软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(3):579–597 [doi: 10.13328/j.cnki.jos.005158]

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

一种云环境中数据流的高效多目标调度方法*

沈 尧[1], 秦小麟[1,3], 鲍芝峰[2]

[1](南京航空航天大学 计算机科学与技术学院,江苏 南京 210016)

[2](School of Computer Science & Information Technology, RMIT University, Melbourne VIC 3000, Australia)

[3](软件新技术与产业化协同创新中心,江苏 南京 210016)

通讯作者: 秦小麟, E-mail: qinxcs@nuaa.edu.cn

摘 要: 在分布式系统中,云计算作为一种新的服务提供模式出现,其执行科学应用数据流时的优势和缺点得到越来越多的关注,其主要特点为拥有大量同质和并发的任务包,并构成了性能瓶颈的主要因素.在云数据流中调度大规模任务是已被证实的 NP 难问题.专注于解决优化云数据流中的调度过程,并受现实世界启发,从不同角度将优化目标分别划分为用户指标(完工时间和经济成本)和云系统指标(网络带宽、存储约束和系统公平度),并将该调度问题制定成为一个新的连续的合作博弈,设计出快速收敛的高效 Muliti-Objective Game(MOG)调度算法,在优化用户指标的同时,实现系统指标的约束,并保证云资源的效率和公平度.通过综合实验,证实该方法与其他相关算法相比,在算法复杂度 $O(l{\cdot}K{\cdot}M)$(明显改进数量级)、结果质量(一些情况下最佳)、系统级别公平性上具有明显的优越性.

关键词: 云计算;多目标优化;合作博弈;数据流处理;任务调度

中图法分类号: TP311

中文引用格式: 沈尧,秦小麟,鲍芝峰.一种云环境中数据流的高效多目标调度方法.软件学报,2017,28(3):579–597. http://www.jos.org.cn/1000-9825/5158.htm

英文引用格式: Shen Y, Qin XL, Bao ZF. Effective multi-objective scheduling strategy of dataflow in cloud. Ruan Jian Xue Bao/Journal of Software, 2017,28(3):579–597 (in Chinese). http://www.jos.org.cn/1000-9825/5158.htm

Effective Multi-Objective Scheduling Strategy of Dataflow in Cloud

SHEN Yao[1], QIN Xiao-Lin[1,3], BAO Zhi-Feng[2]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[2](School of Computer Science & Information Technology, RMIT University, Melbourne VIC 3000, Australia)

[3](Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210016, China)

Abstract: As a new emerging service provider, cloud computing, exhibiting advantages and disadvantages when executing the scientific data flows, is getting more and more attention. One of the main factors that constitute the performance bottleneck is there are many homogeneous and concurrent task packages in cloud. This paper focuses on optimizing the scheduling process in dataflow and transforming the optimization objectives into user metrics (makespan and economic cost) and indicators of cloud systems (network bandwidth, storage constraints and system fairness). An efficient multi-objective game algorithm (MOG) is proposed by formulating the optimization problem as a new cooperative game. The MOG method is able to optimize the user metrics while satisfying the constraint of the system metrics and ensuring the efficiency and fairness of the cloud resources. Comprehensive experiments demonstrate that

* 基金项目: 国家自然科学基金(61373015, 61300052); 中央高校基本科研业务费专项基金(NP2013307); 江苏省研究生培养创新工程(KYLX_0287)

Foundation item: National Natural Science Foundation of China (61373015, 61300052); Fundamental Research Funds for the Central Universities (NP2013307); Funding of Jiangsu Innovation Program for Graduate Education (KYLX_0287)

收稿时间: 2016-07-24; 修改时间: 2016-09-14; 采用时间: 2016-11-01; jos 在线出版时间: 2016-11-29

CNKI 网络优先出版: 2016-11-29 13:34:59, http://www.cnki.net/kcms/detail/11.2560.TP.20161129.1334.005.html

compared with other related algorithms, the proposed MOG method has obvious advantages in terms of algorithm complexity $O(l{\cdot}K{\cdot}M)$ (improvement of magnitude), result quality (optimum in some cases) and system level fairness.

Key words: cloud computing; multi objective optimization; cooperative game; dataflow processing; task scheduling

作为多种技术混合演进发展的云计算,如今已能够给用户提供更加可靠、一致、无处不在并廉价访问高端计算资源的能力,也能支持各种类型的科学应用数据流.为了规划大规模的基础设施,比如混合云,作为传统软件组件的松耦合协调模型的数据流[1]已经出现,并作为科学界最成功的编程范式之一.

研究人员试图解决的最具挑战性的 NP 完全问题之一[2]就是怎样将大规模的科学应用以任务数据流形式调度至分布式和异构资源云环境中,并针对某些目标函数进行优化,比如总执行时间或者在商业及市场化中的经济成本,考虑和实现某些执行约束比如通信成本和存储需求.从最终用户角度来看,最小化成本或者执行时间应当是被优先考虑的功能,然而从云系统的角度看,系统级别的效率和公平则是优先考虑的优化动机,这样有更多计算量的应用能够被分配到更多的资源.目前,仅有少数方案可以同时处理以上这两种角度,比如优化用户目标(完工时间、经济成本)的同时满足其他约束,并且能够给所有用户提供良好的效率和公平.另一方面,在数据流调度中,考虑合适的调度带宽和存储约束已经是亟待处理的问题,因为在大数据环境下,许多应用能够在相对短的时间产生大量的数据集,比如大型强子对撞机[3],预计每年能够产生约 7PB 的数据.

因为在云中会有许多不同类型的应用或者任务,其对可用资源的使用互为竞争.那么一些问题出现了,比如针对不同应用的有效资源配置,考虑它们各自的性能、成本、带宽、存储和其他潜在的约束条件,并从系统角度来考虑资源的最大公平使用.对于公共云,谷歌 2012 年推出了谷歌计算引擎云(GCE),其提供托管的 Linux 虚拟机.谷歌为开发者开放其基础设施并且成为了亚马逊弹性计算云(EC2)的一个强有力的竞争对手.表 1 显示了亚马逊和谷歌不同云服务的带宽,可以看到:谷歌云存储(GCS)比亚马逊简单存储服务(S3)速度慢,GCS 比 EC2 的传输速度慢.有趣的发现是,从 S3 到 GCS 的传输速度大约是 GCS 到 GCE 的 2.2 倍.因此,预计随着更多的云服务被使用,多目标云优化将会成为一项越来越复杂和有前景的事业.

Table 1 Bandwidth in public cloud

表 1 公共云中的带宽数据

从	至	传输带宽(Mbps)
S3	EC2	1 420
S3	GCS	510
GCS	EC2	380
GCS	GCE	230

本文针对云环境中的数据流应用提出了通信和存储约束的多目标调度计划,其特征为大量独立和同质的任务包,通过数据流依赖互联.

1) 多目标调度基于一个连续的合作博弈算法,最小化云应用的期望执行时间和经济成本;
2) 调度算法能够快速收敛,算法执行时间能比相关算法有明显提升;
3) 在传输产生数据时,该调度方法能够考虑它们的带宽和存储约束,同时最小化应用的完工时间和成本,并保证系统资源的最大公平使用.

本文提出的 MOG 调度算法的主要优势是使用竞争者和环境信息来快速收敛,并通过创建合理运动来最小化有关问题定制的需求等决定最有效的搜索方向.本文将 MOG 方法和 5 个相关启发式方法进行了性能比较,结果表明:针对大量任务包的云数据流应用,MOG 算法在复杂度(改进数量级)、结果质量(一些情况下最佳)、系统级别公平性上具有优越性.MOG 算法可能不适合于在任务包间具有复杂依赖关系的应用(比如水文 invmod、气象 MeteoAG 或者生物信息测试[4]等任务流),因为这些调度问题不能够被正确配置成为典型的可解博弈.

本文首先介绍相关工作.抽象数据流模型在第 2 节中描述.第 3 节具体阐述本文设计的 MOG 多目标调度算法.第 4 节中进行具体的实验验证算法的效率和有效性.最后,第 5 节总结并讨论未来工作.

1 相关工作

调度大型数据流应用是云计算中重要且困难的研究课题之一,其发展了很多不同的方法和算法.本节总结了相关领域的重要工作:启发式调度、博弈调度和多目标调度.

1.1 启发式调度

启发式方法通常属于那种没有详尽地搜索整个解空间的方法,该方法依赖于一个特定问题域的信息学习.每一个基于启发式的方法一般被设计用来仅适合一个特定类型的问题.所有方法都采用一组特定的应用规则来增加得到一个较好解的概率.算法通常能花费较少的时间得到一个较优的解.在任务调度问题领域,一般的基于启发式的方法可以被划分为 3 个类别:独立任务调度、列表调度以及基于集群和复制的调度.

最简单的就是独立任务调度,但是它仅仅适合有简单结构的任务流,比如管道类型.它不断地将每个独立就绪任务映射至能够提供其最早完成时间的资源,而不考虑其他任务和它们之间的依赖关系.列表调度算法特别适合有很多任务竞争有限数量资源的工作流.它给每个任务分配一个优先级,然后基于分配的优先级创建一个调度.目前有两种模式的列表调度:批处理模式[5]和依赖模式[6].批处理模式基于任务的执行时间设置其优先级,依赖模式基于关键路径的权重设置优先级.使用列表调度算法的好处是总的执行时间通常是最小化的.然而,该算法不能够处理资源优化问题,其中的任务流包含很多相等关键路径长度的并行任务.另一个缺点是需要反复地计算优先级,尤其在调度大规模任务流时会显著增加计算时间.

基于集群和复制的调度算法[7]聚焦于最小化完工时间.该算法试图通过将通信密集型任务集中映射至相同资源来减少依赖任务之间的通信时间.一些任务也被复制以避免资源之间的数据传输.

尽管上述调度方法都是有效的,但是它们仅仅满足单一的调度目标.例如,它们能够给出一个调度,提供最小的完工时间,但是执行的成本很高.为了修改算法处理多目标问题,计算调度计划所需的时间会大幅度地增加.此外,基于这些启发式方法设计针对多种数据流类型的调度框架是困难的.

