面向状态可变数据流的集群调度综述

许源佳[1)] 吴恒[2)] 杨晨[1)] 吴悦文[1)2)] 张文博[2),3)] 王焘[2)3)]

[1)](中国科学院大学 北京 100190)

[2)](中国科学院软件研究所 软件工程技术研究开发中心 北京 100190))

[3)](中国科学院软件研究所 计算机科学国家重点实验室 北京 100190)

摘要 状态可变数据流（Mutable States Data Flow，MS-DF）是机器学习系统运行时的主要特征，MS-DF 可由有向图来表示，其顶点由算子构成，表示机器学习运算逻辑；边代表算子之间的输入输出依赖关系。MS-DF 的集群调度是保障机器学习系统高效运行的主要工作，如何高效进行 MS-DF 的集群调度已经成为机器学习的研究热点。其中，机器学习系统（TensorFlow，PyTorch 等）作为中间层解耦了机器学习运算逻辑和资源分配（CPU，GPU，FGPA），从而机器学习无需再“独占式”静态绑定资源，而是由机器学习系统运行时动态管理，而算子是该解耦过程的关键要素，这给 MS-DF 的集群调度带来了新的挑战，这些挑战主要由算子资源需求刻画的准确性、算子调度决策的适应性和算子调度调整的差异性这三方面导致的。首先介绍算子资源需求的感知、协同两个机制，以克服多种算子组合导致其自身资源需求难以准确刻画的挑战；然后，通过决策约束、决策模型和决策求解来介绍算子调度决策，以应对算子状态频繁变化带来的适应性挑战；接着，介绍迁移、伸缩、挂起恢复等算子调度调整策略，以适用于不同算子状态同步方式带来的差异性挑战。最后，基于上述三个挑战，对近年来的集群调度最新研究成果进行归纳和分析，并展望 MS-DF 的集群调度，指出算子异构资源需求多层次分析及协同刻画、算子复杂调度约束的灵活定义和发现、学习驱动的算子低成本调度调整技术是其主要发展方向。

关键词 机器学习系统，状态可变数据流，机器学习算子，算子资源需求刻画，算子调度决策，算子调度调整

中图法分类号 TP311

State-of-the-Art Survey of Cluster Scheduling for Mutable States Data Flow

XU Yuanjia[1)] WU Heng[2)] YANG Chen[1)] WU Yuewen[1)2)] ZHANG Wenbo[2)3)] WANG Tao[2)3)]

[1)](University of Chinese Academy of Sciences, Beijing 100190)

[2)](Software Engineering Technology Research And Development center, Institute of Software, Chinese Academy of Science, Beijing 100190)

[3)](State key laboratory of computer science, Institute of Software, Chinese Academy of Science, Beijing 100190)

Abstract Mutable States Data Flow (MS-DF), as a main runtime feature of machine learning systems (e.g. Tensorflow, Pytorch, MxNet), can be represented by a directed graph. Here, each vertex in a MS-DF graph denotes a single operation (e.g. Conv2D, MatMul) which consists of typical machine learning computing processes. And each edge connecting two operations denotes the dependency of these two operations, the term “dependency” means the output of an operation is the input of the other operation linked by an edge. Currently, cluster scheduling for MS-DF is one of the main works that can guarantee the execution efficiency of machine learning systems, and it is one of the hot research topics in machine learning system area. Diving into the principle of cluster scheduling for MS-DF, machine learning systems are key factors that affect the performance

本课题得到国家重点研发计划(2018YFB1003602)、国家自然科学基金(61872344)、北京市自然科学基金(4182070)、中科院青促会人才专项(2018144)资助，以及阿里巴巴2018年度创新研究（AIR）项目的支持.许源佳，博士研究生,主要研究领域为资源调度、分布式系统和机器学习系统.E-mail: xuyuanjia2017@otcaix.iscas.ac.cn. 吴恒，博士，副研究员，CCF会员（64713M）,主要研究领域为容器虚拟化、边缘计算.E-mail: wuheng@otcaix.iscas.ac.cn. 杨晨，硕士研究生，无，否,主要研究领域为资源调度、分布式系统.E-mail: yangchen19@otcaix.iscas.ac.cn. 吴悦文，博士研究生，无，否,主要研究领域为性能建模，基于机器学习的云配置推荐.E-mail: wuyuewen11@otcaix.iscas.ac.cn. 张文博（**通信作者**），博士，研究员，CCF会员（21043M）,主要研究领域为云计算、服务计算.E-mail: zhangwenbo@otcaix.iscas.ac.cn. 王焘，博士，副研究员，CCF高级会员（39430S）,主要研究领域为微服务监测、服务计算.E-mail: wangtao08@otcaix.iscas.ac.cn

of cluster scheduling, since they work as a middle layer to decouple the computing of machine learning and cluster resource (e.g. CPU, GPU and FPGA) allocations. By this way, cluster resources are no longer exclusively and statically bound to one computation of machine learning. Instead, machine learning systems may manage different kinds of resources dynamically, but at the cost of increased complexity of cluster scheduling for MS-DF. Under this circumstance, machine learning operations can heavily affect the dynamic management of cluster resources, thus new challenges arise. We demonstrate that these challenges are caused by the following three aspects: (i) the accuracy of profiling operations' heterogeneous resource requirements. (ii) the adaptability of operation scheduling decisions. (iii) the variability of operation scheduling adjustments.

In addition to above challenges, we analyze and summarize the latest researches of cluster scheduling for MS-DF in recent years, and how these researches cope with challenges. (i) We introduce the mechanisms of both machine learning operation perception and cooperation that can be used to profile operation heterogeneous resource requirement, such mechanisms can estimate the actual operation resource usage through many metrics (e.g. input dataset sizes, operation dependencies), thus we can improve the accuracy of profiling especially when operations have many combinations. (ii) We introduce different scheduling decisions with three main factors including operation scheduling constraints, models and algorithms. Such decisions should be able to be deliberately set with different factors, thus we can adaptively provide operations with preferred resources when their states frequently change. (iii) We introduce several scheduling adjustment strategies to further improve the performance of cluster scheduling. These strategies include operation migration, scaling, and suspend/resume. One strategy or hybrid strategies can be useful under certain operation synchronization patterns (e.g. PS, AllReduce). Thus, we can use different strategies for different patterns. At last, we conclude latest researches with the new trend of machine learning systems and the development of heterogeneous resources. We give our prospective view of the key technologies of cluster scheduling for MS-DF: (i) the multi-level analysis and collaborative characterization of operations' heterogeneous resource requirements. (ii) the flexible definition and discovery of operations' complex scheduling constraints. (iii) learning-driven optimization of operation scheduling at a low cost. They may improve cluster scheduling for MS-DF in a more efficient and intelligent way.

Key words * Machine Learning System; Mutable States Data Flow; Machine Learning Operations; Profiling Operation Resource Requirement; Operation Scheduling Decision; Operation Scheduling Adjustment

1 引言

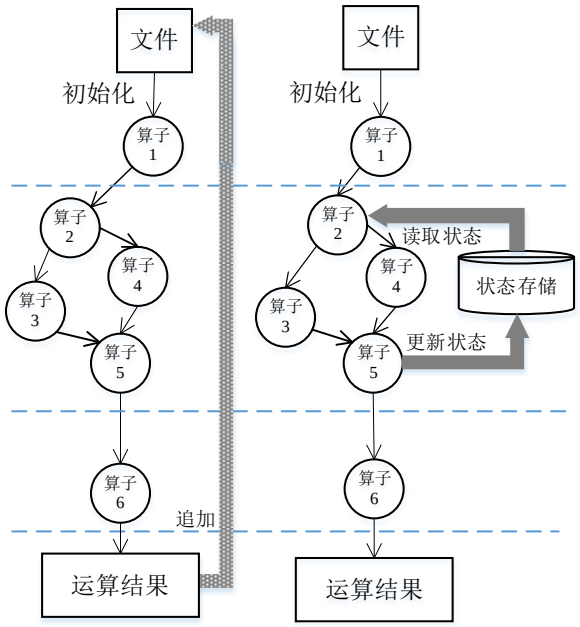
近年来，以 TensorFlow[1]、Pytorch[2]为代表的机器学习系统的成熟推动了机器学习应用的持续创新[3]。机器学习系统通过引入算子 **Error! Reference source not found.**（矩阵乘法、数据读写等）解耦了业务逻辑（语音、图像等机器学习应用）和异构资源（CPU、GPU 等）。而当前机器学习系统通常采用“尽力而为”的调度方法，往往给算子分配与自身实际需要不符的资源，且难以根据算子运行状态的变化动态调整资源供给。相关研究[4]初步表明，在异构资源上合理调度算子，对于优化机器学习应用运行性能，降低集群运维成本具有重要意义。

本文采用机器学习系统 TensorFlow[1]中的定义，将机器学习应用使用算子进行数据处理的过程抽象为状态可变数据流（Mutable States Data Flow，MS-DF）。如图 1 所示，与 Hadoop[5]、Spark[6]等传统数据流相比（图 1(a)），该类数据流增加了状态的读取和更新操作（图 1(b)），从而可以避免前者反复初始化带来的性能开销，能更好支持机器学习应用在集群环境下迭代运行的需要。

状态可变数据流使得算子的资源需求动态、持续地变化，这给集群调度带来了新的挑战。已有综述主要面向传统数据流（大数据分析[7]，工作流[8]，离线作业与在线服务[9]）在网格计算[10]、云计算[11]、私有集群[12][13]等环境下的集群调度，并未考虑资源需求的动态变化。针对上述问题，本文以状态可变数据流这一机器学习运行时特征为基础，梳理归纳了近年来机器学习和集群调度的最新成果，并对相关技术的挑战与发展方向进行了展望。

本文第 2 节围绕算子介绍了状态可变数据流的

运行原理、调度特点和主要挑战，并提出了研究框架。第 3 节从算子感知、协同两方面，讨论了调度需求发现机制，论述了如何动态刻画算子的资源需求；第 4 节基于决策约束，讨论了调度决策模型对算子的支持，论述了如何根据算子状态变化来供给资源；第 5 节讨论了迁移、伸缩、挂起和恢复等策略以及如何结合算子运行时特点进行调度调整；第 6 节对全文工作进行总结，并展望了未来的挑战与发展方向。



(b) 传统数据流　　(b) 状态可变数据流

图 1 两类数据流的对比

2 问题概述

2.1 集群调度目标

对状态可变数据流集合 F 中的任一个数据流 f_i，可通过其基本计算单元[14]算子及算子间的依赖关系来表示，如式(1)所示：

$$f_i = (OP_i, D_i) \tag{1}$$

式中，按照一定的算子组合方法 $C(f_i)$ [14]，得到数据流 f_i 的算子集合 OP_i 及算子依赖集合 D_i。其中，算子集合 OP_i 中的任意算子 $op_{i,j}$ 表示特定的数学运算或若干运算的组合，如卷积、矩阵乘法、求导、累加等；算子依赖集合 D_i 中的任一依赖 $d_{i,j,k} = op_{i,j} \to op_{i,k}, op_{i,j}, op_{i,k} \in OP_i$，表示算子 $op_{i,k}$ 的输入是算子 $op_{i,j}$ 的输出。f_i 运行时的状态变化对应其算子中运算参数的不断更新。集群调度在考虑算子构造数据流时需要达到以下两个调度目标：

(1) 数据流资源使用效率(Flow Resource Efficiency, FRE)：FRE 表示对数据流集合 F 中每个数据流 f_i（共计$|F|$个）和供给资源的每个集群节点 $node_m$（共计$|M|$个），任意算子 $op_{i,j}$ 在第 n 次迭代运行（共计 N_i 次）的资源使用量 $R(op_{i,j}, n)$ 之和占集群每个节点的资源供给量 $P(node_m, n)$ 之和的比例，由式(2)表示：

$$FRE = \frac{\sum_{i,j,n} R(op_{i,j}, n)}{\sum_{m,n} P(node_m, n)}$$

$$i, j, m, n \in N^+, i \le |F|, j \le |C(f_i)|, m \le |M|, n \le N_i \tag{2}$$

其中，i, j, m, n 分别代表集群中每个数据流，数据流中每个算子的编号，供给资源的节点编号和算子迭代运行次数的编号。$0 \le FRE \le 1$，FRE 的大小表明算子资源供给是否和需求匹配，集群资源是否得到了有效利用[4]。

