高技术通讯 2024 年 第34 卷 第5 期:475-485

doi:10.3772/j.issn.1002-0470.2024.05.004

基于任务资源需求预测的人工智能算力调度①

杨明烜②*** 洪学海③* 唐宏伟 * ***

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

(***中国科学院大学南京学院 南京 211135)

摘 要 为提升人工智能(AI)算力的任务执行效率和资源利用率,本文提出一种基于任务资源需求预测的AI算力调度方法,指导资源调度过程。相比于以往大多数研究工作仅围绕着图形处理器(GPU)资源设计的AI算力调度方法,本文充分考虑了多个维度资源对用户任务运行效率和计算集群资源利用的影响。本文基于机器学习方法构建任务资源需求预测模型,分析多维度资源对任务性能的影响,进而完成自适应资源伸缩调度,解决用户超额申请问题。实验结果表明,在相同时间内,该方法实现了更多任务的部署和执行。任务部署量提升25.3%,部署任务的完成率提升15.2%,GPU和内存利用率分别提升7.2%和8.0%,提升了算力资源的总体利用率。

关键词 资源调度;弹性资源分配;人工智能(AI);算力

为了高效训练大规模人工智能(artificial intelligence, AI)机器学习任务(简称AI任务),研究机构、科技企业、云厂商常常构建包含大量图形处理器(graphics processing unit, GPU)、神经网络处理器(neural-network processing unit, NPU)等硬件加速器的大规模计算集群,用于AI任务的增速提效。但从成本与技术角度考虑,AI算力的能力提升不能只依赖于算力硬件资源规模的增长,应该采用新的技术手段,面向现有AI算力资源,实现计算效率的提升。因而,AI算力调度技术就成为需要研究的课题。

AI算力调度,即AI算力资源所构成集群硬件资源和计算任务匹配的调度,可用于提升AI算力的资源利用率和任务训练效率。这项研究近年来逐渐成为人工智能领域研究的重点之一。越来越多的面向AI算力调度的研究方案被提出[1-5],这些工作通过对AI任务负载的特性分析,调整集群调度策略实现调度目标。然而,现有研究工作都将算力集群的GPU硬件资源作为影响任务性能表现的主要因素,忽视了算力集群中的中央处理器(central processing unit, CPU)、内存、网络等其他维度资源的影响。AI算力调度需要考虑算力集群多维度资源的分配和利用。

1 相关工作

此前相关研究工作中,关于AI算力集群调度的研究是在面向大数据任务处理的传统集群调度方法的基础上,根据AI任务特性对GPU资源的分配进行优化,从而提升AI任务性能或GPU资源利用率。文献[1]的调度器考虑了AI任务在训练过程中的周期性特性,以指导GPU分配和共享,实现GPU资源利用率的提升。文献[2]的调度器在设计中考虑

① 国家重点研发计划(2016YFC1401706)资助项目。

② 女,1999年生,博士生;研究方向:分布式计算;E-mail: yangmingxuan20s@ict.ac.cn。

③ 通信作者,E-mail: hxh@ict.ac.cn。

(收稿日期:2023-04-14)

了 AI 任务的抢占特性和位置敏感特性,实现了不同类型的训练任务不同需求下的高效调度。文献[3]基于资源伸缩调整和高效的状态复制,实现了弹性资源调度,该方法有效减少了任务排队时间并提升了资源利用率。文献[4]从 AI 任务性能建模角度入手,设计调度策略,结合任务收敛速度,完成了资源分配和任务放置的动态决策。

然而,这些调度器的设计性能,不能在实际的生产集群中得到完全实现。这是由于 AI 算力调度中资源利用率和任务执行效率,不是只考虑单一的 GPU 资源分配就能够决定的,而是由多维度资源共同作用下所决定的。文献[5]考虑不同 CPU 和内存分配对 AI 任务的影响,将任务对于不同资源的敏感程度引入调度策略,实现了任务性能的提升,但该工作未能考虑多维度硬件资源的利用效率等集群层面资源分配问题。

为了提升硬件资源利用率,大规模 AI 生产集群架构设计通常会考虑多维度资源的影响。很多研究工作面向非硬件加速器(如 GPU 等)资源维度的硬件进行优化,如增加内存容量、内存带宽和 CPU 计算能力等等[6-7]。因此,AI 算力集群调度研究也需要考虑多维度资源影响。

针对上述相关工作,本文认为,研究一种新的基于任务资源需求预测的 AI 算力调度方法非常必要。该方法能够考虑多维度资源的影响,提升用户侧的 AI 任务执行效率,优化资源侧的 AI 算力资源利用率。首先建立任务多维度资源需求预测模型,衡量多维度资源分配对任务执行效率的影响;其次基于该模型,提出一种自适应资源伸缩调度算法及系统设计,以有效优化 AI 算力资源调度分配。

2 AI 任务资源需求预测模型

2.1 任务资源需求概述

任务资源需求是用户根据任务执行所需,在提交任务时申请的资源。调度过程基于任务资源需求进行资源分配。对于 AI 算力调度而言,任务资源需求具体包括 GPU 申请量、CPU 申请量、内存申请量、网络带宽、IO 占用等等。

表 1 对 AI 算力生产集群[6]4 种典型任务的资源需求情况进行数据分析,反映了不同任务类型的资源利用情况。可以看出,不同类型任务的多维度资源需求特征不同。所以,调度策略的设计也需要考虑不同任务的多维度资源特性,保障资源的高效利用。任务类型划分和定义提供了数据层面的支撑依据。

表 1 生产集群日志中常用 AI 任务负载的资源情况

任务负载	CPU 平均用量/核	GPU 平均用量/%	内存平均用量/GB
推荐任务	3.9	6.8	2.8
图神经网络	11.9	0.8	9.5
自然语言处理	10.6	34.1	10.0
图像分类	2.1	30.2	19.9