1.2 博弈调度

就博弈算法而言,一些注重性能的分布式计算领域的研究人员聚焦在系统级的负载均衡上[8],或者资源分配[9],目标在于引入经济和博弈论方法至计算问题中.Siar[8]制订了调度问题作为合作博弈,这里,网格试图最小化任务的期望响应时间.Srinivasa 等人[9]通过考虑不合作机器因素,调查了单个机器自私行为的影响,Ghosh 等人[10]提出了一种策略,制订一种不完全信息,在两个变量上进行交流议价博弈:每资源单元价格和分配带宽百分比.与其工作相比,本文采用了更加实际的定价模型,类似于云资源提供商亚马逊弹性计算云所采用的计价模式.ICENI 项目[11]使用了博弈论算法解决调度问题,采用严格淘汰的支配策略.这里,末位最优解不断被丢弃.由于高时间复杂度,该算法的可行性是存疑的.除了博弈论算法,ICENI 采用了随机、*N*-随机最佳和模拟退火方法提供了调度方案.

1.3 多目标优化调度

许多真实世界的调度问题是多目标性质,比较常见的是,这些问题中有多个目标同时存在,比如最小化调度时间、最小化经济成本、满足存储限制和网络资源等.这些年来,有一些方法先后被用来处理这些多目标问题.

传统上,最普遍的方法是解析并结合多重目标至单一的总和目标函数,并给不同的目标分配权重[12].然而,在很多涉及多目标调度的真实场景下,最好是展示各种折衷方案给用户,以使得最合适的调度被选择.此外,聚合目标函数在目标中通常是非线性的,并且获得的解决方案将取决于指定权重的相关值.由于加权法需要用户喜好的先验信息,因此其本质是主观的.那么,如何寻找一组折衷方案来表现良好的近似帕累托最优前沿,则成为了探索帕累托优化方法的新课题.比如,Fard 等人[12]提出了基于帕累托支配约束向量的方法,并应用至 4 个目标中,作为目标函数的部分(完工时间、成本、能耗和可靠性),本文的方法与之相比有着更低的时间复杂度,并考虑进两个不同的目标(完工时间和成本)和两种约束(带宽和存储资源).

许多元启发式方法原本被用来解决单目标优化问题,现在被推广到生成多目标优化变型.其中,多目标进化

算法已经得到了特别的关注,因为一些研究人员认为这些方法特别适合于处理多目标优化问题.此外,一些多目标的元启发式方法比如模拟退火[13]、遗传算法[14]、禁忌搜索和粒子群优化(PSO)[15]被提出来解决调度问题.我们在此方向也做了一些努力,试着解决更小更简易的问题版本[14].Buyya 等人[16]介绍了新的遗传算法以解决在大规模异构环境中现有遗传算法无法直接应用的情况,由于它们的高时间复杂度,遗传算法在大规模应用中不实际.Arabnejad 等人[17]以经济成本作为数据来计算调度中的部分目标函数,但是其没有考虑存储约束,也不能全局解决性能和成本优化问题.Pandey 等人[15]提出了 PSO 启发式方法来调度应用至云资源中,并考虑进了计算成本和传输成本.蚁群优化(ACO)[18]和人工蜂群(ABC)[19]方法基于蚂蚁和蜜蜂觅食的行为,在代理的子群内做出集体决策,其在搜索过程中可能陷入局部最优解而不一定得到全局最优解.进化算法和以上算法的主要缺点是收敛速度慢而导致的高计算成本[20,21].本文提出的 MOG 方法与此相反,由于全局资源信息的有效利用,而有着更快的收敛速率.

本文介绍的方法成功应用了博弈论概念,能调度多个大规模应用至云环境中.本文建模了一种现实可行的云系统模型,其特征为大量均匀独立的任务通过简单数据流依赖互联,并将调度问题制订为一种多个应用管理者之间的连续合作的博弈,设计快速收敛的高效 MOG 多目标调度算法,优化完工时间和经济成本,在满足通信和存储约束的同时,保证了系统资源的公平度.

2 数据流模型

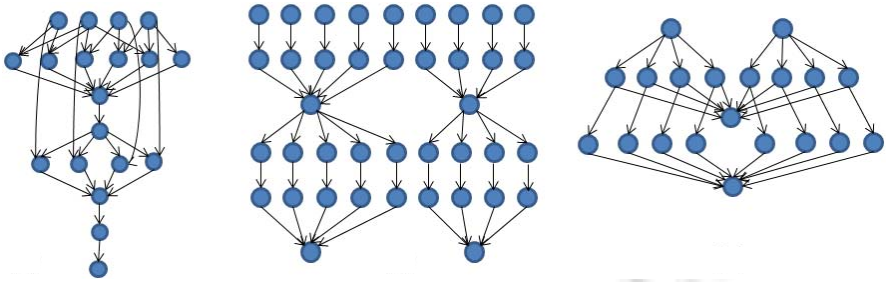
本节描述文中使用的抽象数据流模型,设计原则来自于现实世界的应用和真实的云计算平台.本文研究的大规模数据流,以大量(数千到数百万)同质并行且独立的任务为特征.这些任务决定性能,并通过控制和数据流之间的依赖而互连.

定义 1. 采用 $w=(BS,DD)$定义一种数据流应用,并建模为 DAG.这里, $BS=\bigcup_{k=1}^{K}T_k$ 是 K 个异构任务包的集合, $DD=(T_s<T_d|\{T_s,T_d\}\subset BS)$是数据流的依赖集.这里称 T_s 为 T_d 的前继,并且写做 $T_s=pred(T_d)$.任务包 T_k 定义为 $T_k=\bigcup_{j=1}^{K_k}t_{kj},k\in[1,2,...,K]$,且作为一组同质的并行原子序列任务,它们有着相同任务类型并且可以同时执行,这里的 K_k 是任务包的基数.

大规模应用中的数据流类型一般有 4 种:Montage(如图 1(a)所示)[22],Ligo(如图 1(b)所示)[23],Cybershake(如图 1(c)所示)[24]和 Lattice(如图 2 所示).前 3 个是在实际应用中所使用的真实数据流的抽象:Montage 被 NASA 用来产生自定义天空马赛克,Ligo 被激光干涉引力波观测站用来分析星系的双星系统,Cybershake 被南加州地震中心用来描述地震.

通过观察负载的时间特性,可以识别出并行阶段的高低.并行性降低的情况有两种:(1) 当操作从大量的其他操作收集或者分发数据时;(2) 有大量增加的数据将要被转移时.Montage 任务流出现一个并行量最高的阶段,同时它也有一些瓶颈操作.Ligo 具有并行性更加均匀分布的阶段以及适中的瓶颈操作数量.Cybershake 有两个高并行阶段并仅有 4 个瓶颈操作,同时,它传输了大量数据.而 Lattice 被设计为可以自定义并行度和瓶颈数.Lattice 是一个纯粹的数据流合成系列并被设计成为云环境中典型的 Mapreduce 模型的数据流(如图 2 所示).特定的 Lattice 流有一个确定的高(H)和分支因子(B)并被定义为(H-B)Lattice.例如,图 2(b)中表示(5-2)Lattice.

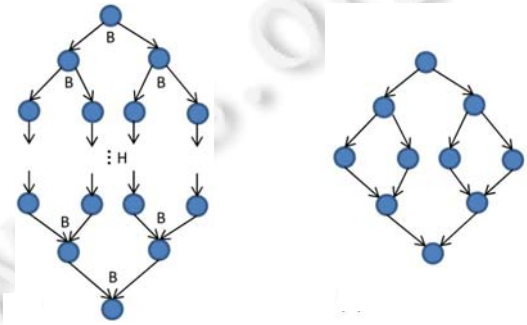
在这些数据流类型中,性能瓶颈的来源都是同质的大规模任务包,连续任务在大规模的应用中相对琐碎,并可以按需调度至最快或者最便宜的可用处理器上.因此,这给优化提供了方向,也带来了空间.



(a) Montage (b) Ligo (c) Cybershake 数据流

Fig.1

图 1



(a) Lattice 数据流 (b) 高 H=5、分支 B=2 的 lattice 示例数据流

Fig.2

图 2

3 多目标博弈调度方法

本文的目标是设计一种新算法来调度一组定义 1 中表示的应用以及在第 2 节中定义的环境模型中调度大量的任务,表 2 描述了本文使用的关键符号列表.本文算法的目标是优化两个目标函数:累计完工时间和经济成本,同时还能够满足带宽和存储约束.

Table 2 List of key notations

表 2 关键符号列表

符号	描述	符号	描述
K	任务包数量	pw_{ki}	性能权重
M	云站点数量	cw_{ki}	成本权重
$S_i(i\in[1,m])$	云站点	bw_{ki}	带宽权重
m_i	S_i 上的处理器数量	sw_{ki}	存储权重
δ_k	S_i 中任务总数	bl_i	S_i 的带宽限制
β_{ki}	S_i 中任务处理速率	br_k	任务包的带宽需求
p_{ki}	S_i 上任务包的预期执行时间	Γ,σ,ν,μ	拉格朗日乘子
pc_{ki}	云任务的计算时间	$S(n)$	博弈的第 n 个阶段
po_{ki}	云任务的通信时间	ω	优化阈值
d_{ki}	S_i 中任务包 K 的数据大小	sl_i	S_i 的存储限制
b_{ki}	S_i 中分配给任务包 K 的带宽	η_i	S_i 的价格
λ_i	S_i 中输入数据带宽	Δ	任务分布矩阵
θ_{ki}	S_i 中分配给任务包的处理器数量	Γ	带宽分布矩阵
sr_k	任务包的存储需求	Π	资源分布矩阵

3.1 问题定义

定义 2. 假定有一系列的 N 个工作流应用(如定义 1 中建模并且忽略他们的到达时间),其由 K 个子任务包组成,运行在 M 个站点的云环境中.在应用 $A_i,i\in[1,n]$中,完工时间由其中任务包的最大完成时间决定.因此,多目标调度问题的目标是寻找一种解决方案,能够分配所有的任务至站点,使得所有应用的完工时间和经济成本的 $F(x)$最小,并且满足带宽和存储需求,公式表达如下:

$$\text{Minimize}F(x)=(f(x),c(x));\text{并且有 } h_i(x)\leqslant\lambda_{x,i},i\in[1,M],g_i(x)\leqslant sl_i,i\in[1,M],x\in S \tag{1}$$

这里,x 作为一个解,S 作为一套可行的解决方法,$F(x)$是解 x 在多目标空间中的像,$f(x)$是性能目标,$c(x)$是经济成本目标,$g(x)$是存储函数,$h(x)$是带宽函数,$\lambda_{x,i}$ 是云站点 S_i 的输入数据带宽,sl_i 是云站点 S_i 的存储限制.