(2) 数据流完成时间（Flow Completion Time, FCT）：FCT 表示集合 F 中每个数据流 f_i 的完成时间之和，由式(3)表示：

$$FCT = \sum_{i=0}^{|F|} \sum_{j=0}^{|\max C(f_i)|} RT(op_{i,j}) + WT(op_{i,j}, d_{i,j}) \tag{3}$$

其中，单个数据流完成时间由其最长依赖路径（通过 $\max C(f_i)$ 计算）上算子的运行时间 $RT(op_{i,j})$ 和等待时间 $WT(op_{i,j}, d_{i,j})$ 决定，$d_{i,j}$ 代表算子 $op_{i,j}$ 的所有依赖。FCT 的长短表明数据流中算子的等待时间是否得到了有效限制 **Error! Reference source not found.**，运行时间是否受到干扰。

2.2 挑战1：难以刻画多种算子组合及其资源需求

与传统数据流不同，状态可变数据流主要以算子为粒度迭代运行[1][2]，在每次迭代时都会根据集群状态尝试组合出新算子（图 2 中无填充节点）。这将导致算子及其依赖关系等发生变化[15]，需要重新估算新算子的资源需求。

如图 2 所示，迭代运行中主要有纵向（图 2(a)）、横向（图 2(b)）和混合（图 2(c)）三种组合方法，每种组合前后的处理逻辑是等价的。已有面向传统数据流的调度系统通常假设算子的资源需求是已知的，难以刻画迭代运行时新算子的需求。因此，如何根据多种算子组合方法，动态刻画算子的资源需求 $R(op_{i,j}, n)$ 面临挑战。

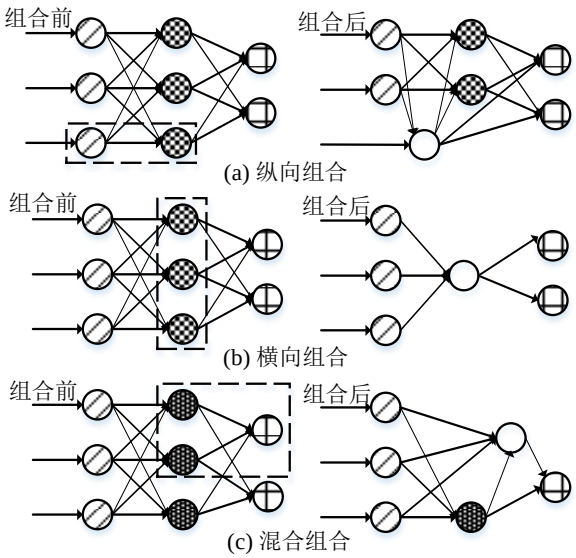


图 2 状态可变数据流存在多种算子组合方法

2.3 挑战2：难以结合算子状态变化进行调度决策

如前文所述，每次迭代运行算子的 GPU 资源需求是动态变化的，调度系统需要综合考虑不同算子在迭代时的 GPU 资源需求，合理供给 GPU 资源，在满足资源需求的同时，提升数据流资源使用效率 FRE（公式(2)）。如图 3 所示的例子，$op_{1,1}$和$op_{2,1}$在不同迭代阶段的资源需求分别是(8,4,4)和(8,4,8)，在第 i、i+1 和 i+2 次迭代时，可分别采用公平供给、优先供给小资源容量节点和优先供给大资源容量节点三种约束[16]，通过第 i+1 和 i+2 次迭代仅分别供给 GPU_1和GPU_2资源，来最小化集群节点资源供给 $P(node_m,n)$（公式(2)）。已有面向传统数据流的调度系统总假设算子之间约束是固定的，因此如何根据算子资源动态变化灵活表示多种约束，并快速做出调度决策面临挑战。

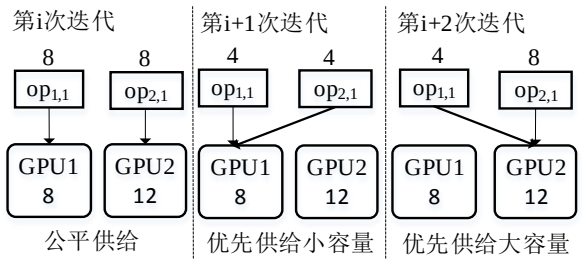


图 3 集群调度需要考虑算子间约束

2.4 挑战3：难以根据算子状态同步方式进行调度调整

算子每次迭代时都需同步状态，如果同步时部分算子运行时间 $RT(op_{i,j})$过长，会增加其他算子的等待时间$WT(op_{i,j},d_{i,j})$，进而导致所在状态可变数据流完成时间过长。

如图 4 所示，当前集群环境下算子状态同步方式主要有以下四种[17]：(1) 主从（图 4(a)）：$op_{1,1}$ 需同步所有的$op_{1,2}$状态后方可进入下一次迭代；(2) P2P（图 4(b)）：所有$op_{1,1}$按照规定顺序循环同步；(3) 管道（图 4(c)）：算子$op_{1,1}$ ~ $op_{1,5}$按照依赖顺序先后同步；(4) 分层（图 4(d)）：不同层中的算子灵活使用多种同步方式。由于存在资源竞争、集群中多种同步方式并存等因素，会导致部分算子（如图 4 中虚线框中的算子）运行、等待时间过长，严重影响数据流完成时间 FCT（公式(3)）。当前工作主要面向单一同步方式降低等待时间，难以适用于其他方式。因此，如何根据不同的算子状态同步方式进行差异化的调度调整面临挑战。

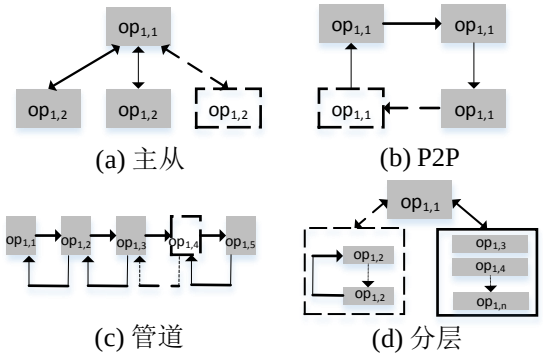


图 4 多种算子状态同步方式

2.5 研究框架

为了应对上述数据流集群调度的挑战，需要首先使用多种感知和协同机制刻画状态可变数据流及其算子的资源需求；然后在决策模型中灵活表示对应的调度约束；最后实施并持续调整调度。综上，形成如图 5 所示的研究框架。

3 算子资源需求刻画

为应对挑战 1，本节首先描述如何通过感知机制估算状态可变数据流中单个算子在纵向、横向和混合组合场景下资源需求；然后阐述如何通过协同机制预测状态可变数据流中多种组合方式下算子资源需求，如图 6 所示。

(1) 算子感知机制：刻画数据集、算子依赖和运算逻辑三个方面和单个算子资源用量之间的关联性，进而感知算子的资源需求；

(2) 算子协同机制：主要考虑训练和推理阶段中算子资源需求的差异性，如训练阶段通常设置算子可正常运行的资源下限 $MinR$，推理阶段则设置可

应对负载变化的资源上限 *MaxR*，不同资源上下限需要多种组合方式下的算子协同调整各自资源需求。

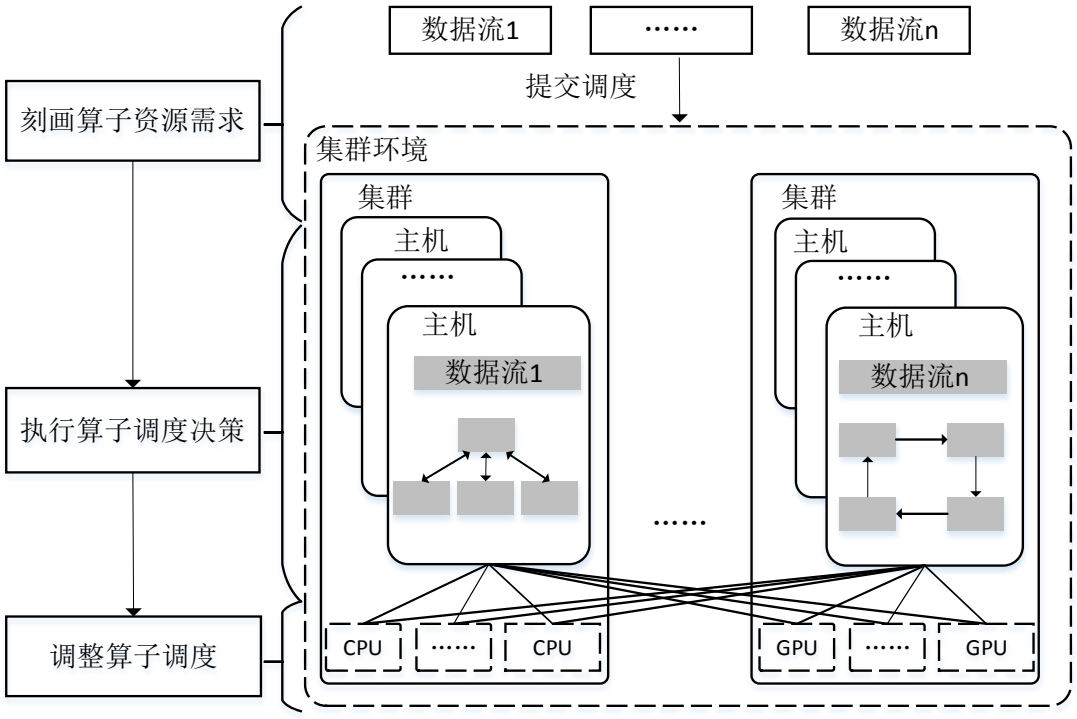


图 5 数据流集群调度的研究框架

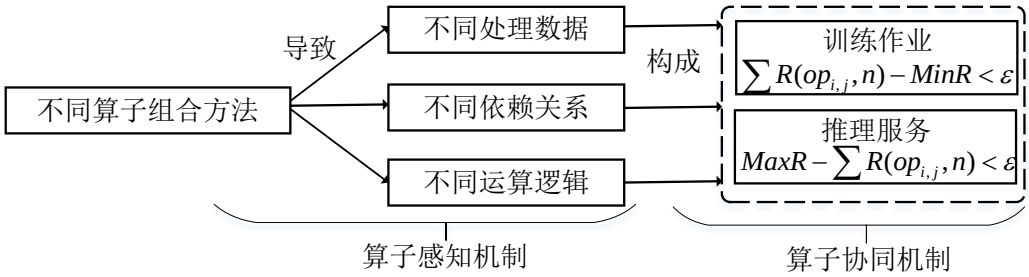


图 6 算子资源需求刻画机制的执行流程

3.1 算子感知机制

3.1.1 感知跨集群数据集

纵向组合（图 2(a)）不改变算子处理的数据输入，可感知输入数据集来确定不同纵向组合下的算子资源需求。算子处理的数据集往往来源于多个集群，除考虑单个集群中数据集的特点，如数据量、数据类型和数据分布等之外，还需要感知多个集群算子协作时带来额外的数据传输、隐私数据加密等 IO 和计算资源开销。

(1) 集群内单数据集：综合使用贝叶斯优化（Cherrypick[18]，AutoDeep[19]）、有监督学习（Ernest[20]，MRTune[21]）等方法，以少量数据作为输入，采样算子运行时间和资源使用关系，进而估算算子资源需求；

(2) 集群间多数据集：构建统一的跨集群资源视图[22]，感知隐私数据加密[23]的资源开销，刻画集群间算子协作通信的资源开销[24][25]，进而估算算子资源需求。