基于上述任务类型的资源需求,本文将 AI 任务类型划分为 GPU 密集型、CPU 密集型和内存密集型 3 类,同时调度过程涉及的任务资源需求也聚焦在 GPU、CPU 和内存 3 个维度。其中,GPU 密集型的神经网络机器翻译任务需要大量 GPU 资源支撑。相反,CPU 密集型的推荐任务整体执行时间的 56% 浪费在等待训练数据输入中[8],有些情况甚至超过了 80%[6];内存密集型的图像分类任务读取数据过程中访存的开销较大,任务训练过程中 10% ~70% 的时间用于数据的读取[9]。也就是说,这些任务对 GPU 资源的利用效率受到了其他维度资源(如 CPU、内存等)的影响。

因此,需要考虑多维度资源影响,建立任务资源需求预测模型,针对不同类型任务的多维度资源特征设计 AI 算力调度策略。

2.2 任务资源需求预测模型建立

为了刻画不同类型 AI 任务的多维度资源特征,构建任务资源需求预测模型,用于衡量多维度资源对不同任务执行效率的影响,指导 AI 算力调度中的自适应资源伸缩过程。

首先,选取构建任务资源需求预测模型所需的数据集。表 2 对比了 3 种 GPU 集群的日志数据集,其中 PAI[6]是 Alibaba 的 AI 算力集群、Helios[7]是商汤科技的深度学习集群、Philly[10]是微软的深度学习集群。Alibaba 的 AI 算力集群的日志数据,包括任务执行中多个维度资源的分配细节,具体包括

GPU 申请量、CPU 申请量、内存申请量、网络带宽、IO 占用等。该数据集还包含了任务类型数据,而其他数据集该内容缺失。综上,任务资源需求预测模型的构建选用 Alibaba 的 AI 算力集群日志数据。

表 2　不同 GPU 集群日志数据集的对比

GPU 集群	PAI	Helios	Philly
GPU 总数量	6 742	6 416	128
时长/d	60	180	83
资源种类	>3 种	2 种	1 种
任务类型	有	无	无

其次,从 Alibaba 的 AI 算力集群数据集中,选择任务多维度资源占用的 m 个特征及 n 个样本,用于构建多维度资源下任务资源需求的回归模型。回归模型数据的特征值包括 CPU 核数、GPU 百分比占用率、内存容量需求、网络流量、显存百分比占用率;而模型数据的目标值 y_i 为第 i 个样本的执行时间。

建模回归过程基于极致梯度提升(extreme gradient boosting,XGBoost)[11]的机器学习方法,分析不同类型任务多维度资源需求,即多维度资源对任务执行时间的影响。XGBoost 是基于提升(即 Boosting)方法来训练一组决策树,并利用梯度提升方法对决策树的残差进行迭代拟合。XGBoost 模型得到的输出结果是所有残差迭代决策树预测值的加权和。基于 XGBoost 的任务资源需求预测模型输出表达式如下:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{1}$$

其中,$\hat{y}_i$ 为 t 次迭代之后的预测值,也就是模型根据资源情况所预测的任务执行时间;k 为迭代得到的决策树数量;f 为每个决策树的输出函数。

基于 XGBoost 的任务资源需求预测模型目标函数为

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{2}$$

其中,l 为衡量模型拟合度的损失函数;Ω 为降低误差损失引入的正则化部分,用于衡量模型的复杂性。

基于 XGBoost 的任务资源需求模型通过迭代 t 次拟合模型的残差,提升决策树模型精度,通过二阶泰勒展开该迭代过程,得到新的迭代目标函数表达式如式(3)所示。

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3}$$

其中,g 为一阶偏导,h 为二阶偏导。

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{4}$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{5}$$

推导得到该函数极值点为

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{6}$$

$$G_j = \sum_{i \in I_j} g_i \tag{7}$$

$$H_j = \sum_{i \in I_j} h_i \tag{8}$$

最终得到基于 XGBoost 的任务资源需求预测模型目标函数极值为

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{9}$$

其中,λ、γ 为超参数,T 表示叶子节点的个数。

综上,任务资源需求预测模型的建模过程需要在最小化目标函数值的基础上,确立模型结构。具体方法是通过最大化分支增益,来选择叶子节点的特征值切分点,为模型中各个特征依次构建树模型的新分支,最终完成建模任务资源需求 XGBoost 模型。分支增益是叶子节点的特征值切分前后目标函数值的差值,表达式为

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma \tag{10}$$

每次选择分支增益最大的特征作为叶子节点的切分特征,依次选择资源特征完成任务资源需求预测模型构建,这样得到的模型总目标函数最小,最终完成数据拟合预测的任务资源需求预测模型建立。

2.3　任务资源需求预测模型结果分析

为了验证任务资源需求预测模型的准确度,选用在数据预测回归中常用的随机森林算法与 2.2 节的 XGBoost 建模方法进行对比。模型准确度评价指标选用了平均绝对误差(mean absolute error, MAE)和加权平均绝对误差百分比(weighted mean absolute percentage error, WMAPE),它们分别衡量模型的拟

合效果及模型预测结果和真实数据间的差异。

MAE 和 WMAPE 表达式分别如下：

$$MAE(y,y') = \frac{1}{m}\sum_{I=1}^{m} | y' - y | \tag{11}$$

$$WMAPE(y,y') = \frac{\sum_{I=1}^{m} | y' - y |}{\sum_{I=1}^{m} y} \tag{12}$$

其中，y' 为预测值，y 为真实值。

两类方法都是决策树集成学习方法，从实现原理分析，XGBoost 是基于提升（即 Boosting）方法来训练一组决策树，利用梯度提升方法对决策树的残差进行迭代拟合。随机森林方法使用装袋（即 Bagging）方法完成一组决策树的训练，每次随机选取数据集中部分数据进行决策树回归。随机森林方法能够最小化回归方差和过拟合，而 XGBoost 能够最小化偏差和欠拟合。