预期计算时间矩阵(ETC)最初由 Braun 等人[25]提出并应用于分布式环境中的调度问题,随后,大量研究采用 ETC 矩阵这一概念并应用在云环境中的任务、资源等调度问题中[26,27],其表示每个云站点 $S_i,i\in[1,M]$的每个任务包 $k\in[1,K]$中子任务包的预期执行时间 p_{ki}.通信开销是异构云环境中的关键问题,在很多情况下是最为重要的因素.对于大规模且有大量任务的云应用,通信和计算可以同时运行和叠加,造成了运行时间可以由通信或者计算决定.因此,基于计算时间 pc_{ki} 和通信时间 po_{ki},可以定义预期执行时间 p_{ki}.它们之间的关系如下所示:

$$p_{ki}=\begin{cases}pc_{ki}, & pc_{ki}\geqslant po_{ki}\\ pc_{ki}+(po_{ki}-pc_{ki})=po_{ki}, & pc_{ki}<po_{ki}\end{cases} \tag{2}$$

其中,通信时间 po_{ki} 表示如下:

$$po_{ki}=\frac{d_{ki}}{b_{ki}}.$$

d_{ki} 表示任务的数据大小,b_{ki} 表示云站点 S_i 中分配给任务包 K 的带宽.那么,云站点的输入数据带宽 $S_i(\lambda_{x,i})$可以表示为云站点 S_i 中的所有 b_{ki} 之和,公式表示如下:

$$\lambda_{x,i}=\sum_{k=1}^{K}\theta_{ki}\cdot b_{ki}=\sum_{k=1}^{K}\frac{\theta_{ki}\cdot d_{ki}}{po_{ki}} \tag{3}$$

公式(3)中的 θ_{ki} 表示云站点 S_i 中分配给任务包 T_k 的处理器数量.θ_{ki} 是实数,任务对少量的资源进行请求,使得我们更加精细地调整资源的分配.在调度的最后,对 θ_{ki} 取整.基于以上的分析,能够发现针对数据密集型应用,预期执行时间 p_{ki} 由计算时间和通信时间决定.通信时间是由分配到每个任务包的带宽决定.

3.2 调度系统架构

多目标调度问题可以归结为应用管理器之间的合作博弈,理论上可以产生最优解,尽管由于问题的高复杂度而并不容易实现.因此,本文将多目标调度问题作为连续的合作博弈来进一步制定和解决,其中需要 3 个重要参数的正确定义:博弈参与者、策略和规范的收益.

科学数据流系统往往采取简单的计算模型,特别在数据流模型中,任务流模块执行顺序是由数据流来决定的.有向无环图的任务流构成了云应用,其中,每个节点代表任务,边表示应用节点之间的依赖关系.在基于 DAG 的任务流中,图为 $G\{V,E\}$,点 $V=\{t_1,\ldots,t_n\}$表示任务流的每个任务,图中的边表示节点之间的任务依赖关系.

图 3 描述了数据流类型可以作为数据流解析器的输入.然后,调度映射器将 DAG 文件格式化为 XML 和其他元数据信息,比如文件大小等.映射完成以后,任务流调度器创建一个任务列表,并且将任务分配至执行站点.一个任务即为用户将要执行的一个程序.

数据流引擎基于任务之间的依赖来管理任务,以确保一项任务只有在其父任务成功完成的情况下才被释放.数据流引擎只会释放自由的任务至调度器.在调度器中,配置博弈多目标调度功能.该博弈多目标调度将会应用博弈规则,并以一种优化的方式来调度任务至虚拟机.在本节提出的算法中,将调度所有的任务至虚拟机,并以公式(1)中描述的约束来完成最终的调度.

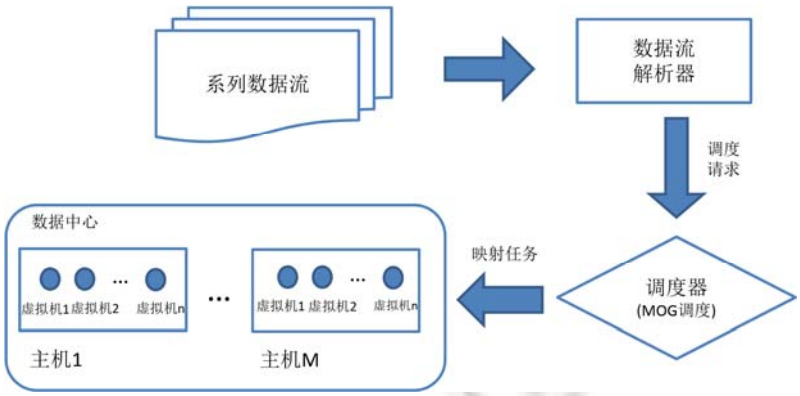


Fig.3 System architecture of multi-objective game scheduling

图 3 多目标博弈调度系统架构

3.3 博弈调度方案

考虑有 K 个参与者的合作博弈,每个应用管理器(作为参与者)试图在一定时间内最小化一个任务包 T_k 的执行时间 t_k,基于每个云站点 S_i 的任务总数 δ_k 和任务处理速率 β_{ki}.为清楚起见,假定每个应用管理器处理一个任务包的执行.每个管理器的目标是最小化执行时间和任务包的经济成本,同时满足存储和带宽的约束,目标函数的公式表示为

$$f_k(\Delta)=\frac{\delta_k}{\beta_k}=\frac{\delta_k}{\sum_{i=1}^{M}\frac{\theta_{ki}}{p_{ki}}} \tag{4}$$

$$c_k(\Delta)=\sum_{i=1}^{M}p_{ki}\cdot\delta_{ki}\cdot\eta_i \tag{5}$$

$$h_i(\Delta,\Gamma)=\sum_{k=1}^{K}\frac{\theta_{ki}\cdot d_{ki}}{p_{ki}}\leqslant\lambda_{x,i} \tag{6}$$

$$g(\Delta)=\sum_{k=1}^{K}sr_k\cdot\theta_{ki}\leqslant sl_i \tag{7}$$

其中,Δ 是任务分布矩阵 $(\delta_{ki})_{K\times M}$,$sr_k$ 是任务包 T_k 的存储需求,sl_i 是云站点 S_i 的存储限制,η_i 是云站点的价格,Γ 是带宽分布矩阵 $(b_{ki})_{K\times M}$.在公式(5)中,假定用户仅仅支付有用的计算,并且价格都独立于使用的处理器数量.Δ 和 Γ 都将策略表示为合作博弈中收益的体现.

合作博弈考虑的是参与群体能够协调自身行为的情况,而这作为算法中最重要的机制,使得博弈具有可转移的效用.换句话说,一个有着增加效用的参与者,具有补偿其他有着减少效用参与者的能力.当设计具有可转移效用的博弈时,主要的考虑是开发针对形式化共享资源公平方法的解决方案.例如,以下定义的项 θ_{ki} 表示云站点 S_i 上任务包 k 的资源分配情况,其中体现了共享资源的公平性.公式(8)将其定义为 S_i 上任务包 T_k 加权汇总执行时间与 S_i 上所有任务包总执行时间的比值和 S_i 上处理器数量 m_i 之间的乘积:

$$\theta_{ki}=m_i\cdot\frac{\delta_{ki}\cdot p_{ki}\cdot w_{ki}}{\sum_{k=1}^{K}\delta_{ki}\cdot p_{ki}\cdot w_{ki}} \tag{8}$$

其中,w_{ki} 表示云站点 S_i 的任务包 T_k 中对性能(pw_{ki})、成本(cw_{ki})、带宽(bw_{ki})和存储(sw_{ki})优化的不同定义的权重,公式表示如下:

$$pw_{ki}=\frac{\dfrac{\min\limits_{i\in[1,M]}\{p_{ki}\}}{p_{ki}}}{\sum_{i=1}^{M}\dfrac{\min\limits_{i\in[1,M]}\{p_{ki}\}}{p_{ki}}}=\frac{\dfrac{1}{p_{ki}}}{\sum_{i=1}^{M}\dfrac{1}{p_{ki}}} \tag{9}$$

$$cw_{ki}=\begin{cases}\dfrac{\dfrac{1}{\eta_i p_{ki}}}{\sum_{i=1}^{M}\dfrac{1}{\eta_i p_{ki}}}, & \eta_i\neq 0\\ \dfrac{\dfrac{1}{p_{ki}}}{\sum_{i=1}^{M}\dfrac{1}{\eta_i p_{ki}}}\to 0, & \eta_i\to 0\end{cases} \tag{10}$$

$$bw_i=\frac{\dfrac{p_{ki}}{d_{ki}}}{\sum_{k=1}^{K}\dfrac{p_{ki}}{d_k}} \tag{11}$$

$$sw_i=\frac{\dfrac{1}{sr_k}}{\sum_{k=1}^{K}\dfrac{1}{sr_k}} \tag{12}$$

必须仔细考虑如何写权重函数的问题,以便能够实现更好的性能和成本,并且针对给定问题可以真正地改善调度.如果权重函数选择不佳或者定义不准确,算法有可能找不到解决问题的方案.例如,采用性能权重 pw_{ki} 在性能方面提高分配的公平性.因为一个站点对不同的任务可能因为不同的原因而产生不同的适应性,比如数据局部性、内存大小、CPU 频率或者 I/O 速率等.举例来说,如果站点 S_i 对任务包 T_k 的执行时间 t_{ki} 比在其他站点上低很多,那么就将该任务在该站点上设置较高优先级,并且分配更多的资源.当目标偏离用户期望或者违反一些约束的时候,MOG 算法动态选择不同的权重.再者,必须保证每个任务包的权重和为 1,那么一个站点的子博弈胜利者必定是其他站点的失败者,否则不能够保证全部合适的分配.在介绍顺序博弈的概念时,本文将解释不同权重的利用.