理论上，相较于人工指定算子资源，上述两种感知机制能刻画算子在单个和多个集群环境下的资源需求。但已有工作通常采用统计或学习的方法，并假设资源是同构的，还存在以下局限性：(1) 统计学习方法的实际效果依赖于历史试运行数据的收集，而状态可变数据流往往运行时间较长，收集跨集群运行数据需要耗费大量资源；(2) 机器学习应用中存在多种数据集，当前该机制只能感知几种有限的数据集下的算子资源需求，其应用场景比较受限。

3.1.2 感知算子依赖

横向组合（图 2(b)）会改变算子依赖关系，需通过依赖情况的变化估算算子资源需求，感知不同算子依赖关系对算子本身资源需求的影响：

(1) 顺序依赖：文献[26][27]依次将算子状态的读取、更新等阶段构造为一个完整的顺序依赖，不同阶段内有不同的算子，基于不同依赖阶段刻画算子的差异化资源需求；

(2) 循环依赖：GRNN[28]将数据流中的循环结构（如循环神经网络等）构造为算子的循环依赖，每次循环时算子所依赖数据的加载时机、缓存读取方式等均与顺序依赖有较大差异，已有方法主要采用启发式和强化学习估算算子资源需求：

(1) 启发式规则[29]：根据依赖关系数据的传输量、满足依赖时算子执行顺序，来估算算子资源需求，避免供给资源时的资源浪费；

(2) 强化学习规则[15][30][31]：将节约的完成时间作为收益，采样收益和资源使用间的关系，通过收益最大时的资源使用估算需求。强化学习规则估算时间较长，但可得到比启发式规则更精确的算子资源需求。

随着多种机器学习系统[32][33]的发展和完善，多系统的协作需求逐渐增大，算子依赖感知机制只能对某一特定机器学习系统中的算子固定依赖有效，受限于算子差异性等因素难以快速扩展到其他系统。

3.1.3 感知运算逻辑

多种混合组合方法（图 2(c)）会改变算子的运算逻辑，可根据运算逻辑确定算子混合组合时的资源需求。不同运算逻辑在异构资源（CPU、GPU 等）上的资源用量有差异，估算实际的资源需求要分析不同算子运算逻辑使用异构资源[34]的情况。

(1) 运算逻辑使用 CPU 资源[35][36]：DeepCPU[35]分析矩阵乘法算子使用 CPU 缓存的情况，通过该类算子的计算过程和缓存用量，估算 CPU 资源需求以避免算子间的缓存冲突；BRM[36]分析不同算子使用 CPU 时访问内存的方式，估算 CPU 资源需求以保障内存高效访问；

(2) 运算逻辑使用 GPU 资源[37]：机器学习系统往往通过显存池[1]方式占用过多 GPU 资源，从而难以精确估算单个算子的 GPU 资源需求。对此，需要建模算子运算逻辑对应使用 GPU 资源的过程[37][38]、并行计算方式[39]，从而能判断算子运算逻辑真实的资源开销[40]，避免资源需求估算超过自身实际的情况。

伴随着集群使用越来越多的异构资源[34]，该机制还存在一定局限性：(1) 现有异构资源上运算逻辑的感知机制只能刻画特定算子组合方法在特定资源上的需求，当前工作不能将感知机制泛化到其他资源或算子，其适用范围较窄；(2) 实现感知机制的驱动往往需要对 NVIDIA 等已有驱动进行大量改造，而现有机器学习系统难以支持上述改造，存在兼容性问题。

3.2 算子协同机制

3.2.1 作业内算子协同

为了使作业在运行时占用较少资源，需要预测对状态可变数据流完成时间影响较小的算子组合，并确定这些组合的资源需求，保障较短的数据流完成时间。

(1) 基于资源偏好组合算子：面向异构资源时，Monotasks[41]、Usra[42]等工作首先刻画算子的资源偏好，如 CPU、GPU 和 IO 等，然后减少相同资源偏好的算子组合[43]的非偏好资源需求时，就不会严重影响完成时间；面向同构资源时，根据算子组合的状态变化阶段[44]、速度[45]来预测资源需求，当状态变化速度较慢时，就可减少算子资源，也不会严重影响数据流完成时间[46]。

(2) 基于状态变化情况组合算子：可组合不同异构资源需求的算子[41][42]；也可以组合不同状态变化阶段和速度的算子[44][45]。根据组合情况预测资源需求，能协同作业内多个算子的并行运行，避免资源浪费。

上述算子协同机制在实际使用时仍然存在一定局限性：(1) 当前工作只能快速刻画算子的一种资源偏好，但由于算子能使用多种异构资源，算子存在多种资源偏好的情况[47]，减少算子其他偏好资源会严重影响完成时间；(2) 合并算子资源需求需要其状态变化相对稳定，当算子出现剧烈的状态变化时，合并方式需要频繁更新，这会影响完成时间。

3.2.2 服务内算子协同

为使服务在应对特定负载时不额外消耗过多资源，需要预测对状态可变数据流完成时间影响较大的算子组合，并确定这些组合的资源需求。当前工作主要面向周期和突发两类负载：

(1) 周期负载：Aliyun[48]、Google[49]、Microsoft[50]对负载进行分析，得到每个算子在不同周期内的资源需求和运行时间，通过人工预设或机器学习预测等方式来估算算子组合及其资源需求，从而使算子的资源需求能够准确反映出负载的变化。如在负载高峰到来前，根据分析结果，增加对应服务中部分算子的资源，避免资源浪费；

(2) 突发负载：当前工作依次设置数据流中每

个算子的资源需求[15][51]以满足服务突发负载的规模及其响应时间要求；同时，也需要构造资源需求和运行时间之间的关联模型[52]，保证资源需求预测的准确性。

该机制面向周期、突发负载时，其预测效果的好坏还受到以下两方面因素的影响：(1) 伴随机器学习应用的不断发展，周期负载的规律性也在不断变化[53]，如算子的并行程度、算子在新硬件上的资源需求等，该机制需要不断更新预设规则或预测模型才能准确把握应用负载变化对资源需求的影响；(2) 只有在突发负载到来后，才能开始依次预测不同组合方法下的算子资源需求，预测过程自身也需要消耗一部分资源，且容易和算子资源需求冲突，当前工作还缺乏对该机制的资源开销评估。

3.3 对比与小结

算子资源需求刻画机制的对比如表 1 所示：基于算子状态感知机制的资源刻画方法存在“采样频率”和“资源需求准确度”之间的矛盾，要得到较高的资源需求准确度，就需要对算子资源使用和运行时间之间的关系进行频繁采样，其代价较大。刻画算子资源需求要根据实际应用场景选择对应感知机制；算子协同机制使用人工分析的启发式规则方法来刻画算子组合对不同资源变化的敏感性，使作业使用少量资源维持自身正常运行，使服务使用有限资源来应对较高负载。当前主要根据特定场景下算子组合的完成时间、状态变化速度等来预测资源需求，未来还需要结合多种机器学习系统，继续扩展预测方法的适用范围，增强对不同算子组合的感知能力，使资源需求进一步逼近服务、作业有限资源的上下限。

表 1　算子感知与协同机制

名称	刻画对象	传统数据流	状态可变数据流	刻画方法	资源需求描述	资源需求类型
AutoDeep[19]	集群内在线数据集	×	√	贝叶斯优化强化学习	数据流	CPU、GPU
MRTune[21]	集群内离线数据集	√	√	启发式搜索	特定算子	CPU、内存
Batchcrypt[23]	集群间隐私数据集	×	√	启发式规则	数据流	CPU、GPU、IO
Turbo[25]	集群间离线数据集	√	√	机器学习预测	数据流	IO
Optimus[26]	算子间顺序依赖	×	√	多项式拟合	特定算子	GPU、IO
GRNN[28]	算子间循环依赖	×	√	启发式规则	特定算子	GPU、IO
Spotlight[30]	算子间泛化依赖	×	√	强化学习	算子	CPU、GPU、IO
DeepCPU[35]	矩阵乘法运算逻辑 CPU 缓存结构	×	√	容量匹配	矩阵乘法算子	CPU
Xsp[37]	运算逻辑调度层次、GPU 结构	×	√	事件驱动监测	算子	GPU
Usra[42]	作业中算子组合的资源偏好	√	√	启发式规则	特定算子	CPU、GPU、IO
HyperSched[45]	作业中算子组合的状态变化速度	×	√	启发式规则	特定算子	GPU
Astra[47]	作业中算子组合的完成时间	×	√	启发式规则	算子	GPU
文献[52]	服务中算子的调度层次、资源用量和运行时间	×	√	启发式规则	算子	CPU、GPU、IO
InferLine[51]	服务完成时间、算子资源用量和运行时间	×	√	启发式规则	算子	GPU、IO

×代表不支持对应的数据流，√支持对应的数据流

4 算子调度决策

为应对挑战 2，算子调度决策需要最小化资源供给 $P(node_m,n)$ 提升资源使用效率，同时又不影响数据流完成时间。如图 7 所示，本节首先描述哪些约束能反应出算子的状态变化，然后确定不同决策模型面向这些约束的表示能力和局限性，以及哪些技术手段能加速调度决策求解的过程。

(1) 决策约束：当前算子调度决策包含多种约束，如图 7 的结果 1 将需要频繁通信的算子都放在主机 a 上（亲和性），能减少算子的 IO 资源供给，但会使主机 b 的数据流资源使用效率降低；对应地，结果 2 能将算子均匀放置在两个主机上（公平性），同时使用两个主机可为算子供给更多的资源，能提升数据流资源使用效率，但算子跨主机通信 IO 会

增加数据流的完成时间；

(2) 决策模型：当前工作主要基于图模型、队列模型来表示需求和供给间的多种约束，并选择求解视角输出调度决策结果。算子状态的不断变化，要求决策模型能够灵活表示、选择多种约束和视角；

(3) 决策求解：针对多种约束要快速做出调度决策。为了降低求解的复杂度，当前工作通过资源供给和需求视角缩小调度决策求解的搜索空间。

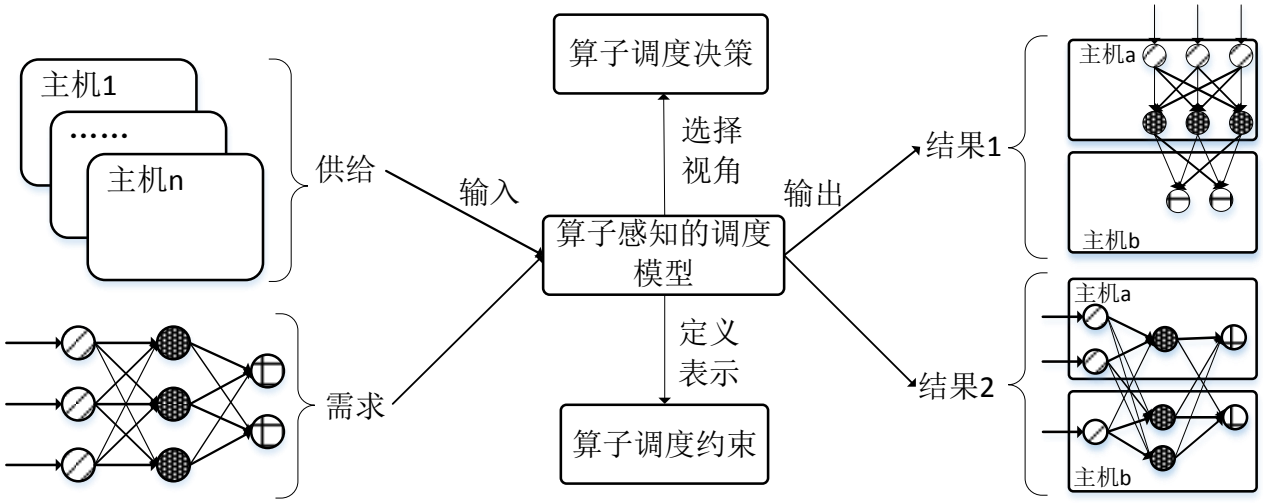


图 7 算子调度决策流程

4.1 决策约束

“约束”代表算子之间、算子和资源之间的关联关系。约束的定义如下所示：

(1) 算子间公平性表示一定时空范围内并行运行的算子都能均等使用资源，避免有些算子得不到资源；

(2) 算子间优先级表示算子资源使用有先后、多少等差别，总能保证部分算子优先运行；

(3) 算子之间亲和性表示某些算子调度时应该放到一起共享资源，以降低算子间通信开销并提升资源使用效率；算子和资源之间的亲和性表示算子应该放到特定资源上，以缩短算子运行时间。