任务资源需求预测模型需要更高的回归准确度，用于调度过程的资源分配，所以选用的评价指标更加关注回归拟合的偏差而不是方差。因此在表 3 的结果中，2.2 节的 XGBoost 建模方法的结果优于随机森林回归建模方法。基于 XGBoost 方法建模任务资源需求预测模型，能够更准确预测多维度资源分配和任务执行效率之间的关系，以指导 AI 算力的自适应资源伸缩调度。

表 3 回归模型的结果对比

回归模型	任务类型	*MAE*	*WMAPE*
XGBOOST	CTR	0.16	0.13
	ResNet	1.45	0.15
	NMT	0.75	0.11
随机森林	CTR	0.17	0.14
	ResNet	1.46	0.16
	NMT	1.01	0.15

注：CTR：click-through rate

NMT：neural machine translation

3 AI 算力自适应资源伸缩调度

3.1 AI 算力调度概述

基于任务资源需求预测的 AI 算力调度是一种面向 AI 算力的自适应资源伸缩调度方法。与传统的面向通用大数据任务处理的集群调度方法相比，AI 算力的调度方法需要充分考虑 AI 任务的任务特性，用于降低实际的任务执行时间，提升 GPU 资源的利用率。

本调度方法通过任务资源需求模型来构建这种任务特性，分析多维度资源对任务性能的影响，调度过程中指导 GPU 集群的自适应动态伸缩资源分配。这种动态调整资源调度的方法，可以优化用户任务申请过程中出现的资源超额申请情况。同时，该方法也优化了由于各维度资源之间的不匹配而导致 AI 任务部分维度资源的低效利用问题，最终提升了 AI 算力实际部署任务的效率和集群资源利用效率。

基于任务资源需求预测的 AI 算力调度可以形式化抽象为 3.2 节的 AI 算力多维度资源调度问题定义。在此基础上，引入任务资源需求预测模型实现自适应动态资源伸缩调度，该调度设计将在 3.3 节进行详细介绍，并于 3.4 节介绍调度系统的整体架构。

3.2 AI 算力多维度资源调度问题定义

AI 算力多维度资源调度的问题定义如下。

假设存在一个 AI 算力集群，具有 M 个不同资源配额的异构服务器列表 $S = \{S_1, S_2, \cdots, S_M\}$，第 i 个服务器对应的 R 维资源容量表示为 $S_i = \{S_i^1, S_i^2, \cdots, S_i^R\}$。第 j 个用户任务对应的资源申请量为 $w_j = \{w_j^1, w_j^2, \cdots, w_j^R\}$。资源分配调度问题的形式化定义如式（13）所示。

$$\max \sum_{j=1}^{J}\sum_{i=1}^{M} p_j x_{i,j}$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{J} w_j^r x_{i,j} \leqslant S_i^r \quad r \in R, i \in M, j \in J \\ \sum_{i=1}^{M} x_{i,j} \leqslant 1 \qquad\qquad x_{i,j} \geqslant 0 \end{cases} \tag{13}$$

其中，变量 $x_{i,j}$ 表示第 j 个用户任务分配部署到第 i 个服务器，调度过程中，任务可以分布在多个服务器上，所以 $x_{i,j}$ 可以是分数；p_j 是任务 j 在当前调度轮次中的任务执行收益。

调度目标是最大化当前调度轮次的任务执行收益。同时，任务资源需求应小于服务节点可用资源

量,保证分配给任务的资源量不超过目前服务器多个维度的可用资源配额,另外也要保证用户任务可以在当前服务器节点集合中执行。

3.3 AI 算力自适应资源伸缩调度算法

AI 算力自适应资源伸缩调度算法结合任务资源需求预测模型,分析多维度资源分配对任务性能的影响,对当前动态队列中任务的多维度资源需求进行自适应伸缩调整,最终将任务部署到合适的集群位置。基于任务资源需求模型的自适应资源伸缩算法伪代码如算法 1 所示,整体 AI 算力的资源伸缩调度算法伪代码如算法 2 所示。

算法 1 自适应资源伸缩算法

输入:当前任务属性,包括任务类型 C_j,用户提交的任务资源申请量 $w_j = \{w_j^1, w_j^2, \cdots, w_j^R\}$,自适应调整率 α,资源伸缩对任务性能影响阈值 θ;

输出:伸缩调整后的资源申请结果. $W_j = \{W_j^1, W_j^2, \cdots, W_j^R\}$.

1. $model \leftarrow getJobModel(C_j)$;/* 当前任务对应的任务模型 */
2. $t_j \leftarrow getPerformance(model, w_j)$; /* 预期任务性能(执行时间) */
3. $important\_list \leftarrow permutation\_importance(model)$; /* 任务模型的排列重要性作为伸缩依据 */
4. while $\alpha < 1$:
5. for each resources type in w_j:
6. if type r is not in $important\_list$:
7. $w_j^r \leftarrow \alpha w_j^r$; /* 资源伸缩调整 */
8. endif
9. Endfor

/* 自适应调整之后的任务性能 */

10. $T_j \leftarrow getPerformance(model, W_j)$;
11. $error \leftarrow | T_j - t_j | / t_j$;
12. if $error < \theta$:

/* 资源伸缩符合任务性能影响的阈值要求,返回资源伸缩结果 */

13. return W_j;
14. endif
15. $\alpha = \alpha + \Delta\alpha$; /* 自适应伸缩率调整 */
16. return w_j;

自适应资源伸缩算法(即算法 1)是基于任务资源需求预测模型的特征重要性的分析,筛选出对该任务性能影响小且存在超额申请情况的资源维度。接下来,在不影响任务性能的前提下,对用户任务最初申请的多维度资源进行自适应的伸缩调整。该算法能够有效解决任务的资源超额申请和多维度资源之间的不匹配问题。

自适应资源伸缩中的特征重要性分析选用排列重要性(permutation importance, PIMP)[12]方法。该方法通过计算随机洗牌某个特征值时模型分数的下降,得到该特征值的预测重要性。机器学习模型的特征重要性能够分析输入各维度特征在目标变量回归中的重要程度和相关性,被广泛用于可解释性机器学习研究中。

相比于基于决策树分裂信息增益计算特征重要性的常规方法,PIMP 可以在未知数据上进行重要性计算,解决了信息增益方法由于过拟合而不能预测未知数据的问题。这样一来,调度过程中到达的新任务也可以基于模型进行重要性分析预测。此外,PIMP 方法除了能够得到特征对预测结果的重要性之外,还能够得到特征对预测结果的正负性影响。这种正负性可以作为资源伸缩调整的依据。

AI 算力的资源伸缩调度算法(即算法 2)先对调度轮次中任务执行资源伸缩调整,再选择部署的集群节点并更新。算法前 5 行筛选出当前调度轮次可调度的任务队列列表。第 7 行的算法根据任务类型和对应的资源需求预测模型,对任务的资源需求进行自适应伸缩调整,具体实现过程见算法 2。最终,第 9 行到第 14 行算法实现任务在集群节点服务器中的部署。第 15 行更新集群状态。

算法 2 AI 算力的资源伸缩调度算法

输入:当前轮次的任务队列 Job_list,服务器节点 $Server\_list$;

输出:任务调度结果.

1. $Job\_list \leftarrow getOnlineJob(t, Job\_list, Pending\_job)$;
2. if Job_list is null:
3. break;
4. Endif

/* 按照调度策略进行任务排序,例如策略为 FIFO 时则按照任务到达时间排序 */

5. $Job\_list \leftarrow sortJobList(Job\_list)$;
6. for each job in Job_list:

/* 根据任务类型完成自适应资源伸缩,更新调整资源申请量(算法 1) */

```
7.    job' ← updateResRequirement(job);
      /* 按可用资源量升序排列,降低资源碎片化 */
8.    sortServerList(Server _list);
9.    if isJobExecutable(Server _list, job'):
        /* 当前服务器集合可以部署该任务 */
10.       jobPlacement(Server _list, job');
11.       Server _list ← updateState (Server _list);
12.   elseif
        /* 下一调度轮次中再次调度 */
13.       Pending _job ← jobPending(job');
14.   endif
      /* 更新任务队列状态 */
15. Job _list ← updateJobState(Job _list);
16. end for
```

该调度算法复杂度的分析如下。首先假设算法中 1 ~4 行的队列初始化过程和 11、13、15 行的状态更新过程复杂度忽略不计。第 5 行基于 Timesort[13] 算法进行任务排序最多需要 $O(n \cdot \log n)$ 次操作,其中 n 代表任务数量。而在内层循环中,第 8 行代表节点堆排序过程,最多需要 $O(M)$ 次操作,其中 M 代表集群节点服务器数量。第 9 行结合可用服务器状态完成任务部署的确认过程,最多需要 $O(M \cdot R)$ 次操作,其中 R 为需要考虑的资源维度大小。另外,第 7 行任务资源需求自适应伸缩调整过程,需要 $O(t \cdot \log F)$ 次操作,其中 t 为自适应伸缩轮次,F 为构造机器学习 XGBoost 模型的样本个数。综上,调度算法复杂度为 $O(n \cdot \log n + n \cdot (M \cdot R + t))$。AI 算力调度算法可以在一定时间复杂度内,利用任务资源需求模型的推理预测能力,动态性对资源进行自适应伸缩调整,保障任务性能前提下提升 AI 算力资源利用率和任务部署效率。

3.4　调度系统设计

AI 算力自适应资源伸缩调度系统的设计框架(见图 1)由多用户、不同类型服务器、任务队列、任务历史运行日志数据库以及不同类型 AI 任务的任务资源预测模型和调度器组成。其中,用户是提交任务及资源需求的主体;服务器是任务执行的单元;任务队列负责维护由在线提交的用户任务所组成的动态任务列表;运行日志数据库负责收集用户任务的多维度资源使用情况、执行时间、任务类型等历史任务日志数据;任务模型负责预测多维度资源需求对性能的影响,指导调度过程中任务的自适应资源伸缩调整过程;调度器是本文方法的实现。