基于资源的分布以及云站点 S_i 的处理速率和任务包的总处理速率的比值,将任务分布定义为任务包 T_k 中的任务数量与站点 S_i 的任务 T_k 处理速率和任务包总处理率比值之间的乘积.

$$\delta_{ki}=\delta_k\cdot\frac{\dfrac{\theta_{ki}}{p_{ki}}}{\sum_{i=1}^{M}\dfrac{\theta_{ki}}{p_{ki}}} \tag{13}$$

定义 3. 以下构成了多目标优化博弈的定义.

(1) K 个任务包的管理者作为博弈参与者;

(2) 一套策略Δ和Γ由如下的一组约束定义:① $\delta_{ki}\geqslant 0$;② $b_{ki}\geqslant 0$;③ $\theta_{ki}\leqslant\delta_{ki}$;④ $b_{ki}\leqslant\lambda_{x,i}$;⑤ $sr_{ki}\leqslant sl_{x,i}$;⑥ $\delta_{ki}=0$, if $\theta_{ki}<1$;⑦ $\sum_{k=1}^{K}\theta_{ki}=m_i$;⑧ $\sum_{k=1}^{K}b_{ki}=\lambda_{x,i}$;⑨ $\sum_{k=1}^{K}sr_{ki}=i$;⑩ $\sum_{k=1}^{K}\delta_{ki}=\delta_k$;

(3) 对于每个博弈参与者 $k\in[1,K]$,目标函数为 $f_k(\Delta)$和 $c_k(\Delta)$.目标是同时最小化所有的 f_k 和 c_k,并且满足 g_k 和 h_k;

(4) 对于每个博弈参与者 $k\in[1,K]$,目标函数的初始值为 $f_k(\Delta^0,\Gamma^0)$,其中, $\Delta^0=(\delta_{ki})^0_{K\times M}$ 是充满了初始任务分布的 $K\times M$ 矩阵, $\Gamma^0=(b_{ki})^0_{K\times M}$ 是含有初始化带宽分布的矩阵.Γ^0 计算基于Δ^0,(Δ^*,Γ^*)表示博弈的最优解.一个最初的分布基于Δ^0和Γ^0 产生,起初,其并不需要是一个能够实现所有约束的可行方案.

对于包含任务包的大规模应用来说,整体完工时间几乎等于总执行时间除以处理器数量.因此,本文的协同

优化博弈的性能目标可以近似为最小化总的完工时间.

$$\arg\min_{\Delta}(\sum_{k=1}^{K} f_k(\Delta)) \tag{14}$$

公式(14)受到定义 3 里的 10 种约束.对于寻找受到一些约束的函数极值问题,典型的方法是通过定义拉格朗日函数.

$$\begin{aligned}\Theta(\delta,\xi,\theta,\mu,b,\sigma)=&\sum_{k=1}^{K}(f_k+c_k+g_k+h_k)+\xi\cdot(\sum_{k=1}^{K}\theta_{ki}-m_i)+\sigma\cdot(\sum_{k=1}^{K}b_{ki}-\lambda_{x,i})+\\&\nu\cdot(\sum_{k=1}^{K}\delta_{ki}-\delta_k)+\sum_{k=1}^{K}\mu_i^k\cdot\delta_{ki}+...\end{aligned} \tag{15}$$

这里的Γ,σ,ν,μ表示的是拉格朗日乘子.通常来说,这种问题的准确最优解是难以获得的,因为该问题有着高复杂度和 $3\cdot K\cdot M$ 个变量.解决方案取决于不同站点上相同任务包中的任务分布、相同站点中不同任务包中的任务分布以及处理器和带宽的分配.换句话说,一个变量的改变,影响了所有其他变量的值.为了克服这种困难,本文将该问题进一步制定为一个连续博弈来近似解决.这里,博弈参与者选择一种策略,其遵循预定义的顺序,并观察在他们前面的参与者的动作.虽然最优的解决方案不会直接从公式(15)中获取,但可以在一组博弈阶段中获得中间的解决方案,基于以下的不平等序列.

$$\sum_{k=1}^{K}F_k^{S(1)}(\Delta^0,\Gamma^0)\geqslant\sum_{k=1}^{K}F_k^{S(2)}(\Delta^{S(1)},\Gamma^{S(1)})\geqslant...\geqslant\sum_{k=1}^{K}F_k^{S(n)}(\Delta^{S(n-1)},\Gamma^{S(n-1)})\geqslant\sum_{k=1}^{K}F_k(\Delta^*,\Gamma^*) \tag{16}$$

这里,S 表示顺序博弈的阶段,$S(n)$表示博弈的第 n 个阶段.在每个阶段,博弈参与者(任务包的管理者)基于上一个阶段的资源分布来提供一系列的策略(任务分布),并且使用公式(8)来产生新的分布.

博弈中的第 1 步是初始化任务分配$\Delta^{S(0)}$和带宽分配$\Gamma^{S(0)}$,每个任务包基于每个站点的处理速度分配到一定量的处理器,在最初阶段 $S(0)$,每个任务包假定所有的处理器和带宽都是可用的.

改写公式(13),得到:

$$\delta_{ki}=\delta_k\cdot\frac{\frac{m_i}{p_{ki}}}{\sum_{i=1}^{M}\frac{m_i}{p_{ki}}} \tag{17}$$

对于云站点 S_i 中分配给任务包 K 的带宽 b_{ki},有:

$$b_{ki}=\lambda_{x,i}\cdot\frac{d_{ki}}{\sum_{i=1}^{K}d_{ki}} \tag{18}$$

从公式(17)、公式(18)中,得到最初的任务分布$\Delta^{S(0)}$和带宽分配$\Gamma^{S(0)}$.

第 n 阶段资源分配为$\Pi^{S(n)}$,资源分配矩阵为$\Pi=(\theta_{ki})_{K\times M}$,基于上一步的任务分配$\Delta^{S(n-1)}$计算得到的.相应地,计算第 n 步的任务分配$\Delta^{S(n)}$是通过第 n 步的资源分配$\Pi^{S(n)}$计算得来的,即:

$$\Pi^{S(n)}=\Pi(\Delta^{S(n-1)});\Delta^{S(n)}=\Delta(\Pi^{S(n)}) \tag{19}$$

公式(19)描述的过程可以通过使用公式(8)和公式(13)计算得到.

3.4 多目标博弈算法MOG

本节提出多目标博弈 MOG 算法,实现了上一节中形式化的博弈调度方法.我们之前的工作[14]提出了针对性能以及经济成本的双目标优化算法.在如何拓展双目标调度算法至多 QoS 约束下的多目标情况的思路中,需要解决如何选择一个或多个目标来引导搜索,最终面向一整套的潜在解决方案.最常用的方法是将多目标问题转化成单目标,可以通过给每个标准分配权重来形成一个目标的线性组合,并且汇总成为一个聚集函数.该技术的主要缺点是没有应用程序、基础设施等相关的先验信息来确定权重,这有可能导致现实非线性问题的不可解.另一种经典的方法是在以下 3 种方法切换:所有目标、每次单个目标和选定作为下一次博弈的权重 w_{ki}.本文这里采用了交替并结合目标的混合型方法.MOG 算法首先同时优化性能和通信标准,并附加经济成本的约束.在确定了性能范围后,MOG 开始优化成本和通信标准.这里的困难是怎样建立标准应该被优化的顺序,因为这能够影响搜索是否成功.

MOG 算法首先需要构建期望执行时间矩阵(ETC 矩阵).ETC 矩阵由云任务和云站点组成,可以由公式(2)

计算得到.图 4 为 ETC 矩阵的一个示例,其表示了 4 种任务$\{t_0,t_1,t_2,t_3\}$在 4 个云站点$\{s_1,s_2,s_3,s_4\}$的期望执行时间.为了清晰起见,在示例中限制每个站点的规模仅包含一个处理器.

$$\begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ t_3 \end{array}\begin{array}{c} \begin{array}{cccc} s_1 & s_2 & s_3 & s_3 \end{array} \\ \begin{pmatrix} 18 & 12 & 19 & 14 \\ 20 & 15 & 78 & 12 \\ 32 & 19 & 12 & 15 \\ 23 & 9 & 61 & 8 \end{pmatrix} \end{array}$$

Fig 4 An example of ETC matrix

图 4 ETC 矩阵示例

博弈调度算法的伪代码形式共包含 3 个阶段,见算法 1.

阶段 1:在获得有关任务和资源的信息以后(例如图 4 中的 ETC 矩阵),生成任务的最初分配Δ^0和权重矩阵(见算法 1 中的第 3 行~第 11 行).为了便于说明和方便比较,在这个简单的例子中,只考虑任务完成时间的优化.该阶段中,用户可以设置性能约束或者通过简单设置应用的权重为 0 来过滤掉不需要的站点,即可以阻止将任务映射至那些站点.为了保证所有的约束都满足,可以在第 3 阶段再次进行验证.

