文章编号:1674-2974(2016)10-0139-09

分布式系统中基于非合作博弈的调度算法*

童 钊[1†],肖 正[2],李肯立[2],刘 宏[1],李 俊[1]
(1.湖南师范大学 数学与计算机科学学院,湖南 长沙 410012;
2.湖南大学 信息科学与工程学院,湖南 长沙 410082)

摘 要:针对分布式系统中任务调度问题,根据分布式环境下的任务调度特性,建立了一个非合作博弈的多角色任务调度框架,在此基础上提出了一种基于纳什均衡联合调度策略的分布式强化学习算法.相比于静态调度算法,该算法需要更少的系统知识.能使调度器主动学习任务到达和执行的相关先验知识,以适应相邻调度器的分配策略,目标是使得调度器的策略趋向纳什均衡.模拟实验结果表明:所提出的算法在任务的预期时间和公平性上相对于OLB(机会主义负载均衡)、MET(最小执行时间)、MCT(最小完成时间)等同类调度算法具有更好的调度性能.

关键词:分布式计算;强化学习;任务调度;负载均衡
中图分类号:TP301.6 **文献标识码**:A

Scheduling Algorithm in Distributed Systems Based on Non-Cooperative Game

TONG Zhao[1†], XIAO Zheng[2], LI Ken-li[2], LIU Hong[1], LI Jun[1]
(1. College of Mathematics and Computer Science, Hunan Normal Univ, Changsha, Hunan 410012, China;
2. College of Computer Science and Electronic Engineering, Hunan Univ, Changsha, Hunan 410082, China)

Abstract: To address the task scheduling problem in distributed systems, based on an important feature of task scheduling in distributed computing environment, we have established a non-cooperative game framework for multi-layer multi-role, and put forward a distributed reinforcement learning algorithm of the joint scheduling strategy of Nash equilibrium. Compared with static scheduling algorithm, the proposed algorithm needs less system information. It enables the scheduler to actively learn task arrival, perform related knowledge and adapt to the adjacent scheduler allocation policy. The target is to move the schedulers strategy toward Nash equilibrium. Simulation experiments show that the proposed algorithm achieves excellent performance in expected response time of tasks and fairness, compared with classical scheduling algorithms such as OLB, MET and MCT.

Key words: distributed computing; reinforcement learning; task scheduling; load balance

---

* **收稿日期**:2015-07-24
基金项目:国家自然科学基金资助项目(61370095,61502165),National Natural Science Foundation of China(61370095,61502165),湖南师范大学大学生创新性实验项目(201501023)
作者简介:童 钊(1985—),男,湖南岳阳人,湖南师范大学讲师,博士
† 通讯联系人,E-mail:tongzhao@hunnu.edu.cn

随着科技的发展,基于 Internet 的计算方式发展迅速.如今云计算试图对线上资源进行虚拟化整合并使得需求更加透明[1-2].可以得知,当今的计算方式从独立的计算模式向网络化方向发展.云计算作为目前广泛部署的分布式系统,该系统可以提供巨大的计算能力满足并发请求,使得云计算在日常生活中变得更加重要.而在云计算系统中,任务的负载均衡是发挥其巨大潜力的关键因素[3].

在分布式系统中,存在大量的不确定性.由于网络不稳定的通信消耗以及计算能力的波动,导致任务的执行时间是随机的,这些参数取决于系统的当前状态.基于历史记录或工作负载建模的预测用来评估工作的执行时间[4].精度差和复杂度高是这类方法的缺点.此外,由于任务随机到达,其大小和 CCR(Computation to Communication Ratio,计算通信比)是无法预测的.因此,在分布式系统中动态算法受到广泛研究.在分布式计算系统中,批处理模式是一类动态调度方法.Min-Min,Max-Min 和 Suffrage 是三个典型的启发式批处理算法,这种算法为了获得任务到达和执行信息,很多时间用于等待和评估,缺乏实时性.相反,在线模式中,任务到达后被立即调度,如机会主义负载均衡(OLB,Opportunistic Load Balancing)的算法,最小执行时间(MET,Minimum Execution Time),最小完成时间(MCT,Minimum Completion Time)等调度算法[5-6].然而它们忽略了后续任务的到达和对整体性能的影响.为了获得全局优化,诸多学者提出了新的动态算法来适应任务的到达和执行过程,Kwok 提供一个资源计划系统来存储下面工作的资源[7].Yang 提出一种基于应用程序级和系统级的性能预测任务调度方案[8-9].

在分布式系统中,负载均衡涉及众多的调度器之间的协作.据统计,这类问题研究较少.就合作型调度而言,几个决策者合作决策以使整个系统性能最佳,Mashayekhy 等基于合作博弈理论研究了该调度问题[10-11];Peter 提出基于经济学中竞标概念的合同网协议[12-13];Subrata 等将网格负载均衡问题建立为一个非合作博弈模型[14].在上述研究中,博弈论是一个建立协同问题模型的主要工具.对于该类问题,现有大多数研究使用全局方法,目标是最小化所有任务的平均响应时间.实际上,这些算法在非马尔科夫环境下可能会导致失败[15].Subrata 等人研究非马尔科夫环境下任务调度算法的执行性能[13-15].