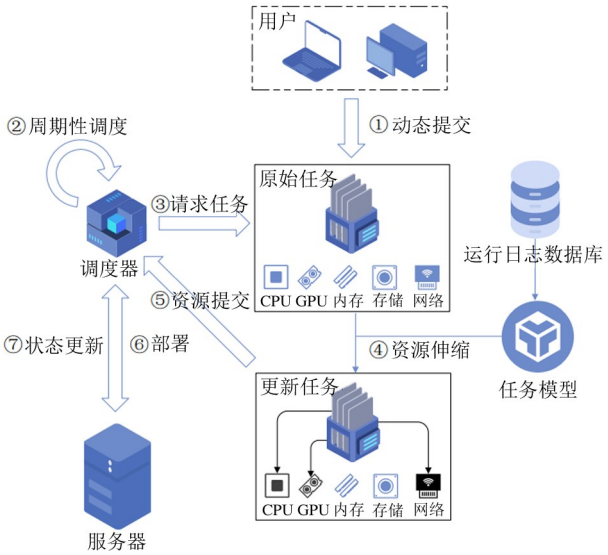


图 1　AI 算力自适应资源伸缩调度系统

在这个框架中,首先,调度器根据用户动态提交的任务,以及任务类型与多维度资源需求等具体任务属性,更新动态任务队列。接下来,当前调度的轮次中,在任务侧,调度器结合任务资源需求预测模型,对任务队列中当前用户任务申请资源下的预期性能进行预测。结合任务预期性能,对多维度资源需求进行自适应的伸缩调整,将更新之后的任务需求资源量返回到调度器。然后,在集群资源侧,调度器结合任务需求伸缩结果和集群的多维度资源使用状态,将任务部署到可以满足资源需求的服务器节点中,实现对集群作业的资源分配情况进行动态优化。最后,调度系统更新任务层次下的执行状态和集群层次下的资源状态,并准备进行下一轮次的调度。

综上,AI 算力自适应资源伸缩调度系统设计考虑了不同类型任务的资源需求特性和集群多维度资源的充分利用。资源伸缩调度方法可以解决任务申请的多个维度资源间的不匹配问题,提升集群层面任务的部署效率和多维度资源的利用率。

4　实验评估

4.1　实验配置与相关指标设计

AI 算力集群在线调度实验主要是验证基于任

务资源需求预测的 AI 算力自适应资源伸缩调度方法,对在线调度中任务的部署和执行过程的优化效果以及对 AI 算力多维度资源利用率的提升效果。本实验首先分析了 AI 算力自适应资源伸缩调度优化后的调度结果(实验 1),并进一步探究了不同类型任务占比、不同调度策略、2 种场景下(实验 2、3),AI 算力自适应资源伸缩调度的提升效果。

本实验是基于 AI 算力生产集群 Alibaba PAI[6] 中的 4 种典型 AI 任务的 10 万余条运行日志数据,仿真实现一个 AI 算力集群在线调度过程。其中,实验中集群配置与日志数据生成的集群配置相同,具体包含 32 个 GPU/2 TB 内存及 384 核 CPU。任务需求及集群配置考虑的多维度资源包括 GPU、CPU 及内存 3 种,即资源维度数目 $R=3$。

实验基于 Python 语言实现了大规模 AI 算力集群中的自适应资源伸缩调度过程。采用动态的任务到达形式,模拟在线生产集群中的用户任务随机到达的在线调度过程。根据设定任务负载的高低,以泊松分布动态加载在线任务,依次以事件流的方式逐一调度部署,完成 AI 算力集群调度的模拟实验。

为了充分验证本文提出方法的效果,需要对实验结果涉及的 AI 算力集群任务调度衡量指标进行定义。

第 1 是任务队列的完成度。它用于表示 AI 算力集群中一定时间内任务队列中完成部分的占比,反映动态任务队列的实际完成情况。任务队列完成度可以衡量单位时间内集群的任务部署执行效率。

$$CompleteRate = \frac{\#jobF}{\#jobF + \#jobQ} \tag{14}$$

其中,$jobF$ 是已完成的任务数量,$jobQ$ 是队列中尚未完成的任务数量。任务队列完成度表示已完成任务占全部任务的百分比。

第 2 是 AI 算力集群资源利用率。为了分别衡量 GPU、CPU 及内存 3 个维度资源的使用情况,实验结果衡量了 AI 算力集群的 GPU 利用率、CPU 利用率和内存利用率 3 个维度。

$$ResUtilization = \frac{resUsed}{resTotal} \tag{15}$$

其中,$resUsed$ 是某维度资源已使用的数量,$resTotal$ 是集群该维度的资源总量。

4.2 结果分析

实验 1 将基于任务资源需求预测的自适应资源伸缩方法前后的 AI 算力调度结果进行对比,验证本文方法对 AI 算力调度的提升效果。实验中对调度中任务优先级的设置采用先到先服务(first input first output, FIFO)策略,即调度系统将按照任务到达时间设置任务调度优先级。

图 2 是相同的运行时间(12 h)内任务量和任务完成度的对比。随着负载量的增加,任务完成度提升幅度增大。这得益于自适应资源伸缩分配调度方法对于资源超额申请情况的优化。本文优化方法效果具体体现在 2 个方面。一是资源侧,资源伸缩之前未能充分利用的资源,在优化之后能够用于部署其他任务。二是任务侧,优化前由于用户超额申请资源而在 AI 算力集群有限资源中不能部署的任务,得益于资源伸缩过程,在超额申请情况改善之后,也可以成功部署到集群中。从图 2 的结果来看,在每小时 70 个任务的负载量下,自适应资源伸缩前后的单位时间内任务完成度提升了 15.2%,同时成功部署的任务数量也提升 25.3%。

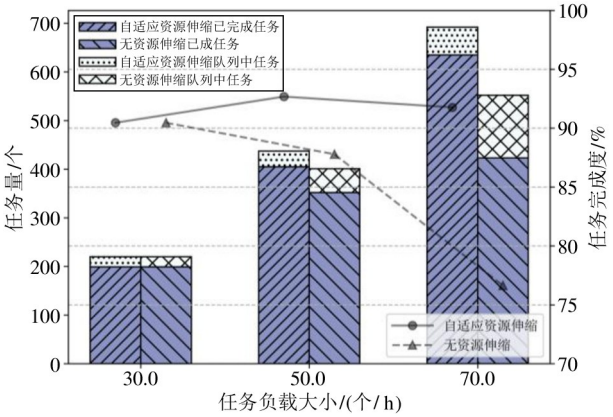
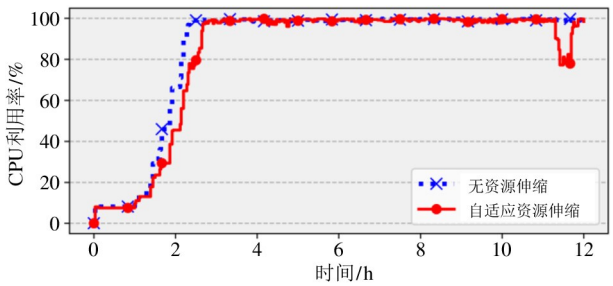