阶段 2:重复循环的每次迭代(算法 1 中的第 12 行~第 15 行)都是一次博弈阶段,每阶段由 M 次子博弈组成(即,每个站点一次).在每个子博弈中,所有的任务包争夺资源分配,并且那些有较大权重的赢得每个站点的子博弈并能够在下一步中获得更多的资源.然而,这些任务由于权重的定义而并不能在每个站点中都保持博弈胜利(即,每个任务包的权重和为 1).因此,在一个站点的子博弈胜利者必定是其他站点的失败者.重复该过程,直到最优.该算法的进一步处理方式取决于算法 1 中第 15 行的性能和成本的评估结果,这里的ω用来控制博弈次数和优化的程度.在算法 1 中的第 13 行,采用不同的权重定义来产生新的资源分配矩阵$\Pi^{S(1)}$.算法 2 描述了多目标优化算法,即,根据多种目标而进行的权重调整,这是博弈参与者在下一博弈阶段的策略的改变.基本来说,博弈参与者的策略需要随时做出改变,尤其在一些约束没有满足的时候,或者完工时间、经济成本偏离了用户的期望.例如,当存储约束在某个站点不能被满足的时候,也就是存储资源的平衡条件首先被有最大存储需求 sr 的用户打破,那么有最大 sr 的参与者将首先被要求改变策略.因此,已知所有参与者的全局信息,一次权重改变能够通过降序排列用户的 sr 来简单计算,然后使用存储权重来代替旧的权重.基于$\Pi^{S(1)}$,采用公式(19)(算法 1 第 14 行)来产生第 1 个任务分布$\Delta^{S(1)}$.然后重复迭代,直到达到预期优化的上限.

阶段 3:最后,最早完成的任务包被删除.为了利用释放的资源,重复以上两步来重新计算剩余任务包的分布,直到所有的应用完成.

算法 1. 博弈调度算法.

输入:一组应用$\mathbb{R}$;任务包数量 K;站点数量 M;站点 $S_i(i\in[1,m])$的处理器数量 m_i;ETC 矩阵$(p_{ki})_{K\times M}$;任务包 $k(k\in[1,K])$的任务数量δ_k;优化阈值ω;站点 $S_i(i\in[1,m])$的带宽限制 bl_i;任务包 $T_k(k\in[1,K])$的带宽需求 br_k;

输出:任务分布矩阵$\Delta^{S(n)}$;资源分配矩阵$\Pi^{S(n)}$.

1. $Gamer\leftarrow\varnothing$ //初始化参与博弈组
2. Repeat //阶段 1:初始化任务分布Δ^0和任务包权重,并选择性应用约束条件
3. for all $\varpi\in\mathbb{R}$ do
4. for all $T_k\in\varpi\wedge T_k$ not yet $scheduled\wedge(pred(T_k)=\varnothing\vee(T_j$ is completed,$\forall T_j\in pred(T_k)))$ do
5. $Gamer\leftarrow Gamer\cup T_k$
6. for all $i\in[1;M]$ do
7. Calculate pw_{ki}, cw_{ki}, sw_{ki},通过公式(9)、公式(10)、公式(12)
8. Calculate δ_{ki},通过公式(13),产生Δ^0
9. end for
10. end for

11. end for
12. Repeat //阶段 2:搜索最终的任务分布和资源分配
13. $\Pi^{S(n)}\leftarrow$*Multiobjective-Schedule*(Π,Δ,m,b,λ,p) //见算法 2
14. 计算$\Delta^{S(n)}=(\delta_{ki})_{|Gamer|\times M}$,由公式(19)
15. until $\sum_{k=1}^{|Gamer|}(f_k(\Delta^{S(n-1)})-f_k(\Delta^{S(n)}))\leqslant\omega\wedge\sum_{k=1}^{K}(c_k(\Delta^{S(n-1)})-c_k(\Delta^{S(n)}))\leqslant\omega$
16. wait for 任务包完成
17. //阶段 3:移出完成的任务包,释放资源,通过重复阶段 1 和阶段 2 来开始一轮新的博弈.
18. *Gamer*$\leftarrow$*Gamer*\T, $\forall T\in$*Gamer*$\wedge T$ completed
19. until $\forall\varpi\in\mathbb{R}$ 完成

算法 2. 多目标优化算法.

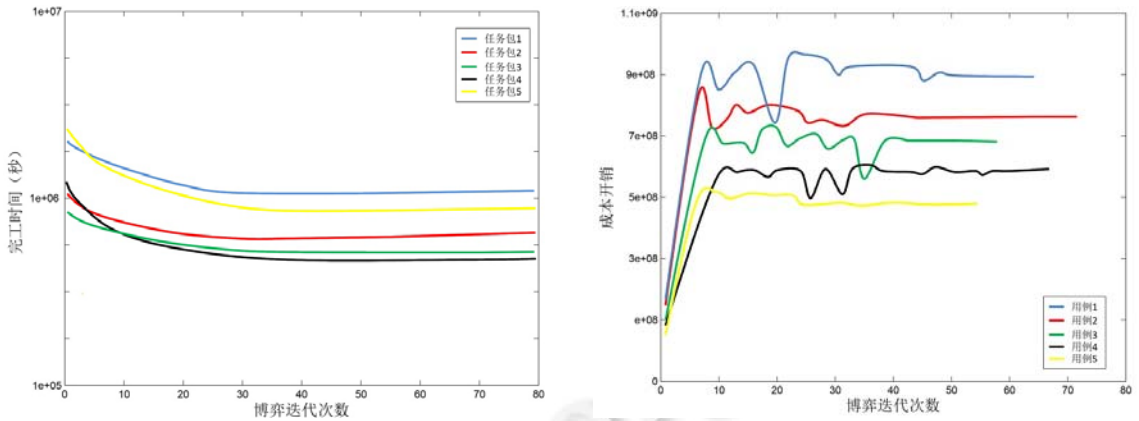
输入:一组应用$\mathcal{R}$;任务包数量 K;站点数量 M;站点 $S_i(i\in[1,m])$的处理器数量 m_i;ETC 矩阵$(p_{ki})_{K\times M}$;任务包$k(k\in[1,K])$的任务数量δ_k;优化阈值ω;站点 $S_i(i\in[1,m])$的存储限制 sl_i;任务包 $T_k(k\in[1,K])$的存储需求 sr_k;云站点上任务包 k 的资源分配情况θ_{ki};

输出:资源分配矩阵$\Pi^{S(n)}$.

1. for all $T_k\in\varpi$ do
2. Caculate all bw_k,通过应用公式(11)计算
3. end for
4. for all $T_k\in\varpi$ do
5. for all $S_i\in[1,M]$ do
6. if $\sum_{k=1}^{K}b_{ki}\cdot\theta_{ki}>\lambda_{x,i}\vee\sum_{k=1}^{K}sr_{ki}\cdot\theta_{ki}>sl_i$ then
7. 重新计算 θ'_{ki},通过公式(8)和公式(11)或者公式(12)计算的权重
8. end if
9. 重新计算 θ''_{ki},通过公式(8)和公式(9)或者公式(10)计算的权重
10. end for
11. end for
12. 评估不同权重下不同资源分布Π的收益
13. return 最佳收益的资源分布Π

3.5 算法分析

图 5(a)和 5(b)描绘了 MOG 算法在总的执行时间和成本随着博弈步骤的收敛性.该实验中,随机产生一些测试用例,分配了 $10^2\times10^4$ 个任务至 $10^2\times10^2$ 个处理器中.在 30~40 次博弈内,大多数例子完成了优化过程.最初,任务基于每个站点的处理速率和性价比分布.因此,开始的成本总是很低并且完成时间较长.最初的成本和性能对应于初始化阶段.快速收敛的原因是:某种程度上来说,每个任务包从自身利益出发去争夺最有效的资源.在竞争最有效资源完成以后,算法机制会让博弈参与者将工作负载移至较弱的站点.换句话说,这一机制使得博弈参与者对每个资源的私有已知偏好逐渐汇总成为一个社会性的选择.该机制确保了当参与者个体行为自私的情况下能产生一个全局最优解.最终,所有的任务包能够达到平衡状态,即,不能获得进一步的改善.



(a) 完工时间收敛性 (b) 成本开销收敛性

Fig.5

图 5

4 实 验

本节首先将 MOG 算法与基于经典的两类调度启发式方法修改的 5 种贪婪算法进行比较.运行所有测试机器的配置为 Intel i7-2640M 2.8GHz 处理器和 4GB 内存,并基于 Cloudsim 平台[28]和数据流生成器 Pegasus[1].

4.1 相关算法分析

- 复杂度为 $O(n\cdot N)$的算法[29].

在任务列表中迭代一次,并按如下方式调度:贪婪最小执行时间(GMET),分配每个任务至提供最快执行速度的机器,同等条件下,则分配至花费成本最小的机器.贪婪最小完成时间(GMCT)分配每个任务至提供最小完成时间或者成本的机器.

- 复杂度为 $O(n\cdot N^2)$的算法[30].

根据以下标准,在选择一个调度之前遍历所有任务:贪婪 min-min(Gmin-min)算法首先映射小的任务,并且映射到执行快的机器上;贪婪 max-min(Gmax-min)算法首先调度大任务,任务到资源的映射是选择最早完成时间最大的任务映射到所对应的机器上;贪婪 suffrage 算法计算每个任务的最小完成时间和次小完成时间的差值,选取所有任务中差值最大的任务和计算资源.

MOG 方法的时间复杂度为 $O(l\cdot K\cdot M)$,对应于算法的 4 个嵌套循环,即,算法 1 第 1 阶段中的第 2 行~第 4 行、第 6 行.这里的 l 表示连续博弈的阶段数量,K 为全部输入任务流$\varpi\in\mathbb{R}$中的任务包总数,M 是云站点数量(每个阶段中的运行工作为常量).算法第 2 阶段并不会影响复杂度,因为每个阶段中的工作是常量,并且仅仅线性取决于站点的数量和少数的博弈参与者.更重要的是复杂度独立于任务总数,与其他相关方法相比,这是巨大的优势.

图 6 为实验结果,体现了 MOG 算法在不同的计算环境下运行在相同机器上的性能表现.不同计算环境的设置表现为不同的云站点数量,例如 10×10 表示为有 10 个云站点,每个站点 1 个处理器;不同的任务包数量,例如 10×10 表示为有 10 个任务包,每个任务包有 10 个子任务.可以观察到:MOG 具有良好的可拓展性,因为博弈阶段数量增长并不会和处理器与任务的数量增加一样快.即使对于 10^6 个任务、10^4 个处理器的计算环境,MOG 算法也仅仅需要 593 个博弈阶段和 0.36s 就完成了调度优化.与 Gmin-min 之类的算法需要高达 1h 的优化调度时间形成了鲜明的对比.基于这些分析,在以后的实验中,对 MOG 方法的最大迭代数量设置为 1 000,认为就可以得到足够的收敛.