由于分布式系统存在的诸多不确定性,越来越多的研究人员将"强化学习"这种方法引入到该研究领域.通常,将"强化学习"定义为智能系统从环境到行为映射的学习,使得奖励信号(即,强化信号)的函数值最大,在强化学习中由环境提供的强化信号是对产生动作好坏的一种评价(通常为标量信号),而不是告诉强化学习系统 RLS(Reinforcement Learning System)如何去产生正确的动作.由于外部环境提供的信息资源较少,使得 RLS 必须靠自身的经历进行学习.通过这种方式,强化学习系统在行动-评价的环境中获得知识,改进行动方案以适合当前环境[15].

分布式环境中的处理机,在本文中称为处理单元 PEs(Process Elements).在分布式环境中,一个高效的任务调度算法是必要的.而在分布式环境的调度中,一个调度器可以将任务分配给其管理的处理单元,或其他相邻调度器.因此,调度器之间存在合作的关系,相对于集中式调度,分布式调度有两个重要的问题需要解决:1)调度器之间,尤其是来自不同域的调度器之间,如何调度.2)调度器之间是如何相互作用.

在本文中,针对自利型调度引入强化学习方法,使得在多个调度器之间进行协同调度时考虑环境具有的不确定性,同时还能适应其他的调度器的策略,达到最优调度的目的.

1 分布式系统调度框架

企业或者是科研机构中的所有资源通过互联网连接在一起,但他们可能属于不同的管理域.每一个域中都有一个或者多个调度器来处理到达的任务.由于缺少必要的资源或是计算效率低,任务可能会被转送到其他域.以分布式计算为例,用户并不关心任务请求是在何地处理的.图 1 是一个分布式计算系统,资源以管理域分割开,系统中有多个调度器.用户提交任务给这些调度器,并且由后台的调度器分派相应的计算单元去执行.

面对如此多的调度器,首先是怎样将它们组织起来.Rao 等提出了一个分布式资源共享框架[16],它提供了两个调度器之间详细的相互关系.因此,本

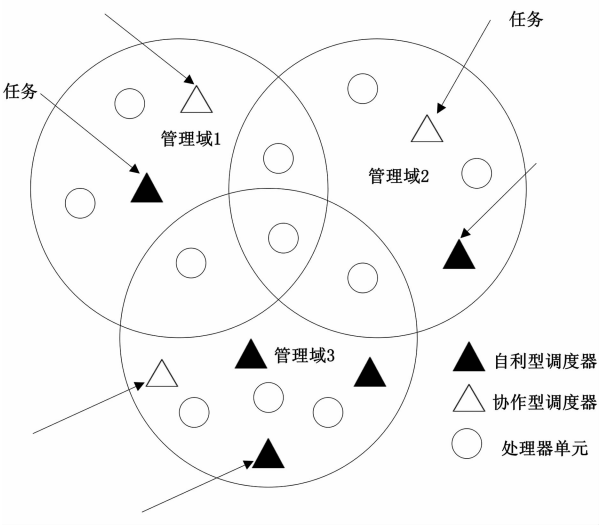

图 1　分布式计算系统
Fig. 1　Distributed computing system

本文提出了一个分层调度模型(图 2). 在更高层次中的调度器具有更广阔的视野,所以这种分层设计可以减少通信和提高效率. 在图 2 中,所有的 PE 都位于资源层,最终是由处理器分配任务给它们. 一个 PE 属于一个或多个自治域并且在多个调度器中管理,一个调度器可以和其管理的 PEs 进行通信. 换而言之,调度器可以知道它管理的每一个 PE 的状态,在同一层的调度器管理下一层的元素. 调度器之间通过细线连接起来,这些细线代表它们是相邻并且可以进行通信,当在调度时,任务可以被分配给调度器直接管理的 PE 或由其相邻的调度器

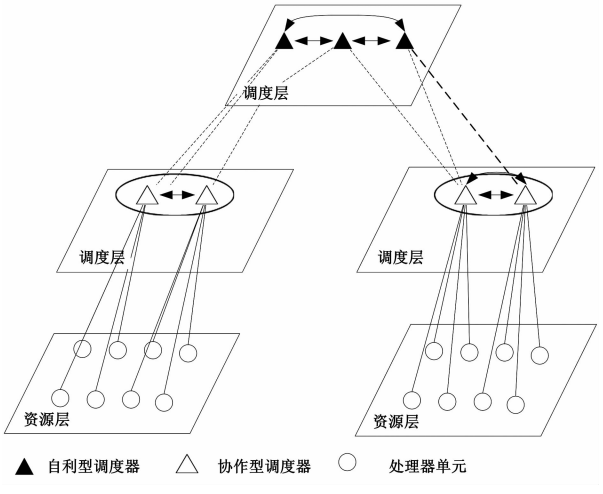

图 2　调度器组织架构
Fig. 2　The scheduler framework

对应管理的 PE 中,如果任务仍然不能被合理分配,则调度器将其移交给更高一层的调度器.

在分布式计算系统的资源可以分为 3 种角色:

1)PE(Process Element):用于计算.

2)协作型调度器:和其它的在相同组的协作型调度器共同去处理接收的任务.

共享相同的利益,在执行任务时又充分合作的调度器被称作协作型调度. 通常一个域形成一个组,本文不做讨论.