图 2 不同负载大小下自适应资源伸缩前后的任务完成度及任务部署量(实验 1)

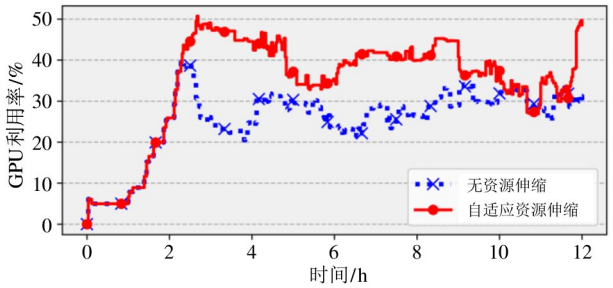
图 3 分别显示了在线调度期间 AI 算力集群中 CPU、GPU 和内存的资源利用率随着时间的变化趋势。结果表明,在每小时 70 个任务的负载大小下,相比于优化前的基线实验,GPU 资源利用率经过自适应资源伸缩优化后提升 7.2%,内存利用率提升 8.0%。而瓶颈资源 CPU 的利用率相比而言下降 2.0%。实现这种优化的主要原因是,自适应资源伸

缩调度方法使得在线任务的 CPU 资源需求申请量下降了 28.5%，可以让更多任务部署到有限的集群资源中，从而完成了 GPU 和内存资源平均利用率的提升。

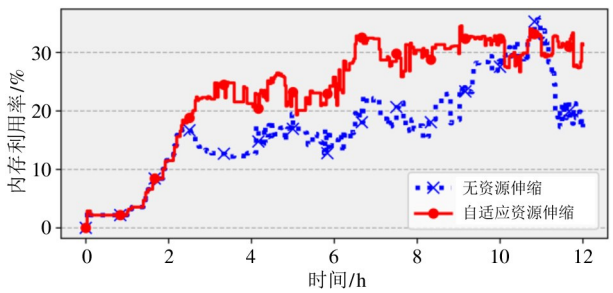
从图 3 中可以发现，优化前基线实验中（图 3 中虚线），调度开始后的 2 h 左右，由于 CPU 资源的使用到达瓶颈，不能进一步部署其他在线任务，所以 GPU 和内存的利用率被限制。而应用本文的自适应资源伸缩调度方法，对 CPU 资源申请量进行优化，可以在实际调度中部署更多任务，使得 GPU 和内存资源被浪费的情况得到缓解。因此，经过优化后，更多的任务能够部署到集群上，充分利用有限且昂贵的 AI 算力集群资源。



（a）资源伸缩前后集群 CPU 利用率



（b）资源伸缩前后集群 GPU 利用率



（c）资源伸缩前后集群内存利用率

图 3　自适应资源伸缩前后资源利用率对比（实验 1）

虽然图 3 中优化前后的资源利用率存在同一时间点交叉的情况，但是通过对任务调度过程的分析发现，出现这种情况是因为在集群调度过程中本文采用了动态在线任务调度。在资源伸缩优化前后，调度过程根据不同的任务资源需求，完成集群的资源调度分配，导致同一时间内集群内实际运行的任务负载并不相同。所以，集群资源利用率提升的效果，需要从调度时间区间内进行分析，而不能直接比对相同时间点的差异情况。

实验 1 的结果有效验证了本文基于任务资源需求预测的自适应资源伸缩调度方法能够在生产环境中对集群多维度资源利用情况实现提升，同时优化任务部署和运行效率。

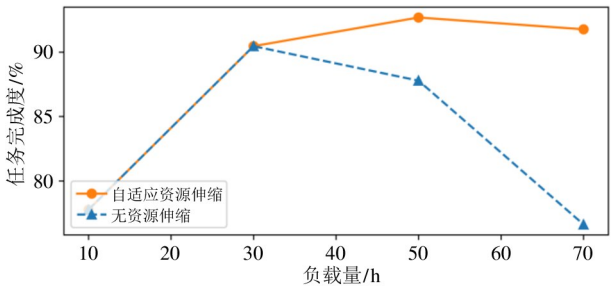
实验 2　针对不同的任务类型占比进行实验设计，探究混合负载中不同类型任务的占比对本文基于任务资源需求预测的自适应资源伸缩调度方法的影响。

该实验考虑 4 种 AI 算力集群典型的任务类型[6,14]，不同任务类型采用不同的任务资源需求模型。4 种任务类型分别是点击率预测的推荐任务[15]（click through rate，CTR）、基于图神经网络的图学习任务[16]、计算机视觉领域的图像分类任务[17]和自然语言处理（natural language processing，NLP）中神经网络机器翻译任务[18]（neural machine translation，NMT）。