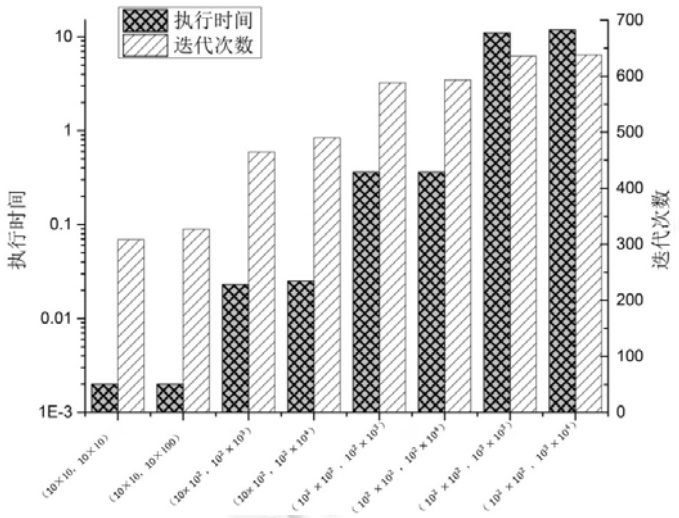


Fig.6 Performance of MOG under different computing environment

图 6 不同计算环境下 MOG 算法性能表现

通过图 7 得到了更多的实证结果,即,MOG 方法和经典贪婪算法在调度 10^5 个任务和 10^3 个处理器情况下的结果比较.MOG 的执行时间少于 0.3s,显著快于其他算法.例外是 GMET 的执行时间少于 1s,其时间复杂度为 $O(m+N)$,这里的 m 是处理器数量,N 是任务数量,N 远大于 M.然而,GMET 有着严重的问题,因为它序列化了大量的任务至最快的站点上.GMCT 的时间复杂度为 $O(m\cdot N)$,但是它的结果比 MOG 相差太多.Gmin-min,Gmax-min 和 Gsuffrage 算法的时间复杂度都是 $O(m\cdot N^2)$,平均执行时间约 200s~300s.文献[31]提到了异构的最早完成时间(HEFT)算法,其最初是为工作流设计,在确定重点的阶段中基于升秩排序任务,因此可以通过首先调度小任务而降级成 Gmin-min 算法.GA 算法[16,21]或者 A 星算法[32]随着任务和处理器数量的增加而可拓展性差,并且有可能需要较长时间来生成类似完工时间或成本等的解决方案.针对该原因,考虑 GA 方法并不适合这里的问题,也不去进一步研究它(即使它可以比 Gmin-min 的完工时间降低 5~10 个百分点[21]).其他算法都和本文以上讨论的类似,或者并不适用于这里的问题,比如,工作队列[32]适合均匀集群.

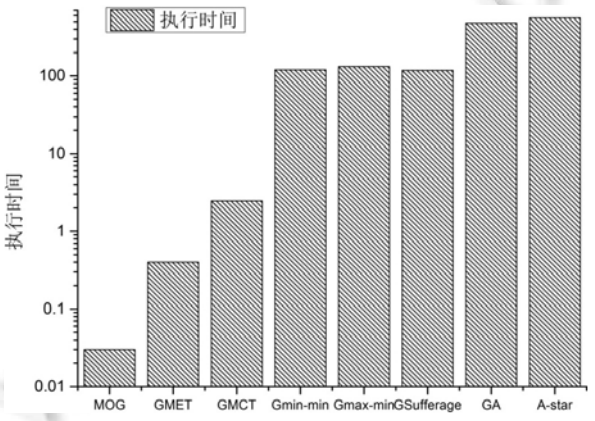


Fig.7 Comparison of different algorithms under 10^5 tasks and 10^3 processors

图 7 不同算法在 10^5 个任务和 10^3 个处理器环境下的性能比较

4.2 实验验证

为了实验的完整性,将 MOG 方法和相关算法进行对比,并根据 ETC 矩阵来模拟评估 MOG 的性能.不同的

ETC 矩阵是根据资源和异构任务的不同程度产生的,其在特定范围内的均匀分布中选出.表 3 给出了模拟环境的细节.

Table 3 Simulation environment settings

表 3 模拟环境设置

场景	处理器	集群	任务	任务包	异构任务	异构资源
HtHr	888	10	153 456	10	[1,1000]	[1,100]
HtLr	960	10	151 234	10	[1,1000]	[1,10]
LtHr	880	10	147 658	10	[1,10]	[1,100]
LtLr	1 000	10	159 867	10	[1,10]	[1,10]

在范围[1,100]的高异构资源引起了站点之间在任务执行时间和成本上的显著差异,而范围在[1,1000]高异构任务表示了不同任务的预期执行时间变化很大.假设每个任务包中的任务数量基于一个均匀分布的范围[10000,20000]内随机产生,每个站点的处理器数量在[64,128]范围变化.任务包在工作流中每对任务包之间有10%的非循环依赖概率.这里不选择模拟大量应用和站点的原因基于两点:(1) Gmin-min 一类的算法复杂度需要几小时来运行,这使得整个模拟不便;(2) 扩大仿真规模只会增加本文提出的 MOG 算法与相关启发式算法相比中的优势.在该模拟环境中,我们设计了 4 个场景中对算法进行了评估:高任务高资源异构(HtHr);高任务异构低资源异构(HtLr);低任务异构,高资源异构(LtHr);低资源低任务异构(LtLr).为了量化用户或者应用是否收到系统资源的公平的份额,本文使用了 Jain 的公平度指数[33].

$$Fairness=\frac{\left(\sum_{k=1}^{K}T_k\right)^2}{K\cdot\sum_{k=1}^{K}T_k^2} \tag{20}$$

这里,K 表示应用的数量,T_i 是应用$\mathcal{R}_i$的执行时间.公平度范围为[0,1],其中,0 表示最坏的情况,1 表示最好的情况(如图 8 所示).

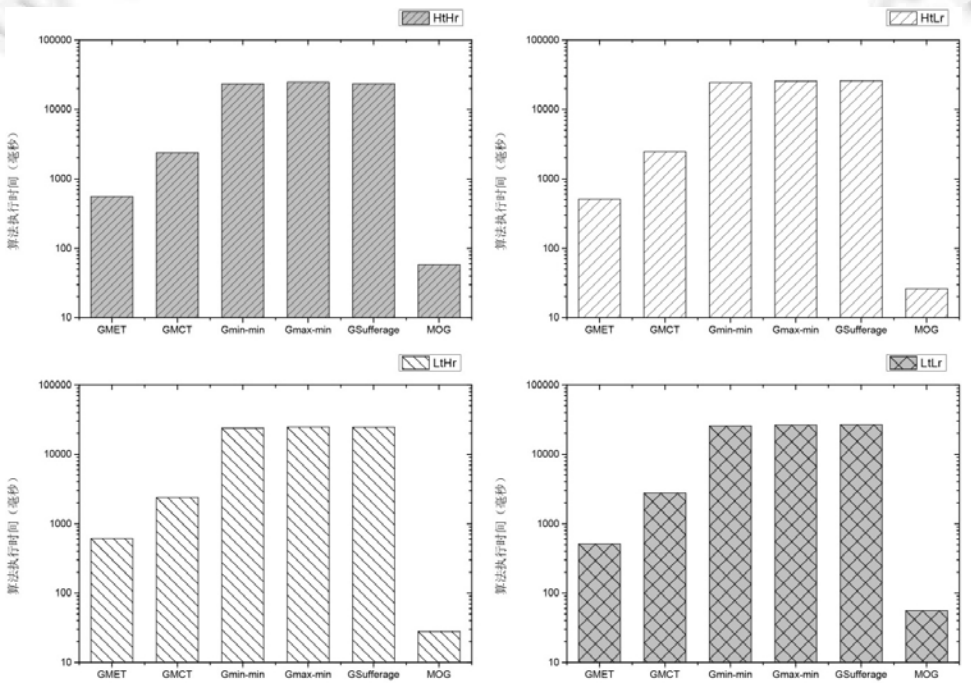


Fig.8 Execution time of algorithms under four scenarios (10^5 tasks and 10^3 processors)

图 8 4 种设定场景中的算法执行时间(10^5 个任务和 10^3 个处理器)

表 3 介绍了 4 种场景下的输入,图 9、图 10 为考虑到完工时间和公平性目标的相应模拟结果.GMET 总是给出最坏的结果,因为它映射所有的任务至最快的机器.Gmax-min 给出了较差的结果,因为它只适合当少量任务比其余任务大得多的情况下,而因为模拟环境采用均匀分布生成,所以不会遇到这一情况.此外,Gmax-min 对小任务缺乏公平,因此,它比大部分算法表现较差.GMCT 对于高度异构机器来说表现非常好,是因为它有更高的可能性来选择最快的机器,尤其针对大的任务.对于低机器异构性表现欠佳,是因为它仅考虑了完成时间但是忽略了任务的执行时间.在高异构机器中,GSuffrage 表现和 GMCT 类似,低异构机器情况下,其优于 GMCT 约 5~10 个百分点,因为它通过考虑任务执行时间,做出更多的智能决策.Gmin-min 在每种情况下给出了第二佳的结果,归因于其均匀的任务执行时间分布,但是因为首先处理最小的任务而导致缺乏公平性.

在图 11 中,展示了各种算法所产生的 skyline 结果.Gmin-min 在开始仅仅处理最小的任务而忽视了更大的任务.然而,最小的任务并非有着最佳的性能/价格比,因此,Gmin-min 不能够很好地执行.GSufferage 表现非常类似 Gmin-min,因为类似的原因.与 Gmin-min 相反,Gmax-min 给出更好的结果,因为其在开始的时候调度大的任务,这使得大任务在站点中更多获得最佳的性价比.G-MCT 给出了第二佳的结果,因为其无意识地选择了具有平均大小的任务,在统计学上有着更大的可能性,具有最佳性能价格比.MOG 在大多数情况下给出了最佳的调度,然而对于任务间具有复杂依赖关系的工作流来说,它可能无法提供最佳结果,因为这里的调度问题不能够被归结为一种典型的可解博弈.