3)自利型调度器:能够自主接受或拒绝任务以达到自身利益最大化;其所代表的组之间通过达成一个联合战略,实现利益的双赢.

图 2 显示了调度器的组织结构和角色之间可能的关系,PE 由一个或多个调度器负责. 在中间层,四边形代表一个域,协作型调度器以组的方式划分,他们分配任务给在同一个组内的其它调度器或附属于它的 PE.

一个重要的问题是,这样一个可能由成千上万的资源组成的分布式计算平台是怎样协同进行工作的,各角色之间的各种协议使之成为一个松散耦合的系统.

1)站内分配协议:适用于 PE 和协作型调度器之间. Min-Min,MET,MCT 等调度算法属于该协议;

2)站内合作协议:适用于协作型调度器之间. 在一个组的协作型调度器互相帮助来优化整个组的性能,再调度和合作型博弈使用这样的协议;

3)站间合作协议:适用于自利型调度器之间. 在其所代表的组之间通过达成一个联合战略,实现利益的共赢. 这个协议的研究相对较少,是本文的研究重点.

这个框架来自于 Sakellariou, R. 提出的分层 semi-selfish 网格模型[17]. 但本文所提出的模型更普遍,是 Hanna H 中的模型成为一个特例[18]. 这个框架具有大规模的分布式计算环境的管理特性.

2　博弈型调度

在分布式系统中,任务的到达是一个服从一定概率分布的随机事件[19],每一个调度器将这些到达的任务作为一个序列缓存起来. 假设分布式计算系统可以处理 m 种不同的任务,集合 $T=\{t_1,t_2,\cdots,$

t_m}代表了所有可能的任务类型,PE 集合定义为 PE={PE_1,PE_2,…,PE_n},其中 n 代表 PE 的数目. 调度器的集合形式为 $S=\{S_1,S_2,\cdots,S_{|S|}\}$,其中 $|S|$ 为调度器数目,$\tau_i^j(k)$ 定义为在时间点 k,到达调度器 S_j 类型为 t_i 的任务,调度器 S_j 中的所有任务根据它们到达的时间定义为任务流,用 TF_j 代表调度器 S_j 上的任务流,每一个独立的任务流可能会有不同的到达过程,一个任务 $\tau_i^j(k)$ 可以被分配到相邻的调度器或者是附属于该调度器的 PE. 本文用一个元组 $\tau_i^j(k)$,$b>$(b∈PE∪S)代表一个任务分配,任务的响应时间是指:从被调度到达完成的时间间隔,以此作为任务调度的目标.

2.1 博弈模型

在图 2 的基础上,假设顶部调度层有 n 个自利型调度器,则自利型调度模型如图 3 所示.

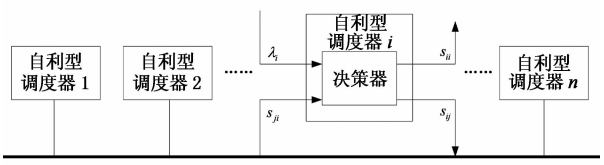

图 3 自利型协同模型

Fig. 3 The collaborative model of heter. scheduler

任务以 λ_i 的速率到达调度器 i;调度器以速率 s_{ii} 接收任务并以速率 s_{ii} 传递剩余的任务给调度器 j;调度器 i 也以速率 s_{ji} 接收来自调度器 j 传递的任务. 这些变量之间存在如下关系:

$$\lambda_i+\sum_{j\neq i}s_{ji}=s_{ii}+\sum_{j\neq i}s_{ij}, \tag{1}$$

$$\sum_i\lambda_i=\sum_i s_{ii}. \tag{2}$$

让 μ_i 为自利型调度器 i 所代表组的服务率,如果这个组建模成 M/M/1 类型排队系统,式(3)确保有限的排队长度.

$$s_{ii}\leqslant\mu_i. \tag{3}$$

根据式(2),式(3)可推导出以下关系:

$$\sum_i\lambda_i\leqslant\sum_i\mu_i. \tag{4}$$

在这个模型中,自利型调度器决定自身的接受率 s_{ii} 和拒绝率 s_{ij},调度器的策略可以用向量 $s_i\triangleq\{s_{i1},s_{i2},\cdots,s_{ii},\cdots,s_{in}\}$ 来表示多个调度器的联合策略表示为 $s\triangleq\{s_1,\cdots,s_n\}$.

自利型调度器有自己的利益目标,调度器根据给定的目标独立决策,对于分布式系统中的大多数应用,响应时间被广泛关注. 本文假设自利型调度器旨在最小化它们接受任务的响应时间,期望响应时间 $RT_i(s)$ 由式(5)给出:

$$RT_i(s)=CT_i(s)+TT_i(s). \tag{5}$$

式中:$CT_i(s)$为调度器 i 接受任务的期望完成时间;$TT_i(s)$为拒绝任务的期望传输时间.

如果 M/M/1 模型是适用的,期望完成时间可按式(6)计算:

$$CT_i(s)=\frac{1}{\mu_i-s_{ii}}. \tag{6}$$

Grosu 当把通信网络也看作为一个 M/M/1 排队系统时[19],一个任务的期望传输时间的近似值可以由式(7)得到. 在式(7)中参数 t 取决于平均任务长度,带宽等.

$$TT_i(s)=\frac{t}{1-t\sum_{i=1}^{n}\sum_n s_{ij}},\quad \sum_{i=1}^{n}\sum_n s_{ij}<\frac{1}{t}. \tag{7}$$

$\sum_{i=1}^{n}\sum_n s_{ij}$ 为通过网络的全部流量.