4.1.1 算子公平性

最大最小公平算法[54]依据多个算子的资源需求平分已有资源，往往只能使算子获取保障其运行的最小资源，而不是最短完成时间对应的最优资源。算子公平性难以使用传统算法来表示：(1) 算子资源需求中存在状态依赖关系，依赖关系不满足时，给算子供给资源时算子也无法运行，会导致资源浪费；(2) 算子资源需求中包含大量异构资源需求，传统算法只能处理同构资源容量的公平分配；(3) 算子使用的异构资源之间存在关联性，对其中一种资源进行传统公平性分配时，会导致其他资源的供给出现竞争或浪费。为了解决上述问题，当前工作结合算子状态变化定义了以下几个公平性：

(1) 增量式同构资源公平性[54]：该公平性是最大最小公平性在支持算子依赖时的扩展，以算子依赖关系为基础，将状态可变数据流中的算子状态计算划分为不同过程，以增量方式分别计算每个过程中的算子公平性，这样就能在依赖关系满足时公平分配算子资源。该公平性总能让数据流把自身暂时不需要的资源提供给其它算子；

(2) 基于主资源的异构资源公平性[55]：该公平性是最大最小公平性在考虑异构资源时的扩展，针对算子的多种异构资源需求，将需求量最大的资源作为主资源，并公平供给算子主资源。此时，调度系统总能保障一个算子的主资源供给，而将次要资源供给其它算子。该公平性在算子的主资源差异性较大时效果较好，但当前机器学习系统都倾向给算子供给 GPU 这一种资源，主资源差异性不明显，此时该算法容易退化为传统最大最小公平性；

(3) 基于资源联系的异构资源公平性[56][57]：该公平性是主资源公平性在考虑资源联系时的扩展，HUG[56]根据算子 IO 资源间的联系，公平分配多种异构 IO 资源；NC-DRF[57]更进一步，监测算子的 IO 资源使用情况，推测出资源联系，降低该公平性的使用门槛。上述工作只能表示 IO 资源之间的联系，未来还需要考虑支持其他计算资源间的联系；

(4) 基于性能交易的异构资源公平性[58][59]：该公平性是主资源公平性算法考虑算子状态变化时的扩展，根据相同状态变化时的算子完成时间和资源容量，文献[58][59]定义“性能交易比例”（如 1

GPU=4 CPU），实现异构资源的公平性分配，避免异构资源浪费；

(5) 基于长期完成时间的同构资源公平性[60]：该公平性不能保证算子在任意时刻都能公平使用同构资源，但能保证数据流运行完成后算子总有相同的总资源使用时间[60]。短期来看，状态变化较好的算子能使用较多资源，长期来看，数据流中算子总能均等使用已分配资源。

4.1.2 算子优先级

已有方法主要采用最短完成时间优先、先来先服务、最少已运行时间优先，往往不能有效缩短数据流完成时间，存在较大的局限性：(1) 不同数据流中的算子运行时间差异大，基于短完成时间设置优先级，在资源竞争时往往部分算子长期得不到资源；(2) 算子运行时间长并不能代表其状态变化符合预期（如机器学习算法不收敛，会导致算子长期占用大量资源），只根据已运行时间来设置优先级会使资源不能优先供给状态变化较好的算子；(3) 大量算子并行运行时，先到来优先不感知算子状态变化，容易退化为最大最小公平资源分配，这往往使后到来算子得不到资源。为解决上述问题，算子优先级从以下三方面扩展传统优先级的内涵：

(1) 基于 Gittins 指数的优先级[61]：该优先级是对最短完成时间优先的扩展，Gittins 指数可根据算子完成时间分布和已运行时间不断设置算子优先级，相较于传统静态优先级的设置方法，该优先级能反映出不同算子的状态变化趋势，避免传统优先级导致部分算子长期无法运行的问题，从而缩短数据流完成时间。Gittins 指数需要预定义算子完成时间分布，使用前还需要大量的统计分析工作；

(2) 基于加权最少已运行时间（LAS）的优先级[62]：该优先级是对最少已运行时间优先的扩展，在考虑算子已运行时间[62]的同时，还更新权值以反映算子当前的状态变化情况。如文献[63]对算子的状态变化好坏进行分级，对不同分级的算子分别设置权重；文献[64]将当前状态变化较好的算子赋予更高权重。该优先级使资源更多地供给状态变化较好的算子，可缩短数据流完成时间，但还需要不断尝试多种权重设置方法，才能达到最短数据流完成时间；

(3) 依赖限制的先到来优先级[65]：该优先级是对先来先服务的扩展，根据多个数据流的资源需求和算子的依赖关系，确定影响数据流完成时间的关键依赖，优先调度关键依赖路径上资源需求较大的算子。该优先级在一定程度上避免先到来优先的退化问题，但其主要针对传统数据流，控制粒度较粗，改造后才能应用于状态可变数据流。

4.1.3 算子亲和性

传统亲和性在调度前已知且固定不变，而算子状态的频繁变化会导致亲和性难以在调度前确定：(1) 算子间亲和性难以确定，状态可变数据流中的并行运行算子数量远大于传统数据流，算子一起放置能缩短通信开销提升亲和性，但算子并行数量过多时又会出现资源竞争导致亲和性降低；(2) 算子与资源间的亲和性难以确定，不同于传统数据流往往使用同构资源，异构资源的加速效果、连接方式等都会影响状态可变数据流的算子运行，它们之间的亲和性变得更加复杂。算子亲和性从以下三方面扩展传统亲和性的内涵：

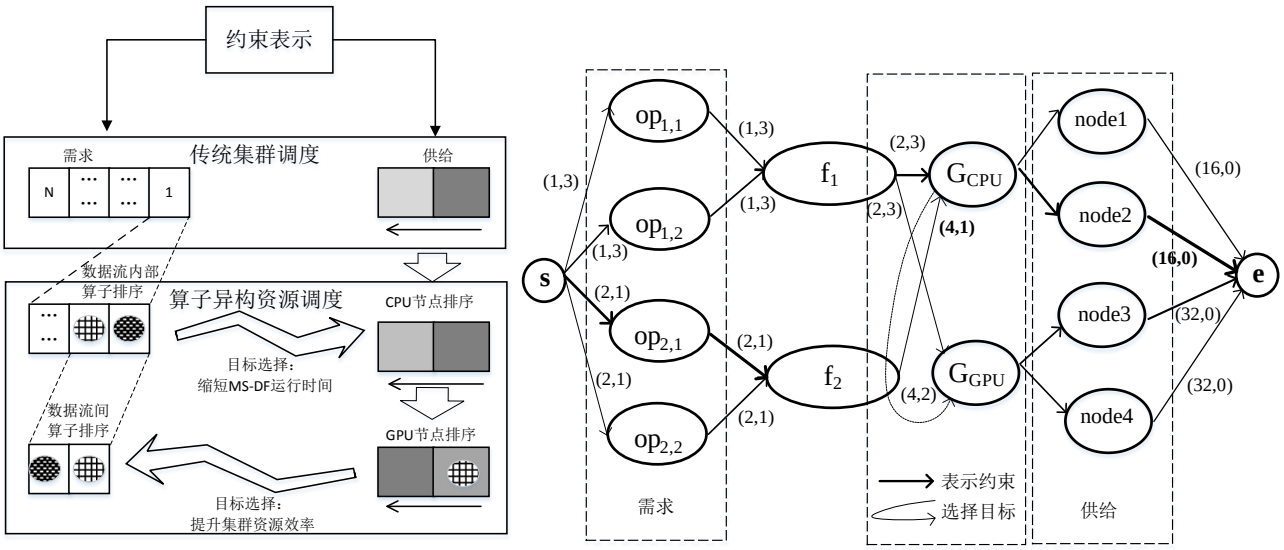
(1) 基于依赖分析的算子间亲和性[66]：为确定一个数据流中算子间亲和性，可分析算子间的依赖，根据依赖数据量定义算子间亲和性，算子间亲和性越高代表其依赖数据量越大，放在一起能降低更多的通信开销[66]。当前工作只考虑了算子的顺序依赖，未来还需要支持循环依赖等情况；

(2) 基于干扰检测的算子间亲和性[67]：为了确定多个数据流中的算子间亲和性，当前工作从 IO 资源[67]、计算资源[68][69]等方面检测算子运行时的资源干扰，没有干扰或干扰较低的算子间存在较高亲和性。基于人工分析的检测[67]只能发现特定的资源干扰，基于强化学习的检测[69]需要数据流大量反复运行，通过干扰检测定义亲和性还需要综合考虑其适用场景和实施代价；

(3) 基于完成时间分析的算子与资源间亲和性[60][70]：当前工作构造数据流完成时间和资源连接方式间的关联关系，分别确定集群节点内[59]、机架内[60]、机架间[70]、不同集群区域间[70]资源连接对算子的实际加速效果，实际加速效果越好，算子和对应资源间的亲和性就越高。当前工作目前只考虑了 GPU 资源的连接情况，还需要扩展对其他资源联接情况的支持。

4.2 决策模型

如图 8 所示，决策模型通过队列、流图两种数据结构来表示多种约束，满足约束后，如图 8(a)中的队列模型，其使用线性数据结构，可通过多队列间的多次重排序进行决策；图 8(b)中的流图模型使用图数据结构，通过寻找增广路径（图中加粗实线）的方式进行调度决策。



(a)队列模型 (b)流图模型

图 8 队列模型与流图模型

4.2.1 队列模型

如图 8(a)所示，算子感知的集群调度系统在传统调度基础上，需要维持算子资源需求和异构资源供给的多个队列[71]，根据算子的状态变化进行需求和供给队列的多次排序以满足约束，并能最大化调度目标。其关键在于如何灵活表示不同约束，并在算子状态变化引起的约束变化时快速求解得到相应调度结果。队列模型的要素如下所示：

(1) 数据结构：算子资源需求和可用资源的多个队列可通过到来时机、资源量大小、运行时间长短来进行排序[72]。不同于传统集群调度只需考虑数据流所在作业或服务的运行时间、资源容量等来构造队列，构造算子队列还需要考虑其依赖关系、异构资源加速效果等；

(2) 表示约束：通过多个队列的重排序来表示算子多种约束。如图 8(a)中首先对算子队列进行排序满足算优先级等约束，然后重排序节点满足亲和性等约束。不同于传统集群调度中约束固定不变，算子感知的队列模型还需要有约束更新并重新决策的能力[16]；

(3) 调度决策：根据(1)中的顺序和(2)中的约束，模型在多次比较算子和资源的绑定关系过程中，往往通过一些启发规则来减少最优调度决策搜索空间。传统集群调度只使用有限的启发规则，而算子状态的频繁变化会使一些规则失效。如整数最优化规则[16]的决策依赖于历史调度结果，状态变化会使历史调度变差；装箱规则[73]只选择能提升资源效率的绑定关系，而忽略数据流完成时间。

4.2.2 图模型

如图 8(b)所示，流图模型中的点分别代表待调度的算子$op_{i,j}$、数据流f_i、机架G_{cpu}和G_{gpu}、集群节点$node_m$，点之间的边权值代表算子资源需求量或每个节点可用资源，边费用代表调度该算子的代价，如边$s \rightarrow op_{1,1}$上的(1,3)代表算子$op_{1,1}$的资源需求是 1，调度代价是 3，边$node_1 \rightarrow e$上的(16,0)代表物理机的资源容量是 16。图中从 s 到 e 的一条连通边集合代表一个算子和资源的绑定关系（加粗实线），如果该关系不满足决策约束，可以重新调度算子（虚线）。流图模型的要素如下所示：