MOG 在这 4 种计算场景下都提供了最佳的性能,因为其采取了最佳的全局决策.它在 LtHr 情况下优于 Gmin-min 算法 10 个百分点,其余 3 种情况皆优于 5 个百分点.可以看到:在确保公平的同时,效率也能得到改善.进一步观察在图 10 中,平均来说,MOG 总是获得较好的公平度 0.95 以上.

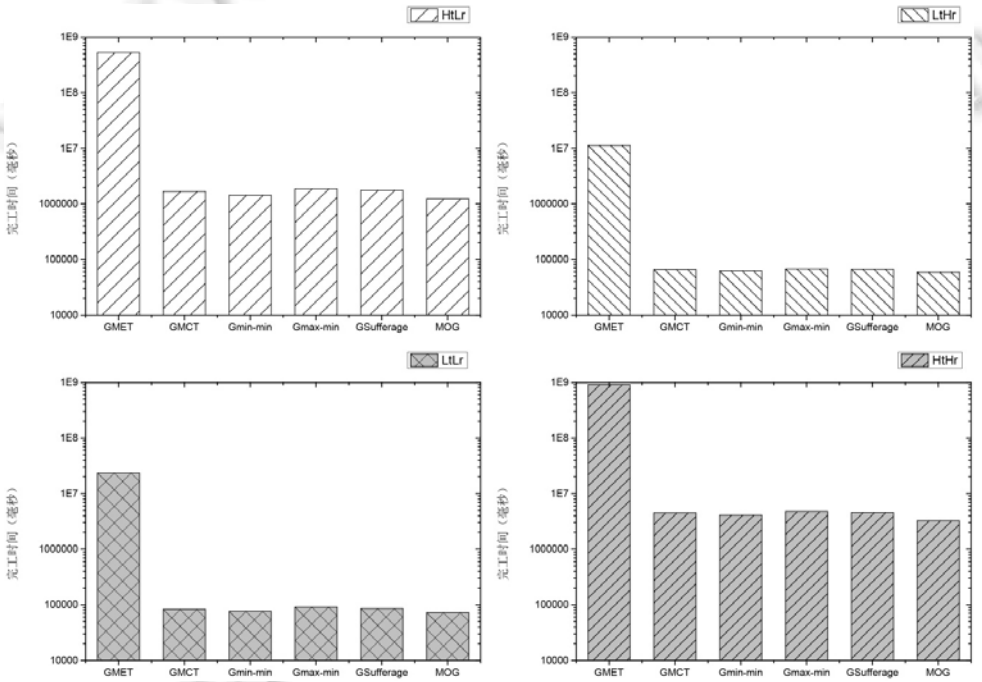


Fig.9 Makespan of tasks under four scenarios

图 9 4 种设定场景中的任务完工时间

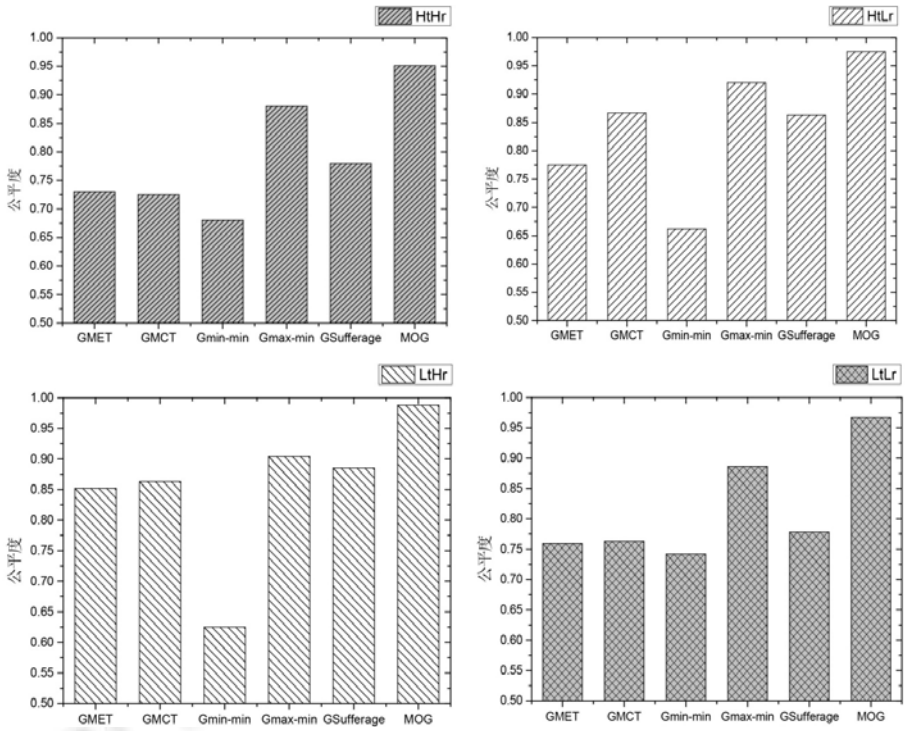


Fig.10 Fairness of algorithms under four scenarios

图 10 4 种设定场景中的算法调度公平度

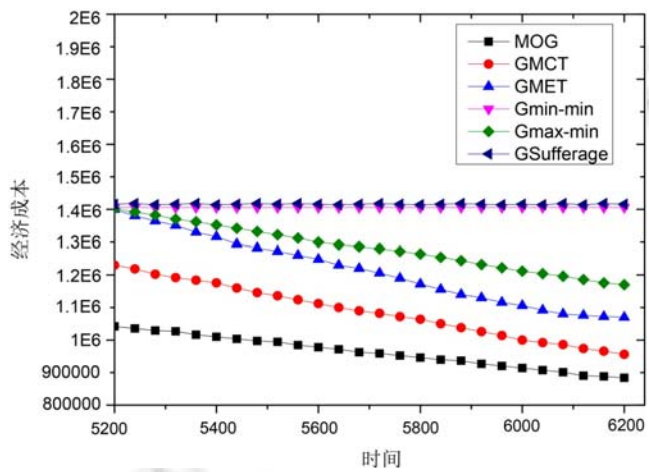


Fig.11 Multi-Objective scheduling of algorithms in consistent environment

图 11 相关算法的多目标调度比较

4.2.1 通信存储约束

表 4 给出了模拟计算环境,这里的真实值是在特定范围内,从均匀分布中随机产生.

Table 4 Environment setting of the communication and storage constraints

表 4 通信存储约束下的环境设置

处理器	集群	任务	任务包	异构任务	异构资源	带宽需求	存储需求
1 020	10	150 000	10	[1,1000]	[1,100]	[1,1000]	[1,100]

正如预期的那样,如图 12 所示,从最好到最坏的相对顺序是:MOG,Gmin-min,GSufferage,GMCT,Gmax-min,GMET.

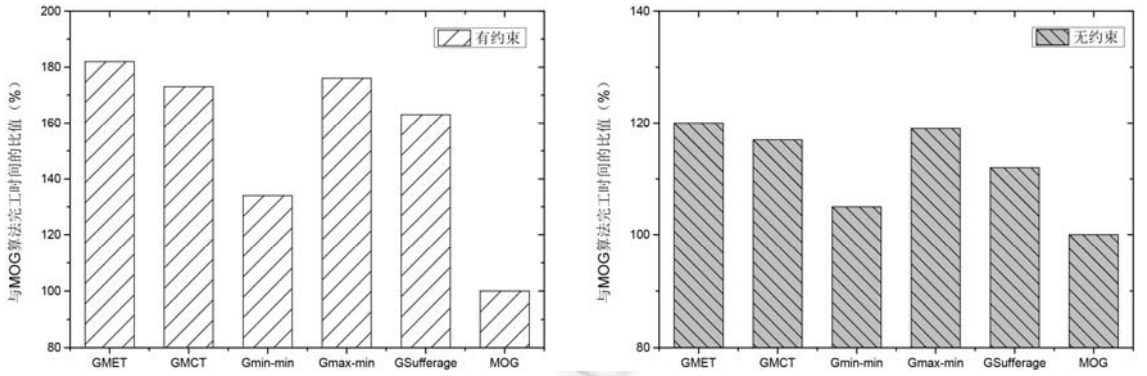


Fig.12 Ratio of related algorithms and MOG on makespan with and without constraint

图 12 在有无约束条件下,相关算法和 MOG 算法完工时间的比值

当站点中没有带宽或者存储约束时,MOG 表现比 Gmin-min 好 5%~10%,比其他算法低至少 26%的成本.在有约束的情况下,MOG 提高了多数据流性能至少 33%,并且降低了至少 70%的成本.如图 13 所示.MOG 提供了最佳的性能,因为其在性能和带宽同时优化方面做出了最佳的全局决策,而其余的启发式算法仅仅找到一种两个目标之间简单的妥协,带宽要求也不能够满足,导致了计算能力的潜在浪费.在成本方面,图 13 说明了所有的算法需要约两倍于 MOG 的成本.