从式(5)~式(7)中可以得知,一个调度器的决策受到其它调度器调度策略的影响,自利型调度可以看作是一个博弈过程,通过竞争,慢慢形成一个没有调度器愿意改变它当前的调度策略的状态. 换而言之,没有调度器可以由单方面调整他们的策略来进一步减小他们的响应时间,这种状态在博弈论中称为纳什均衡. 事实上,自利调度器的协作旨在找到促成纳什平衡的联合策略.

根据博弈理论,本文使用下面的博弈来定义所研究的负载平衡问题.

定义 1(自利型调度器的负载平衡):自利型调度器的负载平衡问题由以下几个部分组成:

选手:n 个自利型调度器.

策略:联合策略 $s\triangleq\{s_1,\ldots,s_n\}$,由每一个调度器的接受率和拒绝率组成.

偏好:每个选手的偏好由它的期望响应时间 $RT_i(s)$ 来表示. 当且仅当 $RT_i(s)$ $RT_i(s')$,则相比于策略 s' 更偏好 s.

由于期望响应时间是连续、递增的凸函数,则上述博弈存在唯一的纳什平衡. Abdallah 提出了两种

基于排队理论的算法来得到博弈的均衡解[20].但是需要预测对象的到达率和服务率等参数.此外,这些算法在 Non-Markovian 环境下可能失败.所以接下来本文提出了一种基于在线学习的算法,它不再需要预测参数或使用受 Markov 限制的式(6),式(7).

2.2　基于学习的博弈调度算法

本文需要通过解决上述的博弈,实现负载均衡.一个解是指纳什均衡对应的联合策略,但满足式(1)~式(3)的限制,见以下的定义.

定义 2(纳什均衡)定义 1 中的负载均衡博弈的纳什均衡是指联合策略 s 对每个调度器 i($i=1,\cdots,n$)满足:

$$s_I \in \operatorname{argmin} RT_i, s_1, \cdots, \bar{s}_i, \cdots, s_n.$$

在给出该算法之前,为方便起见,本文将调度器的策略转变成概率形式,使得最终策略满足式(1)~式(3)而无需在计算时考虑.策略被重新定义成调度器分配任务给其他调度器的概率:

$\beta_i = \{\beta_{i1}, \ldots, \beta_{in}\}$($\sum_{j=1}^{n}\beta_{ij} = 1$) 代表这个新的策略,这里 β_{ij} 为调度器 i 分配任务给调度器 j 的概率,策略 β_i 和 s_i 是等价的.下列关系表示了接受率或拒绝率 s_{ij} 与概率 β_{ij} 之间的关系:

$$s_{ij} = \beta_{ij} \cdot [\lambda_i + \sum_j s_{ji}]. \tag{8}$$

由于 $\sum_{j=1}^{n}\beta_{ij} = 1$,式(1)得以满足.

当解决定义 1,2 的问题时,每一个调度器需要知道任务到达过程和处理过程以及其它调度器的策略和网络性能等,这使问题变得非常复杂.在本文中,使用机器学习方法去学习这个非合作博弈过程,机器学习是一门研究通过样本自动挖掘知识的各种方法的学科.

强化学习是一种在线学习方法,它对比较有利的行为进行强化,样本是历史的分配.这种方法的显著优势就是与模型无关,这就意味着不需要得到像任务到达过程以及网络性能等任何背景信息.本文使用强化学习算法中最常用的 Q 学习求解上述博弈调度问题.

在 Q 学习中 Q 函数代表累积响应时间,当任务被分配并完成后,根据任务的响应时间,更新该分配的 Q 值.式(9)表示一个任务被分配到 j 的时候调度器 i 的更新操作:

$$Q_i(\mathrm{Ag}t_j) = (1-\alpha)Q_i(\mathrm{Ag}t_i) + \alpha[r + \eta \max_k Q_i(\mathrm{Ag}t_k)]. \tag{9}$$

这里 $\mathrm{Ag}t_j$ 为调度器 j;$\alpha \in (0,1]$ 为学习率;r 为一个实时响应时间的单调递减函数;η 为折扣因子(Q 学习的一个参数).

α 按式(10)迭代,确保式(9)的收敛.

$$\alpha = \frac{1}{1 + \mathrm{visits}(\mathrm{Ag}t_j)}. \tag{10}$$

$\mathrm{visits}(\mathrm{Ag}t_j)$ 是迄今为止分配给 $\mathrm{Ag}t_j$ 的次数.

式(11)表示调度器 i 的策略 β_i.

$$\beta_{ij} = \frac{Q_i(\mathrm{Ag}t_j)}{\sum_k Q_i(\mathrm{Ag}t_k)}. \tag{11}$$

式(9)~式(11)给出了寻找最少响应时间的调度策略的算法,但是还存在两个问题:

1)当使用这个算法时,调度器难以稳定在均衡状态,因为每一个调度器的策略是摆动、不断调整的.所以必须设立一个机制去测试是否达到均衡并引导策略的调整.

