet runtime targets. In: Proc. of the IEEE Int'l Conf. on Cloud Computing Technology and Science (CloudCom). IEEE, 2017. 146−153.

[22] Gaussier E, Glesser D, Reis V, Trystram D. Improving backfilling by using machine learning to predict running times. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. ACM, 2015. 1−10.

[23] Wang K, Khan MMH, Nguyen N. A dynamic resource allocation framework for Apache Spark applications. In: Proc. of the 44th IEEE Annual Computers, Software, and Applications Conf. (COMPSAC). IEEE, 2020. 997−1004.

[24] Lee J, Kim B, Chung JM. Time estimation and resource minimization scheme for Apache Spark and Hadoop big data systems with failures. IEEE Access, 2019, 7: 9658−9666.

[25] HoseinyFarahabady MR, Taheri J, Zomaya AY, Tari Z. A dynamic resource controller for resolving quality of service issues in modern streaming processing engines. In: Proc. of the 19th IEEE Int'l Symp. on Network Computing and Applications (NCA). IEEE, 2020. 1−8.

[26] Thamsen L, Renner T, Kao O. Continuously improving the resource utilization of iterative parallel dataflows. In: Proc. of the 36th IEEE Int'l Conf. on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2016. 1−6.

[27] Praveenchandar J, Tamilarasi A. Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing. Journal of Ambient Intelligence and Humanized Computing, 2021, 12(3): 4147−4159.

[28] Belgacem A, Beghdad-Bey K, Nacer H, Bouznad S. Efficient dynamic resource allocation method for cloud computing environment. Cluster Computing, 2020, 23(4): 2871−2889.

[29] Herodotou H, Lim H, Luo G, Borisov N, Dong L, Cetin FB, Babu S. Starfish: A self-tuning system for big data analytics. In: Proc. of the the 5th Int'l Conf. on Innovative Data Systems Research (CIDR). 2011. 261−272.

[30] Morton K, Balazinska M, Grossman D. ParaTimer: A progress indicator for MapReduce DAGs. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2010. 507−518.

[31] Wang K, Khan MMH. Performance prediction for apache spark platform. In: Proc. of the 17th IEEE Int'l Conf. on High Performance Computing and Communications. IEEE, 2015. 166−173.

[32] Al-Sayeh H, Hagedorn S, Sattler KU. A gray-box modeling methodology for runtime prediction of apache spark jobs. Distributed and Parallel Databases, 2020, 38(4): 819−839.

[33] Venkataraman S, Yang Z, Franklin M, Recht B, Stoica I. Ernest: Efficient performance prediction for large-scale advanced analytics. In: Proc. of the 13th USENIX Symp. on Networked Systems Design and Implementation (NSDI). USENIX, 2016. 363−378.

[34] Popescu AD, Balmin A, Ercegovac V, Ailamaki A. PREDIcT: Towards predicting the runtime of large scale iterative analytics. Proc. of the VLDB Endowment, 2013, 6(14): 1678−1689.

[35] Koch J, Thamsen L, Schmidt F, Kao O. SMiPE: Estimating the progress of recurring iterative distributed dataflows. In: Proc. of the 18th Int'l Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT). IEEE, 2017. 156−163.

[36] Lama P, Zhou X. AROMA: Automated resource allocation and configuration of MapReduce environment in the cloud. In: Proc. of the 9th Int'l Conf. on Autonomic Computing (ICAC). ACM, 2012. 63−72.

[37] Guo Y, Shan H, Huang S, Hwang K, Fan J, Yu Z. GML: Efficiently auto-tuning flink's configurations via guided machine learning. IEEE Trans. on Parallel and Distributed Systems, 2021, 32(12): 2921−2935.

[38] HoseinyFarahabady MR, Jannesari A, Taheri J, Bao W, Zomaya AY, Tari Z. Q-Flink: A QoS-aware controller for Apache Flink. In: Proc. of the 20th IEEE Int'l Symp. on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 2020. 629−638.

[39] Jalaparti V, Ballani H, Costa P, Karagiannis T, Rowstron A. Bridging the tenant-provider gap in cloud services. In: Proc. of the 3th ACM Symp. on Cloud Computing (SOCC). ACM, 2012. 1−14.

[40] Thamsen L, Verbitskiy I, Schmidt F, Renner T, Kao O. Selecting resources for distributed dataflow systems according to runtime targets. In: Proc. of the 35th IEEE Int'l Conf. on Computing and Communications Conf. (IPCCC). IEEE, 2016. 1−8.

[41] Kishor A, Niyogi R. An evolutionary approach for optimal multi-objective resource allocation in distributed computing systems. Concurrent Engineering, 2020, 28(2): 97−109.

[42] Verma A, Cherkasova L, Campbell RH. Aria: Automatic resource inference and allocation for MapReduce environments. In: Proc. of the 8th ACM Int'l Conf. on Autonomic Computing. ACM, 2011. 235−244.

[43] Rajan K, Kakadia D, Curino C, Krishnan S. PerfOrator: Eloquent performance models for resource optimization. In: Proc. of the 11th ACM Symp. on Cloud Computing. ACM, 2016. 415−427.

[44] Chen Y, Lu J, Chen C, Hoque M, Tarkoma S. Cost-effective resource provisioning for Spark workloads. In: Proc. of the 28th ACM In: Proc. of the Information and Knowledge Management. ACM, 2019. 2477−2480.

[45] Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: A system for large-scale graph processing. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2010. 135−146.

[46] Gonzalez JE, Low Y, Gu H. Powergraph: Distributed graph-parallel computation on natural graphs. In: Proc. of the 10th USENIX Symp. on Operating Systems Design and Implementation (OSDI). USENIX, 2012. 17−30.

[47] Ewen S, Tzoumas K, Kaufmann M, Markl V. Spinning fast iterative data flows. Proc. of the VLDB Endowment, 2012, 5(11): 1268−1279.

[48] Iterative graph processing in Apache Flink. 2019. https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/libs/gelly

[49] Chowdhury M, Stoica I. Coflow: A networking abstraction for cluster applications. In: Proc. of the 11th ACM Workshop on Hot Topics in Networks. ACM, 2012. 31−36.

[50] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E. Apache Hadoop Yarn: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. ACM, 2013. 1–16.

[51] The KONECT project. 2021. http://konect.cc/

[52] Stanford large network dataset collection. 2021. http://snap.stanford.edu/data/

附中文参考文献:

[11] 代明竹, 高嵩峰. 基于 Hadoop, Spark 及 Flink 大规模数据分析的性能评价. 中国电子科学研究院学报, 2018, 13(2): 149–155.

岳晓飞(1998－), 男, 硕士生, CCF 学生会员, 主要研究领域为大数据管理, 分布式计算.

史岚(1964－), 女, 博士, 副教授, 主要研究领域为计算机体系结构, 网络信息安全.

赵宇海(1975－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据库, 数据挖掘, 机器学习, 软件工程, 生物信息学.

季航旭(1990－), 男, 博士生, CCF 学生会员, 主要研究领域为分布式计算, 网络表示学习.

王国仁(1966－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为不确定数据管理, 数据密集型计算, 可视媒体数据分析管理, 非结构化数据管理, 分布式查询处理与优化, 生物信息学.