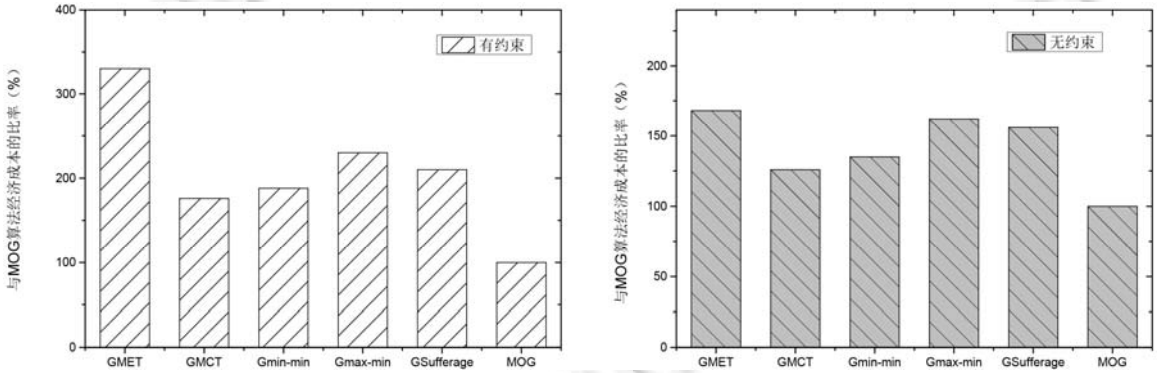


Fig.13 Ratio of related algorithms and MOG on cost with and without constraint

图 13 在有无约束条件下,相关算法和 MOG 算法经济成本的比值

5 结 论

随着混合云中的大规模应用得到越来越多的关注,数据流管理服务如何高效并且有效地调度和动态地引导应用执行显得越来越重要.本文分析了一类以大量同质任务为特征应用的主要瓶颈,提出了一种能够满足通信和存储约束的多目标调度 MOG 解决方案,其中包括 4 种重要指标:完工时间、成本、存储资源和网络带宽.基于模拟的实验结果,表明了本文提出的 MOG 方法在完工时间、经济成本和公平性等方面提供了更好的解决方案,与其他的 Gmin-min,Gmax-min 或者 GSufferage 贪婪方法相比,算法执行时间更少.这些都表明本文成功地克服了其他启发式方法的收敛速度慢和随机结构的缺点.

在未来的研究中,着重研究本文提出的算法如何适应其他指标,如内存、安全性、资源可用性等.以后将从两方面来扩展我们的方法:(1) 寻找新的合适的权重,使得任务包能够更快或者更慢地完成;(2) 对任务包指定

截止期限,并根据分配的期限将其划分为子任务,在每个子任务中应用我们的方法.最后,本文设计的用任务处理率来表达任务分配的方法(见公式(13)),可以很容易地扩展到处理动态任务的在线调度问题.

References:

[1] Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Good J. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 2005,13(3):219−237. [doi: 10.1155/2005/128026]

[2] Alvarez-Valdes R, Fuertes A, Tamarit JM, Gimenez G, Ramos R. A heuristic to schedule flexible job-shop in a glass factory. European Journal of Operational Research, 2005,165(2):525−534. [doi: 10.1016/j.ejor.2004.04.020]

[3] Landsberg G. Black Holes at the Large Hadron Collider. Switzerland: Springer Int'l Publishing, 2015. 267−292. [doi: 10.1007/978-3-319-10852-0_9]

[4] Castro AG, Thoraval S, Garcia LJ, Ragan MA. Workflows in bioinformatics: Meta-Analysis and prototype implementation of a workflow generator. BMC Bioinformatics, 2005,6(1):87. [doi: 10.1186/1471-2105-6-87]

[5] Maheswaran M, Ali S, Siegal HJ, Hensgen D, Freund RF. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Juan S, ed. Proc. of the Heterogeneous Computing Workshop. Washington: IEEE Press, 1999. 30−44. [doi: 10.1109/HCW.1999.765094]

[6] Wu W, Bouteiller A, Bosilca G, Faverge, Dongarra J. Hierarchical dag scheduling for hybrid distributed systems. In: Proc. of the IEEE Symp. on Parallel and Distributed Processing (IPDPS 2015). IEEE Press, 2015. 156−165. [doi: 10.1109/IPDPS.2015.56]

[7] Chaudhuri P, Agrawal A. An algorithm for task scheduling in heterogeneous distributed systems using task duplication. Int'l Journal of Grid & High Performance Computing, 2011,3(1):89−97. [doi: 10.4018/jghpc.2011010105]

[8] Siar H, Kiani K, Chronopoulos AT. An effective game theoretic static load balancing applied to distributed computing. Cluster Computing, 2015,18(4):1609−1623. [doi: 10.1007/s10586-015-0486-0]

[9] Srinivasa KG, Srinidhi S, Kumar KS, Srinidhi S. Game theoretic resource allocation in cloud computing. In: Proc. of the 5th Int'l Conf on Applications of Digital Information and Web Technologies (ICADIWT 2014). IEEE Press, 2014. 36−42. [doi: 10.1109/ICADIWT.2014.6814667]

[10] Ghosh P, Basu K, Das SK. A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. IEEE Trans. on Parallel and Distributed Systems, 2007,18(3):289−306. [doi: 10.1109/TPDS.2007.34]

[11] Young L, McGough S, Newhouse S, Darlington J. Scheduling architecture and algorithms within the ICENI grid middleware. In: Proc. of the UK e-science all hands meeting. EPSRC, 2003. http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/005.pdf

[12] Fard HM, Prodan R, Barrionuevo JJD, Fahringer T. A multi-objective approach for workflow scheduling in heterogeneous environments. In: Proc. of the 12th Int'l Symp. on Cluster, Cloud and Grid Computing. IEEE Computer Society, 2012. 300−309. [doi: 10.1109/CCGrid.2012.114]

[13] Meng FC, Chu DH, Li KQ, Zhou XQ. Solving SaaS components optimization placement problem with hybird genetic and simulated annealing algorithm. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):916−932 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4965.html [doi: 10.13328/j.cnki.jos.004965]

[14] Shen Y, Bao ZF, Qin XL, Wang S. Adaptive task scheduling strategy in cloud: When energy consumption meets performance guarantee. In: Proc. of the World Wide Web-Internet&Web Information Systems. 2016. 1−19. [doi: 10.1007/s11280-016-0382-4]

[15] Pandey S, Wu L, Guru SM, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proc. of the 24th Int'l Conf. on Advanced Information Networking and Applications. IEEE, 2010. 400−407. [doi: 10.1109/AINA.2010.31]

[16] Yu J, Buyya R. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In: Proc. of the Workshop on Workflows in Support of Large-Scale Science. IEEE, 2006. 1−10. [doi: 10.1109/WORKS.2006.5282330]

[17] Arabnejad H, Barbosa JG. A budget constrained scheduling algorithm for workflow applications. Journal of Grid Computing, 2014, 12(4):665−679. [doi: 10.1007/s10723-014-9294-7]

[18] Zuo L, Shu L, Dong S, Zhu C. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. IEEE Access, 2015,3:2687−2699. [doi: 10.1109/ACCESS.2015.2508940]

[19] Kefayat M, Ara AL, Niaki SAN. A hybrid of ant colony optimization and artificial bee colony algorithm for probabilistic optimal placement and sizing of distributed energy resources. Energy Conversion and Management, 2015,92(3):149−161. [doi: 10.1016/j.enconman.2014.12.037]

[20] Kansal NJ, Chana I. Artificial bee colony based energy-aware resource utilization technique for cloud computing. Concurrency and Computation: Practice and Experience, 2015,27(5):1207−1225. [doi: 10.1002/cpe.3295]

[21] Huang SC, Jiau MK, Lin CH. A genetic-algorithm-based approach to solve carpool service problems in cloud computing. IEEE Trans. on Intelligent Transportation Systems, 2015,16(1):352−364. [doi: 10.1109/TITS.2014.2334597]

[22] Jacob JC, Katz DS, Berriman GB, Good JC, Laity AC. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. Int'l Journal of Computational Science and Engineering, 2009,4(2):73−87. [doi: 10.1504/IJCSE.2009.026999]

[23] Deelman E, Kesselman C, Mehta G, Meshkat L. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In: Proc. of the 11th Int'l Symp. on High Performance Distributed Computing. IEEE, 2002. 225−234. [doi: 10.1109/HPDC.2002.1029922]

[24] Deelman E, Callaghan S, Field E, Francoeur, Graves R, Gupta N, Gupta V, Jordan TH, Kesselman C, Maechling P, Mehringer J. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In: Proc. of the 2nd Conf. on e-Science and Grid Computing. IEEE, 2006. 14−14. [doi: 10.1109/E-SCIENCE.2006.261098]

[25] Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther A, Robertson PJ, Theys MD, Yao B, Hensgen D, Freund RF. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 2001,61(6):810−837. [doi: 10.1006/jpdc.2000.1714]

[26] Thaman J, Singh M. Current perspective in task scheduling techniques in cloud computing: A review. Int'l Journal in Foundations of Computer Science & Technology, 2016,6:65−85. [doi: 10.5121/ijfcst.2016.6106]

[27] Panda SK, Jana PK. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. The Journal of Supercomputing, 2015,71(4):1505−1533. [doi: 10.1007/s11227-014-1376-6]

[28] Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 2011,41(1):23−50. [doi: 10.1002/spe.995]

[29] Armstrong R, Hensgen D, Kidd T. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In: Proc. of the Workshop on Heterogeneous Computing (HCW'98). IEEE, 1998. 79−87. [doi: 10.1109/HCW.1998.666547]

[30] Maheswaran M, Ali S, Siegel HJ, Freund RF. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 1999,59(2):107−131. [doi: 10.1006/jpdc.1999.1581]

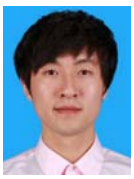
[31] Topcuoglu H, Hariri S, Wu M. Performance-Effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Parallel and Distributed Systems, 2002,13(3):260−274. [doi: 10.1109/71.993206]

[32] Graham RL, Lawler EL, Lenstra JK, Kan AHGR. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979,5(1):287−326. [doi: 10.1016/S0167-5060(08)70356-X]

[33] Jain R, Chiu D, Hawe W. A quantitative measure of fairness and discrimination for resource allocation. In: Proc. of the Shared Computer Systems. Hudson: Computer Science, 1998. http://www.cse.wustl.edu/~jain/papers/ftp/fairness.pdf

附中文参考文献:

[13] 孟凡超,初佃辉,李克秋,周学权.基于混合遗传模拟退火算法的 SaaS 构件优化放置.软件学报,2016,27(4):916−932. http://www.jos.org.cn/1000-9825/4965.html [doi: 10.13328/j.cnki.jos.004965]



沈尧(1986−),男,江苏扬州人,博士生,CCF 学生会员,主要研究领域为云环境中的资源调度,虚拟化技术.

秦小麟(1953−),男,教授,博士生导师,CCF 高级会员,主要研究领域为分布式数据管理与安全,时空数据处理技术.

鲍芝峰(1983−),男,博士,助教,主要研究领域为分布式数据处理,大数据探索.