当达到均衡时,没有一个调度器愿意违反他们当前的策略.当调度器企图从它的老策略 β_i^{old} 变成新策略 β_i^{new} 前,观察其它调度器的性能,如果 m 个调度器的性能由于上述策略的调整而得到提高,那么就表示 m 个调度器受益于改变,然后调度器 i 的策略根据式(12)的概率被更新.

$$\beta_i \overset{\frac{m}{n}}{\leftarrow} \beta_i^{\mathrm{old}} \text{ or } \beta_i \overset{1-\frac{m}{n}}{\leftarrow} \beta_i^{\mathrm{new}}. \tag{12}$$

在式(12)中,如果 $m=0$,则没有调度器性能提高,则 β_i^{new} 以概率一的方式被接受.如果 $m=n$,则说明离均衡存在较大的偏差,β_i^{old} 被保留.如果所有的调度器停止更新,那么当前的策略就认为是一个纳什均衡的联合策略.

2)任务转发循环.由于转发增加了任务的响应时间和浪费了网络带宽,所以多次转发的概率很小,但仍然存在这种可能性,任务一直被转发,形成了循环.为了避免这种异常情况的发生,一个任务在被调度器接受之前允许最多转发 3 次.由于不超过 3 次的转发限制,最终所有的到达任务都被接受,因此,式(2)得到满足.

单个调度器的算法如下所示.它是一种分布式算法,每个调度器独立运行这个算法,最后停止到唯一的均衡策略.

算法 1 基于强制学习的非博弈调度算法

Input: //输入
Set $\beta_{ij}=\frac{1}{n}$, $Q_i(j)=0, 1<j<n$;
Output: //输出
strategyβ_i
at Nash equilibrium; //纳什均衡策略
if Forward Times = 3 then //循环转发次数
Allocate jobs to itself; //分配任务
else
Allocate jobs randomly; //随机分配任务
Increase its forward times by 1;
end if
if Current job is assigned to scheduler j
then
Wait for its response time r
from$j$
; //等待响应时间
Compute new strategy β_{ij}^{new}
by Eq. 9 and 11; //按方法迭代
Observe the performance of other schedulers; //观察调度器性能
Update its strategy by Eq. 12; //更新操作
if $m=0$ then
inform other schedulers this possible equilibrium strategy;
if All the schedulers report their equilibrium strategy then
return β_{ij}^{new} ;
end if
end if
end if //整个算法结束

3 实验及结果分析

本节通过模拟实验研究在不同系统利用率、异构性下所提出方法的性能. 实验中使用的系统参数类似于 Abdallah 中的设置[20]. 有 32 个完全连接的候选调度器. 表 1 给出了参数配置. 调度器 i 的任务到达率根据到达比例因子 q_i 计算: $\lambda_i=q_i\times\lambda$,这里 λ 为全部到达率之和 $\sum_i\lambda_i$. 平均通信时间 t 被假设成 0.001 s,即传输一个任务平均花费 t s.

表 1 系统配置 I
Tab. 1 The system configuration I

调度器	1.6	7.12	13	14.18	19.22	23.26	27.32
服务率	10	50	100	100	150	200	250
到达比	0.002 5	0.01	0.02	0.025	0.04	0.05	0.07

使用的性能指标是均衡时的期望响应时间和公平指数. 公平指数在自利型调度器中是重要的,它被定义为:

$$\boldsymbol{I}(\boldsymbol{C})=\frac{\left[\sum_{j=1}^{n}C_j\right]^2}{n\sum_{j=1}^{n}C_j^2}. \tag{13}$$

输入 $\boldsymbol{C}$ 是向量, $\boldsymbol{C}=(C_1,C_2,\ldots,C_n)$,这里 C_j 是调度器 j 的期望响应时间. 如果 $\boldsymbol{I}=1$,所有的调度器有相同的响应时间,这表明负载达到均衡. 如果 $\boldsymbol{I}<1$,表明某些调度器可以承担更多的负载.

出于比较的目的,实现了两个负载均衡算法:

全局最优算法(GOS)[21]:这个算法的目标是最小化系统执行任务的期望响应时间,提供最优解. 通过解决下列非线性最优问题而得到每一个调度器的策略:

$$\min\ \overline{RT}=\frac{1}{\lambda}\sum_i\left[\sum_j s_{ij}CT_j+(\lambda_i-s_{ii})TT_i\right]. \tag{14}$$

比例算法(PROP_M):每个调度器依据各个调度器的处理速度,按比例分配给它自己或者其他调度器,如式(15)所示:

$$s_{ij}=\lambda_i\times\frac{\mu_j}{\sum_k\mu_k}. \tag{15}$$

下面将展示并分析实验的相关结果.

3.1 最佳响应策略的收敛性

每个调度器 i 的初始策略是元素 $1/n$ 组成的向量. 然后每个调度器在每一次迭代中改进和更新其策略. 在第一组实验中,本文研究最佳响应策略的收敛性,即当其余的调度器保持策略不变时的最好调度策略. 实验模拟一个 32 个调度器的异构系统以及设置系统利用率为 60%,即整个系统的到达率 λ 是 2 316. 初始调度策略在满足式(1)~式(3)的条件下随机产生,并且在其它调度器保持他们初始策略时让一个调度器运行该算法.