(1) 数据结构：如果 s、e 节点各只有一个，图模型即是最大流结构[74]，如果有多个则是二分流结构[75]。传统调度基于这两种结构，只能表示数据流的整体资源需求和节点的同构资源容量[75]，难以表示算子之间的依赖关系和异构资源[74]等；

(2) 表示约束：图模型通过设置边费用来表示约束，如 Firmament[76]使边费用和数据本地性关联，可表达亲和性；Aladdin[77][78]使边费用与调度顺序关联，可表达优先级。这些基于边费用的方法，往往只能表达一种约束，算子往往需要同时表达多种约束，这需要扩展图模型的边费用维度；

(3) 调度决策：流图模型通过不断寻找增广路径（一条从 s 到 e 的连续边，能增大流图的流量或减少费用）来得到最优调度决策[76]，增广路径要能满足算子约束。由于单个算子的资源需求只能由同一节点资源供给，会导致边上的流量不可分，进而影响现有最小费用或最大流算法的鲁棒性；同时，

增广路径的寻找过程会触发重调度，传统集群重调度的频率不高，但面向算子复杂约束时会经常出发重调度并影响算子的稳定运行。

4.3 决策求解

决策求解要在尽量短的时间内，计算得出满足约束且能达到预期目标的算子与资源间绑定关系。为应对算子状态频繁变化导致的决策求解复杂度高的问题，当前工作主要通过以下三个视角来排除达不到最优调度目标的绑定关系，从而减少最优调度决策的搜索空间。

4.3.1 资源供给视角

已有工作对现有集群中的算子资源供给情况进行分析，发现供给存在一定规律，决策时就可忽略不满足这些规律的算子资源绑定关系，从而降低搜索空间。

(1) 基于空闲资源减少决策搜索空间[79][80]：根据大量算子在不同时间的资源使用量和节点可用资源总量，文献[79]计算出任意时间的空闲资源容量，文献[80]能确定这些空闲容量应该供给哪些算子，并排除不包含上述算子的绑定关系。当前工作的空闲资源分析只能对单一资源生效，还需进一步联合分析多种异构空闲资源情况；

(2) 基于已用资源减少决策搜索空间[81][82]：文献[81]发现集群节点中所有算子已用资源时间呈现出一定的概率分布，文献[82]发现集群节点中所有算子的资源使用总量也符合一定分布。据此，就可以排除不符合上述分布的资源算子绑定关系。但已有工作只在 Google 集群中得到了有效性验证，由于集群运维技术水平等因素，在其他集群中验证算子资源相关的概率分布还存在较大难度。

4.3.2 资源需求视角

一个节点上大量算子之间存在复杂的约束关系，可使用多种启发式规则降低这些约束的验证复杂度，减少最优调度的搜索空间。

(1) 基于领域特定语言减少决策搜索空间[16][83]：文献[83]提出了面向资源需求和多种约束的描述语言；文献[16]可在描述语言的基础上，通过线性规划方法判断当前绑定关系的约束违约情况，及时排除不符合约束的调度决策。这种基于领域特定语言的描述和判断方法能降低资源需求和约束的表示难度，然而当前主流机器学习系统都没有支持这种领域语言，使用时还需要考虑 API 兼容性问题；

(2) 基于需求与状态变化关系减少决策搜索空间[26][84]：文献[26][84]构造了资源需求和状态变化的关联关系，调度决策可排除不能最大化数据流状态变化速度的算子资源绑定关系。该关联关系的有效性取决于算子状态变化是否平稳，如果状态变化剧烈，就难以判断决策是否能稳定保持调度目标。

4.3.3 同时考虑需求与供给

可通过调度系统的分层、分区架构来分别并行调度不同的算子和资源，减少决策时的搜索空间。

(1) 基于分层调度减少决策搜索空间[85][86]：当前工作一般使用两层调度架构，上层可调度数据流，下层调度数据流中的算子，通过控制调度模型中的具体参数[85]，如队列长度、每个队列中的资源类型[86]等，来区分不同算子和资源。当前工作只支持两层调度的自上而下协同，不支持同层不同节点调度器间的协同，调度架构难以对算子状态变化做出及时跨节点协同；

(2) 基于分区调度减少决策搜索空间[72][87]：Pigeon[72]以数据流需要的完成时间为分区标准，完成时间相似的数据流可以划分到相同分区；Kubernetes[87]可为集群节点打上不同标签，同构资源节点会一般会得到相同标签并组成一个分区。每个分区的决策搜索空间都比较小，但当前工作的分区划分标准比较简单，难以根据算子状态进行分区协同，分区调度器也难以与机器学习系统中的调度器进行有效协作。

4.4 对比与小结

算子调度决策的约束、模型和求解视角如表 2 所示：结合算子状态情况，当前工作从算子使用的异构资源、运行时间等分别定义公平性、优先级和亲和性，但部分约束的定义如 HUG[56]、GRAPHENE[65]等还没有在状态可变数据流中进行充分验证，这需要进一步结合算子迭代运行时的状态变化扩展约束内涵；队列模型能支持所有约束，但其代价是需要为每一种约束额外开发对应的支持能力；流图模型通过自身的数据结构设置就能支持部分优先级、公平性，但难以支持异构资源公平性、算子间依赖和亲和性等，还需要扩展最小费用最大流算法；通过资源需求和供给两个视角，能快速得到最优的调度决策，如 DRF[55]基于资源需求视角区分算子不同主资源，然后基于供给视角选择对应主资源节点，直接进行这些算子和节点的匹配，就能减少调度决策搜索空间，当前这些求解优化技术一般只对单个约束有效，未来还需要增加多种视角的求解优化能力适用于多个约束。

表 2 算子调度约束

名称	约束类型	传统数据流	状态可变数据流	图模型	队列模型	预期目标	算子支持	主要决策求解视角
文献[54]	增量式的同构资源公平性	√	√	√	√	FRE	不同算子组合间的依赖	基于时空空闲资源减少决策搜索空间
DRF[55]	基于主资源的异构资源公平性	√	√	×	√	FRE	表示算子使用最多的资源	基于分层调度减少决策搜索空间
HUG[56]	基于资源联系的异构资源公平性	√	×	×	√	FRE+FCT	表示算子不同资源间的关联性	基于已用资源减少决策搜索空间
Gavel[59]	基于性能交易的异构资源公平性	×	√	√	√	FRE	表示算子组合在异构资源上的运行时间	基于已用资源减少决策搜索空间
Themis[58]	基于长期完成时间的同构资源公平性	×	√	√	√	FRE+FCT	表示算子状态变化的差异性	基于需求与状态变化关系减少决策搜索空间
Tiresias[61]	基于 Gittins 指数的优先级	×	√	√	√	FCT	表示算子运行时间分布	基于已用资源减少决策搜索空间
E-LAS[63]	基于加权最小已运行时间的优先级	×	√	√	√	FCT	表示算子状态距离计算完毕的次数	基于领域特定语言减少决策搜索空间
GRAPHENE[65]	依赖限制的先到来优先级	√	×	×	√	FRE+FCT	表示算子组合间的关键依赖路径	基于时空空闲资源减少决策搜索空间
文献[66]	基于依赖分析的算子间亲和性	√	√	×	√	FCT	表示算子间的通信开销	基于已用资源减少决策搜索空间
Salus[67]	基于干扰检测的算子间亲和性	×	√	×	√	FCT	表示算子间的显存周期使用互补关系	基于已用资源减少决策搜索空间 基于时空空闲资源减少决策搜索空间
Hived[70]	基于完成时间分析的算子与资源间亲和性	√	√	√	√	FCT	表示算子间在不同资源拓扑下的运行时间	基于需求与状态变化关系减少决策搜索空间

×代表不支持对应的数据流或模型，√支持对应的数据流和模型

5 算子调度调整

为应对挑战 3，调度调整策略要能缩短算子等待时间 $WT(op_{i,j})$ 和运行时间 $RT(op_{i,j})$ 以长期保障调度目标，如图 9 所示，集群调度系统可面向不同状态同步方式，采取有针对性的调度调整策略：

(1) 针对不同状态同步方式的调度调整策略：如图 9 所示，t_0 时刻两个数据流同时到来，主从方式同步状态的等待时间较长，可以将算子迁移到其他算子的同步等待的间段，避免等待时的资源浪费，使状态同步和状态计算过程的算子充分交叠，缓解同时使用资源的竞争(图 9(a))；P2P 方式计算状态耗时较长，可适当伸缩算子资源，缩短算子运行时间，进而缩短数据流完成时间（图 9(b)）；管道同步时不同算子使用资源不一样，可选择资源消耗少的算子进行挂起和恢复（图 9(c)），缩短算子等待时间；

(2) 针对多种同步方式的混合策略：集群调度系统也可对集群中同时存在的多种同步方式使用混合策略，充分挖掘调度调整的潜力。当前工作基于启发式规则来选择实施特定混合策略。

5.1 算子迁移

主从同步方式下（图 9(a)），为了充分利用同步状态时的计算资源，可以进行算子迁移：(1) 将算子从资源竞争较大的节点迁移到资源容量足够的节点上，以缩短数据流完成时间；(2) 将算子迁移到节点的不同运行时间段，以提升数据流使用资源的效率。其关键在于结合状态变化特征选择迁移哪

些算子，及限制迁移时的资源开销。当前工作通过迁移算子选择和迁移成本控制，来实施调度策略：

(1) 迁移算子选择[67][88][89]：在分析算子长期状态变化及对应资源需求供给情况的基础上，根据算子资源使用周期波动[67]，选择周期互补的算子作为迁移对象，提升资源使用效率；根据异构资源对不同算子的实际加速效果[88][89]，迁移部分算子以最大化加速效果，缩短算子运行时间；

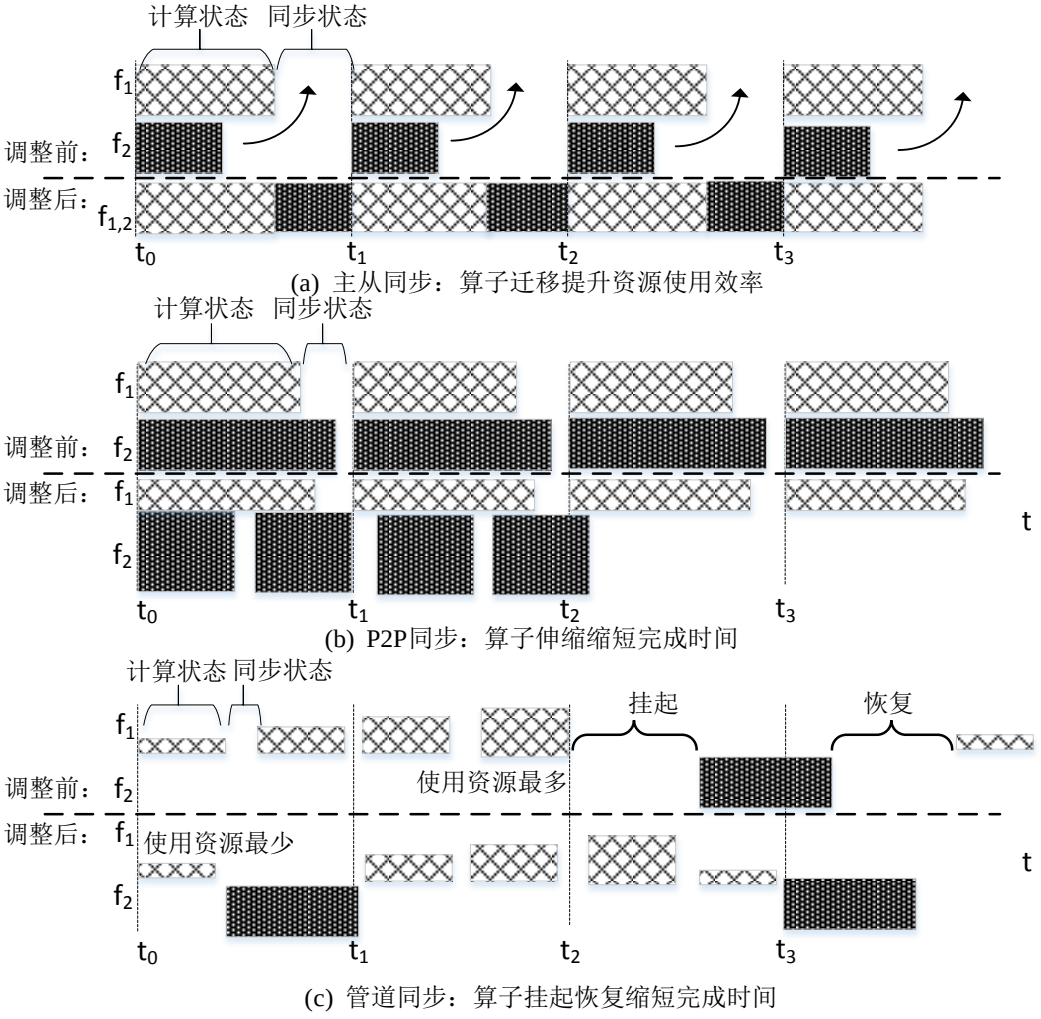


图 9 不同状态同步方式下的算子调度调整策略

(2) 迁移成本控制[90]：选择算子进行迁移时，需要充分减少迁移的资源开销，文献[90]通过状态分割传输和复用算子的内存副本等策略减少算子迁移时的资源开销。

当前迁移策略只能基于算子的历史状态变化，选择需要迁移的算子，难以通过状态变化预测来提前迁移算子。未来调度系统还需要进一步结合智能预测方法，提升迁移策略的实施效果。