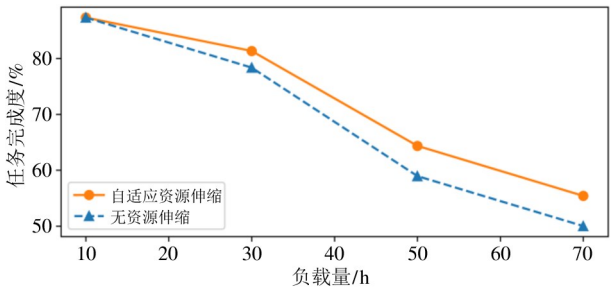
图 4 展示了 3 种不同的任务类型比例下，调度过程采用自适应资源伸缩调度方法前后的任务完成度，其中（ResNet∶NMT∶GraphLearn∶CTR）的比例为（5∶15∶15∶65）的混合负载，较为接近实际生产集群数据集上的任务占比情况。

通过对仿真采用的生产集群及任务数据集进行的特性分析，发现在该集群内，CPU 资源是瓶颈资源。根据对各类型负载的资源需求预测模型的特征重要性分析，可以发现 CTR 任务在调度过程中对 CPU 资源分配的平均伸缩程度最大，其次是 ResNet 任务。

实验 2（如图 4）结果显示，随着负载量的不断提升，（ResNet∶NMT∶GraphLearn∶CTR）为（5∶15∶15∶65）的任务比例下（即图 4（a））任务完成度提升最明显，而（40∶30∶20∶10）的任务比例下（即图 4（b））任务完成度的优化有所下降。这说明，通



(a) 任务比例为(5∶15∶15∶65)



(b) 任务比例为(40∶30∶20∶10)

图 4　不同任务比例下任务完成度的对比(实验 2)

过对更多的 CPU 超额申请的任务在调度中进行资源伸缩,集群整体利用性能相比而言得到更大的改善。

实验 2 的结果有效证明了,不同的任务比例影响了本文提出的自适应资源伸缩调度方法对集群调度的任务完成度的优化。本文提出的自适应资源伸缩方法,对 AI 集群任务性能瓶颈的资源(本实验中的 CPU)进行申请需求量的伸缩,可以优化相同时间内任务调度,实现任务完成度的提升。

实验 3　针对不同任务优先级的调度算法进行实验,探究不同任务调度策略下,本文基于任务资源需求预测的自适应资源伸缩调度方法对任务部署完成情况和资源利用情况的优化程度。

实验中选用 3 种不同任务优先级设置的调度策略。先到先服务调度策略下,调度系统根据任务到达时间设置任务调度优先级;最短时间剩余优先(shortest remaining time first, SRTF)调度策略下,调度系统根据任务剩余时间设置任务调度优先级;主资源公平(dominant resource fairness, DRF)调度策略下,调度系统则根据任务申请的多维度资源公平性设置任务优先级。实验 3 的结果显示了 3 种调度策略下,自适应资源伸缩优化前后任务完成度的提升情况(见图 5)以及多个维度资源利用率的提升情况(见图 6)。

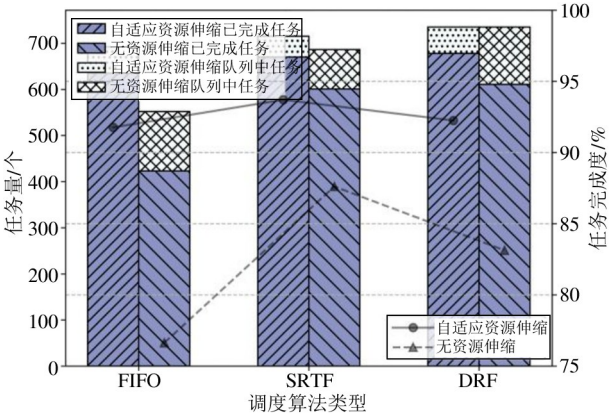


图 5　不同调度算法下任务完成度的对比(实验 3)

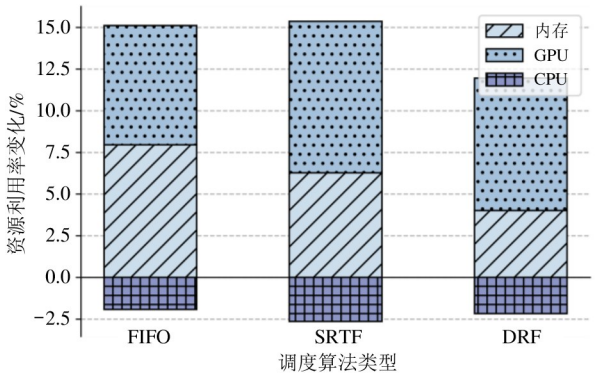


图 6　资源伸缩调度前后集群利用率的变化(实验 3)

实验 3 的结果有效证明了,在 3 种不同调度策略下,任务部署完成情况以及整体资源利用水平都在自适应资源伸缩调度方法的优化下得到提升,验证了本文提出的调度优化方法具有普遍性和可推广性。

综上,本文的自适应资源伸缩调度方法,在各种集群调度场景下,都能够有效提升单位时间内任务完成情况和集群多维度资源利用率水平。该方法可以在保证任务本身执行效率的前提下,结合任务资源性能特性的机器学习建模,对于用户提交的在线任务超额申请的资源进行伸缩调整,从而使得多维度资源之间互相匹配,起到能够充分利用集群有效资源的作用。

5　结　论

本文提出一种基于任务资源需求预测的 AI 算

力调度方法。相比于此前局限于 GPU 资源分配而设计的 AI 算力调度方法,本方法能够综合考虑多维度资源对任务性能的影响,完成自适应资源动态调度,解决用户提交任务过程中广泛存在的资源超额申请问题,有效提升 AI 算力调度中任务部署完成情况和多维度资源利用率。