上述实验重复 20 次,每一次调度器的初始策略是随机生成的,取 20 次实验的平均响应时间. 图 4 显示了当系统利用率为 60%时,调度器的最后两个策略的期望响应时间的差. 大约 200 次迭代之后绝对差下降到 10^{-4},被认为达到收敛. 算法的收敛速度是 F. Noriguki[22]和 N. N. Dang[23]静态算法的 10 倍,这是因为算法需要更多的时间用于学习. 但本文认为较慢速度是可以容忍,因为在分布式系统处理的数以千计的任务中只有数百个任务受到收敛时间的影响[24-25].

图 5 显示了在不同的系统利用率下收敛时的迭代次数. 随着系统的利用率越来越高,本文的算法需要更多的迭代来获得最佳响应策略. 这是因为系统

只有在大量的任务到来之后才会达到稳定的状态.对于利用率为 90%时,收敛速度增加到大约 460 次迭代.

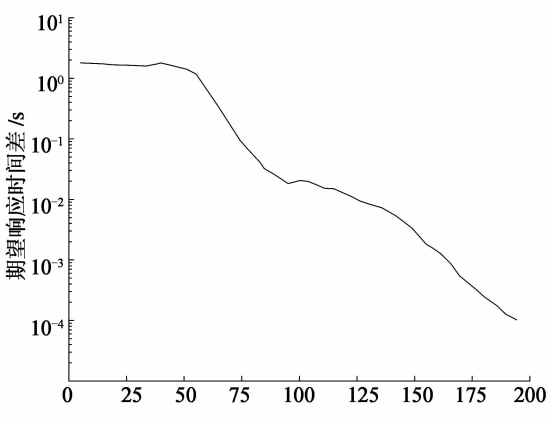

图 4　最佳响应策略的收敛性

Fig. 4　The best response strategy of convergence

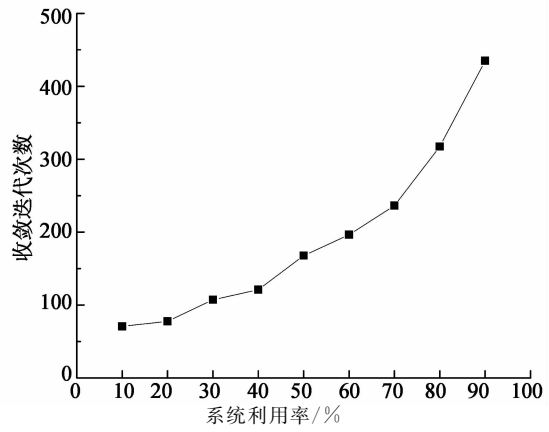

图 5　系统利用率对收敛性的影响

Fig. 5　The influence of system utilization of convergence

3.2　系统利用率的影响

模拟一个异构系统用于研究系统利用率的影响.这个系统由 32 个调度器(见表 1)组成.用 ρ 表示系统利用率.它由公式 $\lambda/\sum_i \mu_i$ 及条件 $0<\rho\leqslant 1$ 计算而得.且 ρ 越大,系统负载越大.

图 6,图 7 是当 ρ 从 10%提高到 90%时对应的计算结果.其中图 6 是不同系统利用率下期望响应时间.随着系统的利用率升高,期望响应时间变得更长.PROP_M 算法效果较差,因为对于性能较差的调度器负载过大.而 GOS 算法效果最好,它提供了最优系统解决方案.我们的算法有一个次优的性能,例如,当系统利用率为 60%时,响应时间比 PROP_M 算法少 25%左右,比 GOS 多 10%左右.

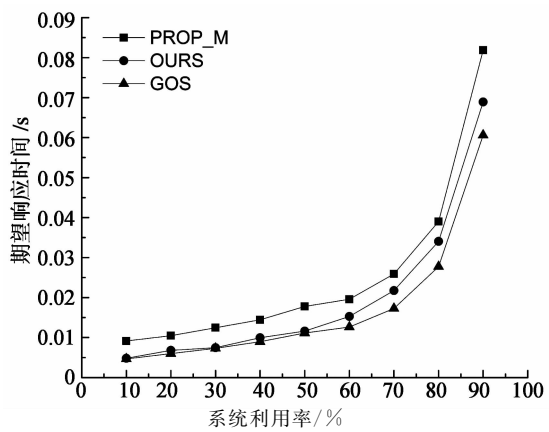

图 6　响应时间

Fig. 6　Expected response time

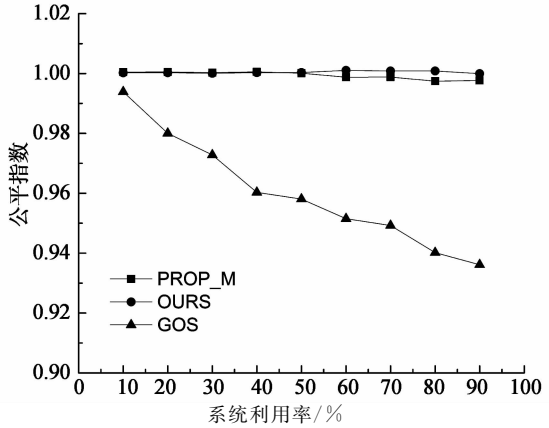

图 7　公平指数

Fig. 7　Fair index

图 7 显示了 3 种算法的公平指数.PROP_M 算法和我们算法的公平指数在 1 左右,对所有的调度器都很公平.GOS 算法的公平指数介于 1 和重负载情况下的 0.935 之间.图 7 也说明了调度器特性:由于自利性而缺乏充分的合作.