5.2 算子资源伸缩

P2P 同步方式下（图 9(b)），伸缩算子的资源可缩短数据流完成时间，应对算子弹性伸缩需要解决两方面的问题：(1) 当前集群调度系统如 Kubernetes[87]只对 CPU 资源提供了有限的伸缩策略，而机器学习系统为了加速算子运行还需要伸缩 GPU 资源；(2) 多个算子往往共享 GPU 资源来并行运行，除调整单个算子资源外，伸缩策略还需要协同调整多个同时运行的算子资源。

(1) 单个算子的 GPU 资源伸缩[91][92]：GSLICE[91]在 CUDA 驱动和用户态网络协议的基础上使算子能够按照不同服务质量要求进行伸缩；Capuchin[92]观察到机器学习系统总是给算子分配过多资源，使用伸缩容量调整方法避免算子对资源的无效占用；

(2) 多个算子的 GPU 资源伸缩[93][94]:文献[93]通过热点算子缓存来加快算子间伸缩速度，文献[94]根据算子的处理数据量、数据分布和运算逻辑同时配置多个算子的 GPU 的线程数。除控制伸缩时的缓存和线程并发数量外，GPU 的显存、张量核等都

有可能是影响伸缩性能的瓶颈，算子资源伸缩还需同时考虑上述多种因素，并根据实际效果为算子提供对应的资源伸缩策略。

5.3 挂起恢复策略

管道同步方式下（图 9(c)），可以挂起当前不运行的算子，执行时再恢复算子。通过固定、动态时间片能实现算子的挂起恢复。

5.3.1 固定时间片

通过控制使用资源的时间片数量，算子能分时复用多种异构资源，当前集群调度系统中 GPU 的计算资源复用都通过固定时间片复用来实现，但存在以下问题：(1) 为了最大化状态变化速度以缩短数据流完成时间，算子在不同状态时有差异化的资源供给，对应不同的时间片数量，传统调度系统往往给算子分配相同的时间片；(2) 传统调度系统只能基于操作系统内核进行整个数据流的挂起恢复，会导致分时复用的集群资源浪费。为解决上述两个问题，当前工作从算子时间片分配和时间片切换两方面，来实施调度调整策略：

(1) 算子时间片分配[95][96]：每个迭代周期内调度系统可根据算子状态变化情况分配不同的时间片，文献[95]基于令牌分发方式确认算子时间片数量，文献[96]基于资源比例设置，确认算子时间片占总时间的比例。这两种方式只能控制算子在不同状态下的计算资源，不能控制存储资源；此外，当前工作还与特定的异构资源驱动耦合[97]，其在除 GPU 以外资源上的挂起恢复效果还需要持续验证；

(2) 算子时间片切换[4][98][99]：文献[4]基于数据流资源使用的周期波动确定不同算子的时间片长短，并在资源使用的波谷执行挂起，可减少由挂起导致的算子等待时间；文献[98]使用算子粒度的挂起恢复来代替对整个数据流的挂起恢复，减少挂起恢复的资源开销。当前工作都能致力于解决 GPU 资源下的复用开销，但机器学习系统的研究表明[99]，CPU、内存等也会影响时间片切换，调度调整需要同时考虑多种资源。

5.3.2 动态时间片

集群调度系统使用固定时间片复用虽然能简化调度调整的复杂度，但这种方式供给的算子资源时间一般不能与其运行时间完全吻合，从而导致资源闲置，降低资源使用效率，也会增加部分算子的等待时间。针对上述两个问题进行调度调整，当前工作通过主动和被动两种时间驱动方式来设置时间片长度：

(1) 主动事件驱动[100][101]：FELIPE[100]面向 GPU 资源，算子在用完 GPU 后向调度系统主动发送事件，调度系统就直接进行时间片切换，从而避免算子资源的无效占用。但 FELIPE 只在虚拟机环境中验证了有效性，而算子还有可能在容器环境中运行。Gloop[101]提出一套事件驱动编程接口，基于自定义 CUDA 驱动能灵活管理 GPU 资源的共享、独占和隔离等需求，但该接口还未与主流机器学习系统集成；

(2) 被动事件驱动[102][103]：当前工作都针对集群的瞬时资源（这种资源稳定供给一小段时间后就会被回收）回收事件来设置时间片长度，如 Spotnik[102]在瞬时资源回收事件触发后，调整 P2P 状态同步方式避免该事件对数据流完成时间的影响；文献[103]基于回归模型预测被动事件的触发时机，并在事件触发前为算子分配新的时间片。被动事件驱动方式当前只能对 Google、AWS 等商业服务的公有云集群有效，其适用场景有限。

5.4 混合策略

混合策略包括以下两方面的组合方式：

(1) 预定义策略组合[4]：经过对算子状态变化的人工分析，Gandiva[4]综合使用各类调度策略，面向主从状态同步，使用固定时间片复用避免算子独占 GPU，使用迁移避免作业间干扰，使用资源伸缩能提升资源使用效率。这种方式需要大量的人力投入和长期机器学习和集群调度的技术积累；

(2) 结合状态变化趋势的策略组合[104]：Prague[104]、RNA[105]面向主从和 P2P 状态同步方式，综合使用挂起恢复和算子迁移策略，避免瓶颈算子对数据流完成时间的影响；HetPipe[17]根据异构资源类型分组节点，在不同组内实施差异化的调度调整策略和状态同步方式；Cerebro[106]随机选择有限的几个调度策略以降低算子的等待时间。集群中会存在多种同步方式，未来还需要扩展策略组合对不同同步方式的自适应能力。

5.5 对比与小结

迁移、伸缩、挂起恢复策略及这些策略的混合选择方法的对比如表 3 所示：当前工作大都只验证了一种策略在某一个或两个状态同步方式下的有效性，甚至部分策略未考虑算子的状态同步方式，缺乏对不同同步方式的调整效果验证，未来需要根据算子在不同状态同步下资源使用情况，能自适应地选择调度策略；混合策略只能以整体数据流作为

实施对象，只能间接控制算子，对单个算子的调度调整能力较弱，未来需要根据数据流中的多种算子组合方法，提升调度调整能力。

表 3 算子调度调整策略

名称	保障策略	传统数据流	状态可变数据流	主要保障目标	策略实施对象	支持的状态同步方式
文献[67]	迁移算子到资源用量互补的时刻	×	√	FRE+FCT	算子	集中
文献[89]	迁移算子到加速比高的闲置资源	×	√	FRE+FCT	算子	集中
DeepClone[90]	减少迁移时的算子数量	√	√	FCT	算子	集中
GSLICE[91]	基于用户态协议栈快速伸缩 GPU	×	√	FRE	算子	管道
Capuchin[92]	减少伸缩时的显存浪费	×	√	FRE	算子	未考虑状态同步
文献[94]	协同考虑多个算子的线程伸缩	×	√	FCT	算子	未考虑状态同步
KubeShare[95]	基于令牌有效期的分时共享	√	√	FRE	数据流	集中
PipeSwitch[98]	基于算子间依赖分时共享	×	√	FCT	算子	管道
FELIPE[100]	基于 GPU 使用事件确定时间片长度	√	√	FRE	数据流	未考虑状态同步
Gloop[101]	基于事件编程定义 GPU 分时复用	√	√	FRE	数据流	未考虑状态同步
Spotnik[102]	基于瞬时资源可用性调整算子运行	×	√	FCT	数据流	P2P
Gandiva[4]	固定时间片+算子迁移	×	√	FRE+FCT	数据流	集中
Prague[104]	同步速度+避免资源冲突	×	√	FRE+FCT	数据流	P2P，分层
HetPipe[17]	根据资源分子实施不同时空策略	×	√	FRE+FCT	数据流	集中，P2P
Cerebro[106]	根据算子状态实时随机时空策略	×	√	FRE+FCT	数据流	集中

×代表不支持对应的数据流，√支持对应的数据流

6 总结及展望

6.1 总结

本文围绕状态可变数据流的基本单元：算子，从资源需求刻画、调度决策和调度调整三方面对状态可变数据流集群调度进行了综述：

(1) 算子资源需求：随着边缘计算、云端组合等新兴场景的不断出现，函数、无服务器计算的不断成熟，算子需要使用更多的异构资源，当前集群调度工作针对 MS-DF 中的算子，根据其运行逻辑、处理数据、依赖关系和其所处作业、服务，使用一系列感知、协同机制，重点刻画算子的 GPU 等异构资源需求；

(2) 算子调度决策：随着强化学习、模仿学习等新理论的出现和不断应用，数据流支撑的深度/机器学习应用规模、结构等会发生频繁变化，以 TVM[107]、MNN[108]为代表的优化工具也会改变算子的资源使用，当前集群调度工作据此设计了调度决策的多种模型、约束和求解算法，满足不同数据流的差异化调度需求；

(3) 算子调度调整：各类机器学习集成系统[32]基于统一资源视图和 API，旨在屏蔽算子迭代运行和状态同步方式等运行细节，当前集群调度将迁移、伸缩、挂起恢复等策略应用于主从、P2P 等状态同步方式，结合算子的状态变化情况，可选择特定的调度调整策略以长期保障 MS-DF 高效运行。

6.2 展望

在集群资源呈现更强的异构性、新理论带来数据流的多样性和数据流集成系统带来的通用性等发展背景下，虽然算子能够利用其自身的迭代、通信等特征来表达约束并在一定程度上保障调度目标，但其算子状态时空变化特点利用仍然不够充分和智能，当前集群调度系统也比较依赖于机器学习领域知识。针对这些问题，本小节展望未来可能的研究方向，具体如下三个方面所示：

(1) 算子异构资源需求多层次分析及协同刻画

作业、服务、算子组合、算子等不同层次的资源需求供给方法，逐渐从传统的静态资源预留转移到结合算子特征的动态、异构资源供给，能在各自层次内取得理论最优结果。然而对集群整体而言，单一层次的最优并不能保障全局最优调度决策。机器学习应用的不断出现，导致算子依赖关系、组合方法等发生变化，进而需要重新刻画算子资源需求，如果服务、作业等资源需求不能随之按需调整，

多层次资源就无法有效协同，无论如何准确刻画单个算子资源需求，也会存在大量的资源浪费或资源冲突；同时，如果没有对应新算子运算逻辑的资源驱动，盲目增加高性能异构资源也不会带来预期的加速效果。因此，如何对算子所在的多层次资源进行充分分析，整合各类定制化的异构资源驱动，围绕算子实现全局异构资源的多层次协同，是状态可变数据流集群调度需要重点解决的问题。