本文的主要贡献包括以下 3 个方面。

(1)基于 XGBoost 方法分析 AI 任务运行日志,得到多维度资源对不同类型任务性能的影响,建立任务资源需求预测模型。

(2)提出一种基于任务资源需求预测的 AI 算力调度方法,利用任务模型完成自适应资源弹性伸缩调度,减少了调度过程的资源浪费。

(3)通过大规模 AI 算力调度的仿真实验,验证了本文方法能有效提升 AI 算力集群资源的利用率水平,并能优化任务分配部署。

本文 AI 算力调度方法还可以进一步进行优化。文中重点对 CPU、内存和 GPU 3 个维度的资源对任务的影响进行分析,但 AI 算力调度研究还可以从算力集群中网络带宽、磁盘读写等资源方面进行自适应调度。另外,本文将调度过程调整用户申请的资源作为重点,而未来的研究重点是通过分析任务特征直接给出满足性能指标的合理资源量,并考虑任务间的资源征用进行调度设计。

参考文献

[1] XIAO W, BHARDWAJ R, RAMJEE R, et al. Gandiva: introspective cluster scheduling for deep learning[C]// Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation. Carlsbad, USA: USENIX Association, 2018:595-610.

[2] GAO W, YE Z, SUN P, et al. Chronus: a novel deadline-aware scheduler for deep learning training jobs[C]// Proceedings of the ACM Symposium on Cloud Computing. New York, USA: Association for Computing Machinery, 2021:609-623.

[3] XIE L, ZHAI J, WU B, et al. Elan: towards generic and efficient elastic training for deep learning[C]// 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). Singapore: IEEE, 2020:78-88.

[4] PENG Y, BAO Y, CHEN Y, et al. Optimus: an efficient dynamic resource scheduler for deep learning clusters[C]// Proceedings of the 13th EuroSys Conference. New York, USA: Association for Computing Machinery, 2018:1-14.

[5] MOHAN J, PHANISHAYEE A, KULKARNI J, et al. Looking beyond GPUs for DNN scheduling on multi-tenant clusters[C]// USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA: USENIX Association, 2022:579-596.

[6] WENG Q, XIAO W, YU Y, et al. MLaaS in the wild: workload analysis and scheduling in large-scale heterogeneous GPU clusters[C]// The 19th USENIX Symposium on Networked Systems Design and Implementation. Carlsbad, USA:USENIX Association, 2022:945-960.

[7] HU Q, SUN P, YAN S, et al. Characterization and prediction of deep learning workloads in large-scale GPU data centers[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York, USA: Association for Computing Machinery, 2021:1-15.

[8] ZHAO M, AGARWAL N, BASANT A, et al. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product[C]// Proceedings of the 49th Annual International Symposium on Computer Architecture. Davis, USA: Association for Computing Machinery, 2022:1042-1057.

[9] MOHAN J, PHANISHAYEE A, RANIWALA A, et al. Analyzing and mitigating data stalls in DNN training[EB/OL]. (2020-07-14)[2023-04-10]. https://arxiv.org/pdf/2007.06775.pdf.

[10] JEON M, VENKATARAMAN S, PHANISHAYEE A, et al. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads[C]// Proceedings of the 2019 USENIX Conference on USENIX Annual Technical Conference. Renton, USA: USENIX Association, 2019:947-960.

[11] CHEN T, GUESTRIN C. XGBoost: a scalable tree boosting system[C]// Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, USA: Association for Computing Machinery, 2016:785-794.

[12] ALTMANN A, TOLOŞI L, SANDERO, et al. Permutation importance: a corrected feature importance measure[J]. Bioinformatics, 2010,26(10):1340-1347.

[13] MCILROY P. Optimistic sorting and information theoretic complexity[C]//Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms. Austin, USA: Society for Industrial and Applied Mathematics, 1993: 467-474.

[14] ACUN B, MURPHY M, WANG X, et al. Understanding training efficiency of deep learning recommendation models at scale[C]//2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). Seoul, Korea: IEEE, 2021:802-814.

[15] GUO H, TANG R, YE Y, et al. DeepFM: a factorization-machine based neural network for CTR prediction [EB/OL]. (2017-03-13)[2023-04-10]. https://arxiv.org/pdf/1703.04247.pdf.

[16] ZHU R, ZHAO K, YANG H, et al. AliGraph:a comprehensive graph neural network platform[J]. Proceedings of the VLDB Endowment, 2019,12(12):2094-2105.

[17] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA: IEEE, 2016:770-778.

[18] WU Y, SCHUSTER M, CHEN Z, et al. Google's neural machine translation system: bridging the gap between human and machine translation[EB/OL]. (2016-09-26)[2023-04-10]. https://arxiv.org/pdf/1609.08144.pdf.

Artificial intelligence computing power cluster scheduling based on task resource demand prediction

YANG Mingxuan[*,**], HONG Xuehai[*], TANG Hongwei[*,***]

([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

([***]University of Chinese Academy of Sciences, Nanjing 211135)

Abstract

A scheduling method based on task resource demand prediction is proposed to improve the job execution and resource utilization of artificial intelligence (AI) computing power cluster. Existing schedulers are designed by optimizing the graphics processing unit (GPU) resources allocation, which ignore the effect of multidimensional resources on AI task executing. In this work, the impact of multi-dimension resources on job execution and cluster resource utilization is considered. First, the multi-dimensional resource requirements of jobs are modeled through machine learning methods. Then, an adaptive resource scaling scheduling method is proposed, which reduce the over claim resource waste. It is found that compared with the basic strategy, this method makes more tasks allocated and executed in the same period. Evaluation results shows that the job deployment increases by 25.3%, the completion rate of deployed tasks increases by 15.2%. The GPU and memory utilization rates have been increased by 7.2% and 8.0% respectively, leading to an improvement in the overall utillization of computing resources.

Key words: resource scheduling, elastic resource allocation, artificial intelligence (AI), computing power