3.3　异构性的影响

在本节中,研究异构性对负载均衡性能的影响.处理器的速率是描述系统异构性一种简单的方法.速度偏斜度(speed skewness)常用于表征异构性,其定义为系统中计算机的最大处理速率与最小处理速率之比.通过改变速度偏斜度研究负载均衡方案的有效性.

实验模拟了一个包含 32 个调度器的系统,调度器分为 3 组,如表 2 所示.调度器 1～8 代表快速组,9～24 代表中速组,25～32 代表慢速组.起初,模拟一个同构系统,通过改变速度偏斜度进行了 6 个实验,系统利用率均设为 60%.总的达到速率如表 2 所示,任务到达比例 q_i 如表 1 所示.

表 2 系统配置 II

Tab. 2 The system configuration II

Speed skewness	1	4	16	36	64	100
$\mu_i(i=1:8)$	10	100	200	300	400	500
$\mu_i(i=9:24)$	10	50	50	50	50	50
$\mu_i(i=25:32)$	10	25	12.5	8.33	6.25	5
Over arrival rate	192	1 080	1 500	1 960	2 430	2 904

实验结果如图 8,图 9 所示,随着偏斜度的增加,系统的计算能力增强,进而期望响应时间减少. 对于同构系统而言,这 3 个算法有相同的性能. 当系统资源性能存在差异时,本文的算法和 GOS 期望响应时间减少的更快. 当偏斜度超过 50,本文的算法与 GOS 算法接近. 这意味着在高度异构系统下,本文所提出的算法是有效的.

从图 9 中的公平指数可知:增加速度偏斜度时,本文所提出的算法和 PROP_M 算法公平指数几乎为 1. GOS 公平指数从低偏斜度时的 1 下降到高偏斜度时的 0.82. GOS 算法的分配并不能保证平等的期望响应时间,尤其在高偏斜度情况下. 均衡的负载和接近最优的性能是本文提出算法的主要优势.

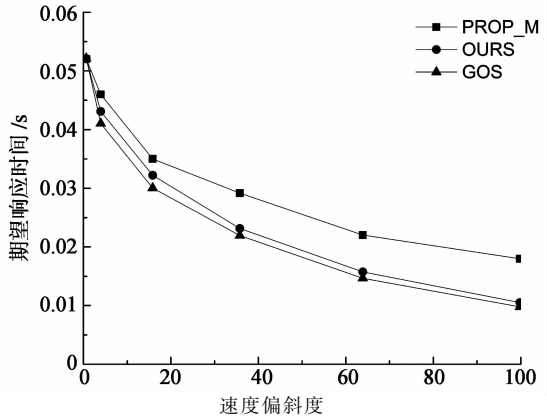

图 8 期望响应时间

Fig. 8 Expected response time

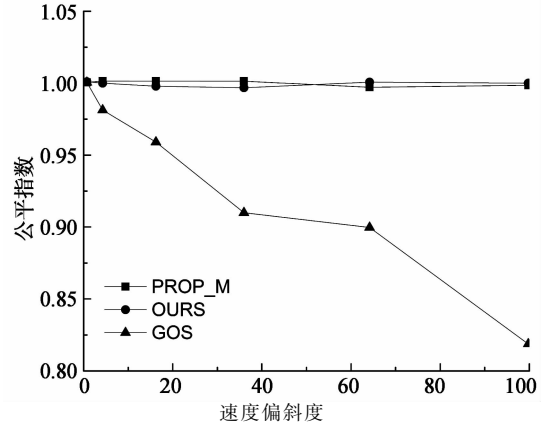

图 9 公平指数

Fig. 9 Fair index

4 结 论

本文研究并行分布式系统中调度问题. 在这种情形下,调度器不仅适应任务到达的随机性和系统负载的多变性,而且适应其他调度器的分配策略. 本文基于强化学习提出了相应的调度算法. 调度器主动学习任务到达和执行以及与之相邻的调度器行为知识,通过这种方法,在一定程度上实现了调度器之间的协作并且降低了平均响应时间. 模拟实验证明,该算法相比于几个经典的调度算法在任务的预期时间和公平性上具有更好的性能.

参考文献

[1] XU Yu-ming, LI Ken-li, HE Li-gang, *et al*. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems[J]. IEEE Transactions on Parallel & Distributed Systems, 2014,26(12):3208－3222.

[2] 郑明玲,蒋句平,袁远,等. 一种面向大规模计算机的监控管理系统[J]. 湖南大学学报:自然科学版,2015,42(4):107－113.
ZHENG Ming-ling, JIANG Ju-ping, YUAN Yuan, *et al*. A monitoring and management system for the large-scale computer[J]. Journal of Hunan University: Natural Sciences, 2015, 42(4):107－113. (In Chinese)

[3] 林闯, 苏文博, 孟坤, 等. 云计算安全:架构、机制与模型评价[J]. 计算机学报, 2013,36(9):1765－1784.
LIN Chuang, SU Wen-bo, MENG Kun, *et al*. Cloud computing security: architecture, mechanism and modeling[J]. Chinese Journal of Computers, 2013, 36(9): 1765－1784. (In Chinese)

[4] GUTIERREZ-GARCIA J O, RAMIREZ-NAFARRATE A. Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines[J]. IEEE Transactions on Services Computing, 2015,8(6):916－929.