(2) 算子复杂调度约束的灵活定义和发现

决策模型求解时，一方面由于最优化、博弈论等理论的引入，调度正确性已得到数学证明，决策模型可在确定的需求、供给和约束情况下得到最优调度结果；另一方面由于算子状态的频繁变化，使得算子的需求、供给和调度约束总在不断更新。因此，算子调度决策模型除考虑决策的正确性外，如何灵活实现需求、供给和约束的定义和发现，及在决策搜索空间中快速找到最优调度结果，也是状态可变数据流集群调度需要重点解决的问题。

(3) 学习驱动的算子低成本调度调整

迁移、伸缩等调度调整策略在早期 Web 应用等环境中就验证了其自身有效性，但需要充分结合算子在不同状态同步方式后，才能定位算子调度时的瓶颈算子。调度调整大都依赖人工分析得到的具体应用类型和运行时上下文，其适用场景有限。未来还需要尝试机器学习方法来代替人工分析，扩展场景适用性，如传统数据流工作中，使用 SVM 预测数据倾斜发生位置，基于强化学习预测资源干扰，基于深度学习同时保障数据流资源使用效率和完成时间等，都可以在引入算子状态变化的基础上适用于更多的状态同步方式。因此，如何借鉴其他调度应用环境中的策略类型和策略组合方法，结合算子状态变化特点，智能化选择调度调整策略，是状态可变数据流集群调度需要重点解决的问题。

致谢 感谢各位审稿人的专业、严谨评审。本综述得到国家重点研发计划(2018YFB1003602)、国家自然科学基金(61872344)、北京市自然科学基金(4182070)、中科院青促会人才专项(2018144)资助，以及阿里巴巴 2018 年度创新研究（AIR）项目的支持。

参 考 文 献

[1] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning//12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), USENIX. Savannah, USA, 2016: 265-283.

[2] Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems, 2019, 32: 8026-8037.

[3] Top 10 Strategic Technology Trends for 2020: A Gartner Trend Insight Report, Gartner, 2021, https://www.gartner.com/en/documents/3981959/top-10-strategic-technology-trends-for-2020-a-gartner-tr.

[4] Xiao W, Bhardwaj R, Ramjee R, et al. Gandiva: Introspective cluster scheduling for deep learning//13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18), USENIX. Carlsbad, USA, 2018: 595-610.

[5] Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator//Proceedings of the 4th annual Symposium on Cloud Computing. Santa Clara, USA, 2013: 1-16.

[6] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), USENIX. Oakland, USA, 2012: 15-28.

[7] Chunliang H, Jie S, Heng Z, et al. Structures and State-of-Art Research of Cluster Scheduling in Big Data Background. Journal of Computer Research and Development, 2018, 55(1): 53. (in Chinese)
郝春亮, 沈捷, 张珩,等. 大数据背景下集群调度结构与研究进展, 计算机研究与发展, 2018, 55(1): 53.

[8] Adhikari M, Amgoth T and Srirama S N, A survey on scheduling strategies for workflows in cloud environment and emerging trends, ACM Computing Surveys (CSUR), 2019, 52(4): 1-36.

[9] Wang KJ, Jia T, Li Y, State-of-the-art survey of scheduling and resource management technology for colocation jobs.Journal of Software, 2020, 31(10): 3100–3119. (in Chinese)
王康瑾, 贾统, 李影, 在离线混部作业调度与资源管理技术研究综述, 软件学报, 2020, 31(10): 3100-3119.

[10] Grehant X, Demeure I and Jarp S, A survey of task mapping on production grids, ACM Computing Surveys (CSUR), 2013, 45(3): 1-25.

[11] Weerasiri D, Barukh M C, Benatallah B, et al. A taxonomy and survey of cloud resource orchestration techniques, ACM Computing Surveys (CSUR), 2017, 50(2): 1-41.

[12] Wenxin L, Heng Q, Renhai X, et al. Data Center Network Flow Scheduling Progress and Trends, Chinese Journal of Computers, 2020, 043(004): 600-617. (in Chinese)
李文信, 齐恒, 徐仁海, et al. 数据中心网络流量调度的研究进展与趋势, 计算机学报, 2020, 43(04): 600-617.

[13] Ben-Nun T and Hoefler T, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, ACM Computing Surveys (CSUR), 2019, 52(4): 1-43.

[14] Zheng L, Jia C, Sun M, et al. Ansor: Generating high-performance tensor programs for deep learning//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 863-879.

[15] Mirhoseini A, Pham H, Le Q V, et al. Device placement optimization with reinforcement learning//International Conference on Machine Learning, PMLR. Sydney, Australia, 2017: 2430-2439.

[16] Garefalakis P, Karanasos K, Pietzuch P, et al. Medea: scheduling of long running applications in shared production clusters//Proceedings of the Thirteenth EuroSys Conference. Porto, Portugal, 2018: 1-13.

[17] Park J H, Yun G, Chang M Y, et al. Hetpipe: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism//2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), USENIX, 2020: 307-321.

[18] Alipourfard O, Liu H H, Chen J, et al. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics//14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), USENIX. Boston, USA, 2017: 469-482.

[19] Li Y, Han Z, Zhang Q, et al. Automating cloud deployment for deep learning inference of real-time online services//IEEE INFOCOM 2020-IEEE Conference on Computer Communications. Toronto, Canada, 2020: 1668-1677.

[20] Venkataraman S, Yang Z, Franklin M, et al. Ernest: Efficient performance prediction for large-scale advanced analytics//13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), USENIX. Santa Clara, USA, 2016: 363-378.

[21] Shi J, Zou J, Lu J, et al. Mrtuner: a toolkit to enable holistic optimization for mapreduce jobs, Proceedings of the VLDB Endowment, 2014, 7(13): 1319-1330.

[22] Tang C, Yu K, Veeraraghavan K, et al. Twine: a unified cluster management system for shared infrastructure//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 787-803.

[23] Zhang C, Li S, Xia J, et al. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning//2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), USENIX, 2020: 493-506.

[24] Chai Z, Chen Y, Zhao L, et al. Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data, arXiv preprint arXiv:2010.05958, 2020.

[25] Wang H, Niu D and Li B, Dynamic and decentralized global analytics via machine learning//Proceedings of the ACM Symposium on Cloud Computing. Carlsbad, USA, 2018: 14-25.

[26] Peng Y, Bao Y, Chen Y, et al. Optimus: an efficient dynamic resource scheduler for deep learning clusters//Proceedings of the Thirteenth EuroSys Conference. Porto, Portugal, 2018: 1-14.

[27] Yi X, Luo Z, Meng C, et al. Fast Training of Deep Learning Models over Multiple GPUs//Proceedings of the 21st International Middleware Conference. Delft, The Netherlands, 2020: 105-118.

[28] Holmes C, Mawhirter D, He Y, et al. Grnn: Low-latency and scalable rnn inference on gpus//Proceedings of the Fourteenth EuroSys Conference 2019. Dresden, Germany, 2019: 1-16.

[29] Jeon B, Cai L, Srivastava P, et al. Baechi: fast device placement of machine learning graphs//Proceedings of the 11th ACM Symposium on Cloud Computing. USA, 2020: 416-430.

[30] Addanki R, Venkatakrishnan S B, Gupta S, et al. Placeto: Learning generalizable device placement algorithms for distributed machine learning, arXiv preprint arXiv:1906.08879, 2019.

[31] Gao Y, Chen L and Li B, Spotlight: Optimizing device placement for training deep neural networks//International Conference on Machine Learning, PMLR. Stockholm, Sweden, 2018: 1676-1684.

[32] Sridhar V, Subramanian S, Arteaga D, et al. Model governance: Reducing the anarchy of production {ML}//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), USENIX. Boston, USA, 2018: 351-358.

[33] Dai J J, Wang Y, Qiu X, et al. Bigdl: A distributed deep learning framework for big data//Proceedings of the ACM Symposium on Cloud Computing. Santa Cruz, USA, 2019: 50-60.

[34] Ma L, Xie Z, Yang Z, et al. Rammer: Enabling Holistic Deep Learning Compiler Optimizations with rTasks//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 881-897.

[35] Zhang M, Rajbhandari S, Wang W, et al. Deepcpu: Serving rnn-based deep learning models 10x faster//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), USENIX. Boston, USA, 2018: 951-965.

[36] Rao J, Wang K, Zhou X, et al. Optimizing virtual machine scheduling in NUMA multicore systems//2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). Shenzhen, China, 2013: 306-317.

[37] Li C, Dakkak A, Xiong J, et al. Xsp: Across-stack profiling and analysis of machine learning models on gpus//2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). New Orleans, USA, 2020: 326-327.

[38] Guo F, Li Y, Lui J C, et al. DCUDA: Dynamic GPU Scheduling with Live Migration Support//Proceedings of the ACM Symposium on Cloud Computing. Santa Cruz, USA, 2019: 114-125.

[39] Fu H, Tang S, He B, et al. GLP4NN: A convergence-invariant and network-agnostic light-weight parallelization framework for deep neural networks on modern GPUs//Proceedings of the 47th International Conference on Parallel Processing. Eugene, USA, 2018: 1-10.

[40] Hu Y, Rallapalli S, Ko B, et al. Olympian: Scheduling gpu usage in a deep neural network model serving system//Proceedings of the 19th International Middleware Conference. Rennes, France, 2018: 53-65.

[41] Ousterhout K, Canel C, Ratnasamy S, et al. Monotasks: Architecting for performance clarity in data analytics frameworks//Proceedings of the 26th Symposium on Operating Systems Principles. Shanghai, China, 2017: 184-200.

[42] Jin T, Cai Z, Li B, et al. Improving resource utilization by timely fine-grained scheduling//Proceedings of the Fifteenth European Conference on Computer Systems. Heraklion, Greece, 2020: 1-16.

[43] Hashemi S H, Jyothi S A and Campbell R H, Tictac: Accelerating distributed deep learning with communication scheduling, arXiv preprint arXiv:1803.03288, 2018.

[44] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: generalized pipeline parallelism for DNN training//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville, Canada, 2019: 1-15.

[45] Liaw R, Bhardwaj R, Dunlap L, et al. Hypersched: Dynamic resource reallocation for model development on a deadline//Proceedings of the ACM Symposium on Cloud Computing. Santa Cruz, USA, 2019: 61-73.

[46] Sharma P, Ali-Eldin A and Shenoy P, Resource deflation: A new approach for transient resource reclamation//Proceedings of the Fourteenth EuroSys Conference 2019. Dresden, Germany, 2019: 1-17.

[47] Sivathanu M, Chugh T, Singapuram S S, et al. Astra: Exploiting predictability to optimize deep learning//Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. Providence, USA, 2019: 909-923.

[48] Liu Q and Yu Z, The elasticity and plasticity in semi-containerized co-locating cloud workload: a view from alibaba trace//Proceedings of the ACM Symposium on Cloud Computing. Carlsbad, USA, 2018: 347-360.

[49] Reiss C, Tumanov A, Ganger G R, et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis//Proceedings of the third ACM symposium on cloud computing. San Jose, USA, 2012: 1-13.

[50] Cortez E, Bonde A, Muzio A, et al. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms//Proceedings of the 26th Symposium on Operating Systems Principles. Shanghai, China, 2017: 153-167.

[51] Crankshaw D, Sela G-E, Mo X, et al. InferLine: latency-aware provisioning and scaling for prediction serving pipelines//Proceedings of the 11th ACM Symposium on Cloud Computing. USA, 2020: 477-491.

[52] Gujarati A, Karimi R, Alzayat S, et al. Serving DNNs like clockwork: Performance predictability from the bottom up//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 443-462.

[53] Tirmazi M, Barker A, Deng N, et al. Borg: the next generation//Proceedings of the Fifteenth European Conference on Computer Systems. Heraklion, Greece, 2020: 1-14.