[5] BRAUN T D, SIEGEL H J, BECK N, *et al*. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems[J]. Journal of Parallel and Distributed Systems,2011,61(6):810－837.

[6] LI Ken-li, TANG Xiao-yong, LI Ke-qin. Energy-efficient stochastic task scheduling on heterogeneous computing systems [J]. IEEE Transactions on Parallel and Distributed System, 2014, 25(11): 2867－2876.

[7] KWOK Y K, HWANG K, SONG S S. Selfish grids: game-theoretic modeling and NAS/PSA benchmark evaluation[J]. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(5):621－636.

[8] YANG L, SCHOPF J M, FOSTER I. Conservative scheduling: using predicted variance to improve scheduling decisions in dynamic environments[C]//International Conference on Supercomputing. Leipzig, Germany: IEEE Computer Society,

2003: 15－21.

[9] ZHANG Long-xin, LI Ken-li, ZHANG Fan, *et al*. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster[J]. Information Science, 2015,319(20):113－131.

[10] 朱夏, 宋爱波, 东方, 等. 云计算环境下基于协同过滤的个性化推荐机制[J]. 计算机研究与发展, 2014, 51(10):2255－2269.
ZHU Xia, SONG Ai-bo, DONG Fang, *et al*. A collaborative filter recommendation mechanism for cloud computing[J]. Journal of Computer Research and Development, 2014, 51(10):2255－2269. (In Chinese)

[11] MASHAYWKHY L, NEJAD M M, GROSU D, *et al*. Energy-aware scheduling of MapReduce jobs for big data applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2015,26(10):2720－2733.

[12] TANG Zhuo, MO Yan-qing, LI Ken-li, *et al*. Dynamic forecast schedule algorithm for virtual machine placement in cloud computing environment[J]. Journal of Super Computing, 2014,70(3):1279－1296.

[13] PETER S, GIVARGIS T. Component-based synthesis of embedded systems using satisfiability modulo theories[J]. ACM Transactions on Design Automation of Electronic Systems, 2015,20(4):1－27.

[14] SUBRATA R, ZOMAYA A Y, LANDFELDT B. Cooperative power-aware scheduling in grid computing environments[J]. Journal of Parallel and Distributed Computing, 2010,70(2):84－91.

[15] LIU Chu-bo, LI Ken-li, LI Ke-qin. Strategy configurations of multiple users competition for cloud service reservation[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(2):508－520.

[16] RAO I, HUH E N, LEE S Y, *et al*. Distributed, scalable and reconfigurable inter-grid resource sharing framework[C]// Computational Science and Its Applications. Glasgow, UK: Springer, 2006: 390－399.

[17] SAKELLARIOU R, ZHAO H. A low-cost rescheduling policy for efficient mapping of workflows on grid systems[J]. Scientific Programming, 2004,12(4):253－262.

[18] HANNA H, MOUADDIB A I. Task selection problem under uncertainty as decision-making[C]//First International Joint Conference on Autonomous Agents and Multiagent Systems: Part I. Bologna, Italy: ACM, 2002: 1303－1308.

[19] 陈康, 郑纬民. 云计算:系统实例与研究现状[J]. 软件学报, 2009,20(5):1337－1348.
CHEN Kang, ZHENG Wei-ming. Cloud computing: system instances and current research[J]. Journal of Software, 2009, 20(5):1337－1348. (In Chinese)

[20] GROSU D, CHEONOPOULOS A T, LEUNG M Y. Noncooperative load balancing in distributed systems[J]. Concurrency & Computation Practice & Experience, 2008,20(16):1953－1976.

[21] ABDALLAH S, LESSER V. Learning task allocation via multi-level policy gradient algorithm with dynamic learning rate[C] //International Joint Conference on Artificial Intelligence. Sydney, Australia: Springer, 2005: 76－82.

[22] KIM S, WEISSMAN J B. A genetic algorithm based approach for scheduling decomposable data grid applications[C]//International Conference on Parallel Processing. Dresden, Germany: ACM, 2004: 406－413.

[23] FUJIMOTO N, HAGIHAEA K. A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks [C]//International Symposium on Applications and the Internet Workshops. Tokyo, Japan: IEEE Computer Society, 2004: 674.

[24] N N DANG, S HWANG, S B LIM. Improvement of data grid's performance by combining job scheduling with dynamic replication strategy[C]//the 6th International Conference on Grid and Cooperative Computing. Xinjiang , China: IEEE Computer Society, 2007: 513－520.

[25] 费长江, 吴纯青, 赵宝康, 等. 一种卫星移动通信语音业务半持续调度机制[J]. 湖南大学学报:自然科学版, 2015,42(8): 108－115.
FEI Chang-jiang, WU Chun-qing, ZHAO Bao-kang, *et al*. A semi-persistent scheduling mechanism for voice service in satellite mobile communication[J]. Journal of Hunan University: Natural Sciences, 2015,42(8):108－115. (In Chinese)