[54] Guan Y, Li C and Tang X, On max-min fair resource allocation for distributed job execution//Proceedings of the 48th International Conference on Parallel Processing. Kyoto, Japan, 2019: 1-10.

[55] Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types//NSDI'2011, USENIX. Boston, USA, 2011: 24-24.

[56] Chowdhury M, Liu Z, Ghodsi A, et al. {HUG}: Multi-resource fairness for correlated and elastic demands//13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), USENIX. Santa Clara, USA, 2016: 407-424.

[57] Wang L and Wang W, Fair coflow scheduling without prior knowledge//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Vienna, Austria, 2018: 22-32.

[58] Chaudhary S, Ramjee R, Sivathanu M, et al. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning//Proceedings of the Fifteenth European Conference on Computer Systems. Heraklion, Greece, 2020: 1-16.

[59] Narayanan D, Santhanam K, Kazhamiaka F, et al. Heterogeneity-aware cluster scheduling policies for deep learning workloads//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 481-498.

[60] Mahajan K, Balasubramanian A, Singhvi A, et al. Themis: Fair and efficient {GPU} cluster scheduling//17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), USENIX. Santa Clara, USA, 2020: 289-304.

[61] Gu J, Chowdhury M, Shin K G, et al. Tiresias: A {GPU} cluster manager for distributed deep learning//16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19), USENIX. Boston, USA, 2019: 485-500.

[62] Delgado P, Didona D, Dinu F, et al. Kairos: Preemptive data center scheduling without runtime estimates//Proceedings of the ACM Symposium on Cloud Computing. Carlsbad, USA, 2018: 135-148.

[63] Sultana A, Chen L, Xu F, et al. E-LAS: Design and Analysis of Completion-Time Agnostic Scheduling for Distributed Deep Learning Cluster//49th International Conference on Parallel Processing-ICPP. Edmonton, Canada, 2020: 1-11.

[64] Scully Z, Grosof I and Harchol-Balter M, Optimal multiserver scheduling with unknown job sizes in heavy traffic, Performance Evaluation, 2021, 145: 102150.

[65] Grandl R, Kandula S, Rao S, et al. {GRAPHENE}: Packing and dependency-aware scheduling for data-parallel clusters//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), USENIX. Savannah, USA, 2016: 81-97.

[66] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling//Proceedings of the 5th European conference on Computer systems. Paris, France, 2010: 265-278.

[67] Yu P and Chowdhury M, Salus: Fine-grained gpu sharing primitives for deep learning applications, arXiv preprint arXiv:1902.04610, 2019.

[68] Bao Y, Peng Y and Wu C, Deep learning-based job placement in distributed machine learning clusters//IEEE INFOCOM 2019-IEEE conference on computer communications. Paris, France, 2019: 505-513.

[69] Kim S and Kim Y, Co-scheml: interference-aware container co-scheduling scheme using machine learning application profiles for GPU clusters//2020 IEEE International Conference on Cluster Computing (CLUSTER). Kobe, Japan, 2020: 104-108.

[70] Zhao H, Han Z, Yang Z, et al. Hived: sharing a {GPU} cluster for deep learning with guarantees//14th {USENIX} symposium on operating systems design and implementation ({OSDI} 20), USENIX. 2020: 515-532.

[71] Rasley J, Karanasos K, Kandula S, et al. Efficient queue management for cluster scheduling//Proceedings of the Eleventh European Conference on Computer Systems. London, UK, 2016: 1-15.

[72] Wang Z, Li H, Li Z, et al. Pigeon: An effective distributed, hierarchical datacenter job scheduler//Proceedings of the ACM Symposium on Cloud Computing. Santa Cruz, USA, 2019: 246-258.

[73] Grandl R, Ananthanarayanan G, Kandula S, et al. Multi-resource packing for cluster schedulers, ACM SIGCOMM Computer Communication Review, 2014, 44(4): 455-466.

[74] Isard M, Prabhakaran V, Currey J, et al. Quincy: fair scheduling for distributed computing clusters//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. New York, USA, 2009: 261-276.

[75] Le T N, Sun X, Chowdhury M, et al. AlloX: compute allocation in hybrid clusters//Proceedings of the Fifteenth European Conference on Computer Systems. Heraklion, Greece, 2020: 1-16.

[76] Gog I, Schwarzkopf M, Gleave A, et al. Firmament: Fast, centralized cluster scheduling at scale//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), USENIX. Savannah, USA, 2016: 99-115.

[77] Chen X, Wu H, Wu Y, et al. Large-scale resource scheduling based on minimum cost maximum flow. Journal of Software, 2017, 28(3): 598-610. (in Chinese)
陈晓旭, 吴恒, 吴悦文, 等. 基于最小费用最大流的大规模资源调度方法. 软件学报, 2017, 28(3):598-610.

[78] Wu H, Zhang W, Xu Y, et al. Aladdin: optimized maximum flow management for shared production clusters//2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Rio de Janeiro, Brazil, 2019: 696-707.

[79] Grandl R, Chowdhury M, Akella A, et al. Altruistic scheduling in multi-resource clusters//12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), USENIX. Savannah, USA, 2016: 65-80.

[80] Hu Z, Li B, Chen C, et al. Flowtime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Vienna, Austria, 2018: 929-938.

[81] Park J W, Tumanov A, Jiang A, et al. 3sigma: distribution-based cluster scheduling for runtime uncertainty//Proceedings of the Thirteenth EuroSys Conference. Porto, Portugal, 2018: 1-17.

[82] Janus P and Rzadca K, Slo-aware colocation of data center tasks based on instantaneous processor requirements//Proceedings of the 2017 Symposium on Cloud Computing. Santa Clara, USA, 2017: 256-268.

[83] Tumanov A, Zhu T, Park J W, et al. TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters//Proceedings of the Eleventh European Conference on Computer Systems. London, UK, 2016: 1-16.

[84] Zhang H, Stafman L, Or A, et al. Slaq: quality-driven scheduling for distributed machine learning//Proceedings of the 2017 Symposium on Cloud Computing. Santa Clara, USA, 2017: 390-404.

[85] Karanasos K, Rao S, Curino C, et al. Mercury: Hybrid centralized and distributed scheduling in large shared clusters//2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15), USENIX. Santa Clara, USA, 2015: 485-497.

[86] Ousterhout K, Wendell P, Zaharia M, et al. Sparrow: distributed, low latency scheduling//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. Farmington, USA, 2013: 69-84.

[87] Kubernetes, Linux Foundation, 2021, https://kubernetes.io/zh/docs/home/.

[88] Jiang Y, Zhu Y, Lan C, et al. A Unified Architecture for Accelerating Distributed {DNN} Training in Heterogeneous GPU/CPU Clusters//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 463-479.

[89] Xiao W, Ren S, Li Y, et al. AntMan: Dynamic Scaling on {GPU} Clusters for Deep Learning//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), AntMan: Dynamic Scaling on {GPU} Clusters for Deep Learning. 2020: 533-548.

[90] Nicolae B, Wozniak J M, Dorier M, et al. DeepClone: Lightweight state replication of deep learning models for data parallel training//2020 IEEE International Conference on Cluster Computing (CLUSTER). Kobe, Japan, 2020: 226-236.

[91] Dhakal A, Kulkarni S G and Ramakrishnan K, Gslice: controlled spatial sharing of gpus for a scalable inference platform//Proceedings of the 11th ACM Symposium on Cloud Computing. USA. 2020: 492-506.

[92] Peng X, Shi X, Dai H, et al. Capuchin: Tensor-based gpu memory management for deep learning//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland, 2020: 891-905.

[93] Lin Z, Li C, Miao Y, et al. Pagraph: Scaling gnn training on large graphs via computation-aware caching//Proceedings of the 11th ACM Symposium on Cloud Computing. USA, 2020: 401-415.

[94] Liu J, Li D, Kestor G, et al. Runtime concurrency control and operation scheduling for high performance neural network training//2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Rio de Janeiro, Brazil, 2019: 188-199.

[95] Yeh T-A, Chen H-H and Chou J, KubeShare: a framework to manage GPUs as first-class and shared resources in container cloud//Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing. Stockholm, Sweden, 2020: 173-184.

[96] vcuda-controller, tkestack, 2021, https://github.com/tkestack/vcuda-controller.

[97] k8s-device-plugin, NVIDIA, 2021, https://github.com/NVIDIA/k8s-device-plugin.

[98] Bai Z, Zhang Z, Zhu Y, et al. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications//14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), USENIX. 2020: 499-514.

[99] Wang Y, Wei G-Y and Brooks D, A Systematic Methodology for Analysis of Deep Learning Hardware and Software Platforms//MLSys, mlsys.org. Austin, USA, 2020.

[100] Zhao X, Yao J, Gao P, et al. Efficient sharing and fine-grained scheduling of virtualized GPU resources//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Vienna, Austria, 2018: 742-752.

[101] Suzuki Y, Yamada H, Kato S, et al. Gloop: an event-driven runtime for consolidating gpgpu applications//Proceedings of the 2017 Symposium on Cloud Computing. Santa Clara, USA, 2017: 80-93.

[102] Wagenländer M, Mai L, Li G, et al. Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources//12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20), USENIX. Boston, USA, 2020.

[103] Li S, Walls R J and Guo T, Characterizing and modeling distributed training with transient cloud gpu servers//2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). Singapore, 2020: 943-953.

[104] Luo Q, He J, Zhuo Y, et al. Prague: High-performance heterogeneity-aware asynchronous decentralized training//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland, 2020: 401-416.

[105] Yang D, Rang W and Cheng D, Mitigating Stragglers in the Decentralized Training on Heterogeneous Clusters//Proceedings of the 21st International Middleware Conference. Delft, The Netherlands, 2020: 386-399.

[106] Nakandala S, Zhang Y and Kumar A, Cerebro: A data system for optimized deep learning model selection, Proceedings of the VLDB Endowment, 2020, 13(12): 2159-2173.

[107] Chen T, Moreau T, Jiang Z, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning//13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18), USENIX. Carlsbad, USA, 2018: 578-594.

[108] Jiang X, Wang H, Chen Y, et al. Mnn: A universal and efficient inference engine, arXiv preprint arXiv:2002.12418, 2020.

附录X.

XU Yuan-Jia, Ph.D. candidate. His research interests include resource scheduling， distributed system and machine learning system.

WU Heng, Ph.D., associate professor. His current research interests include container based virtualization and edge computing.

WU Yue-Wen, Ph.D. candidate. His current research interests include performance modeling and machine learning based cloud configuration recommendation.

ZHANG Wen-Bo, Ph.D., professor, Ph.D. supervisor. His current research interests include cloud computing and service computing.

YANG Chen, Master candidate. His research interests include resource scheduling and distributed system.

WANG Tao, Ph.D., associate professor. His current research interests include micro-service monitoring and service computing.

Background

With the rapid development of Machine Learning, it is widely used in various areas such as speech recognition, image processing and so on. Currently, machine learning systems introduce operation, which works as a key factor to decouple the computation of machine learning and resources allocation. In this article, we abstract the data processing procedure of operation as Mutable States Data Flow. Researches show that allocating resources to operations in data flow reasonably and effectively can improve resource efficiency and reduce costs of data center.

Efficient cluster scheduling for MS-DF with good quality improves heterogeneous cluster resource utilization and minimize the execution time of machine learning. However, MS-DF, with its diverse resource usage in time and space, brings several challenges to traditional cluster scheduling in three aspects: (i) mechanisms to discover operation heterogeneous resource requirements, (ii) scheduling decisions with various constraint, model and algorithm settings, (iii) scheduling adjustment strategies to improve scheduling performance. To discover bottlenecks caused by operations, we conduct a holistic survey on recent research progress of MS-DF by analyzing and summarizing from these aspects mentioned above. At last, we give our prospective view of key technologies of cluster scheduling for MS-DF.

This research was partially supported by National Key Research and Development Program of China (2018YFB1003602), National Natural Science Foundation of China (61872344), Natural Science Foundation of Beijing (4182070), Youth Innovation Promotion Association at CAS (2018144), and Alibaba Innovative Research 2018.