分布式系统中多用户网络应用的概率型调度算法研究

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

摘 要: 多用户网络应用是分布式计算中最主要的形式之一. 为了充分挖掘分布式系统中的计算资源,任务调度是解决该问题的关键. 然而,由于多用户网络应用中存在的不确定性,使得当前的调度方法在动态性、实时性、适应性等方面都存在诸多不足. 考虑到用户实时性需求,本文提出了概率型调度的思想. 该思想将任务的分配看作概率事件,以用户角度的最短响应时间为目标,给出了多用户网络应用的排队模型,并进一步将调度定义为一个非线性规划问题. 分析表明上述方法在任务到达过程、服务率方面存在限制,进而提出了一个基于强化学习理论自适应调度算法. 该算法首先利用 Markov 决策过程(MDP)描述该调度问题,然后对任务到达过程和服务率知识进行在线的学习. 一旦获得任务分配概率,遵从该概率可进行快速的任务调度. 实验表明上述两个算法相比于 Min-Min、Max-Min、Suffrage、ECT 四种经典调度算法具有更短的平均响应时间. 除此性能外,通过实验分析了该概率型调度方法的稳定性.

关键词: 分布式计算; 多用户; 任务调度; 排队模型; 概率型调度
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2016)07-1679-10
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.07.023

A Queueing Model and Probabilistic Scheduling for Multi-user Network Applications

TONG Zhao[1,2], XIAO Zheng[2], LI Ken-li[2]
(1. *College of Mathematics and Computer Science, Hunan Normal University, Changsha, Hunan* 410082, *China*;
2. *College of Information Science and Engineering, Hunan University, Changsha, Hunan* 410082, *China*)

Abstract: Multi-user network application is one of the most popular forms of distributed computing. To fully exploit computing resources in distributed systems, task scheduling is critical. However, in scheduling of multi-user network application because lots of uncertainties exist such as task arrival, task completion time, etc., the state of the art scheduling approaches fail in dynamic, real time, or adaptability. On account of the real time property, we put forward the concept of probabilistic scheduling to compress scheduling time, which regards task allocation as a probabilistic event. Unexpectedly, compared with traditional scheduling approaches which are always determinate, probabilistic scheduling has other advantages like insensitivity to task execution time estimation and steady performance. Based on the idea of probabilistic scheduling and considering the shortest response time from the perspective of users, a queueing model is given for multi-user network application, and scheduling is defined as a non-linear programming problem. But due to its limitation on task arrival process and service rate, a self-adaptive algorithm is proposed by use of reinforcement learning theory. The scheduling problem is described by Markov Decision Process (MDP), and then task arrival process and service rate can be learned on line. Once the allocation probability is acquired, scheduling is quite fast by following such probability distribution. The two algorithms are validated and they outperform such four classic algorithms as Min-Min, Max-Min, Suffrage, and ECT at the average response time. Except for the mean of response time, their variance is also examined to confirm the stability generated by probabilistic scheduling.

Key words: distributed computing; multi-user; task scheduling; queueing model; probabilistic scheduling

---

收稿日期:2013-02-27;修回日期:2013-07-26;责任编辑:郭游
基金项目:国家自然科学基金重点项目(No. 61133005);国家自然科学基金青年项目(No. 61402157);湖南省自然科学基金(No. 13JJ4038);湖南师范大学青年基金(No. 11404)

1 引言

多用户网络应用是分布式计算中最主要的形式之一,大量应用对分布式系统的实时响应能力提出了更高的要求.其中,数据库管理系统的查询处理,尤其是Web数据库,是一类较常见的多用户网络应用.在这些应用中,多个查询请求的随机到达,它们的处理必须分配到后台分布式系统中某个处理单元上执行.任务调度即决定在这种情况下,何时何资源处理这些查询.任务调度是挖掘并行分布式系统巨大处理能力的关键[1].

Google搜索服务是多用户网络应用的一个实例,遍布全球的大量用户向Google服务器发送关键字查询请求.Google搜索引擎采用MapReduce技术将原始请求分割为几类任务,然后将这些任务映射到服务器上处理.对于这样的多用户共享的分布式系统及其多用户网络应用包含三种不确定性:

(1)何时及任务将到达的数目未知.用户何时将发起何种查询请求无法预知.

(2)因为处理器和网络的动态性,处理单元处理任务的完成时间不可预测.

(3)任务在被处理之前的等待时间不可预测.因为系统不是专用的,来自其他用户的任务也将分配到这些处理资源上.

由于上述的不确定性,系统完整的信息不能提前预知.因此调度只能够发生在运行时.只有充分利用计算资源,才能适应这些不确定性.另外,在多用户网络应用中,用户通常在线等待响应.因此实时性是这类调度算法的一个重要制约因素.所以复杂的调度算法将不可行,短的响应时间(任务从用户提交至完成的时间)是调度的目标.然而,上述状态的不确定性和系统的异构性给该调度问题带来了挑战.

任务调度是NP完全问题[2,3],即使是独立非抢占式调度.研究发现目前的调度算法是在任务和处理单元之间建立一个确定的匹配,为分配的任务指定唯一的单元.确定型任务调度在每次调度时需要重新计算分配方案.为了压缩调度时间,首次提出概率型调度,依据一定的概率分布来为任务指定处理单元.由于概率型分配不会立即引起性能的较大下滑,而且不需要实时更新.因此相比于确定型调度的重新计算,缩减了大量的调度时间.此外,确定型调度性能取决于状态估计的准确度.如果状态估计时发生偏离,调度的性能也将发生下降.而概率型调度能够缓解这种性能的降级.第三,因为任务到达和服务时间是随机的,因此性能指标的方差变得重要.方差越小,表明调度算法越稳定.实验表明概率型调度同时也能降低性能的方差.

出于上述原因,本文与以往工作不同在于基于概率调度的思想对能够自适应固有的不确定性,快速生成响应时间较低的动态调度算法进行了探索.提出的概率型调度算法具有:任务执行时间估计的不敏感和性能稳定的两个特性.

基于排队论建立调度模型,将系统状态的动态性质转化为概率,这使得系统状态具有一定的可预测性,从而使得静态调度方法成为可能.进一步将多用户网络应用的调度问题定义为一个非线性规划问题,并获得一组最优的分配概率.任务直接根据此概率分布进行分配,大大提高了调度的速度.但是由于排队论中的一些假设前提与实际环境不符,上述方法的适用性较低,因此将机器学习方法引入调度中,提出了一种能适应系统状态动态性的调度算法,相比于第一种方法,此方法没有明显增加调度的时间复杂度.首先,MDP被用来描述该调度问题,然后给出解决该MDP问题的基于强化学习的在线调度算法.实验表明,本文提出的算法在平均响应时间上优于Min-Min[4,5],Max-Min[4,5],XSufrage[6],和ECT[6]四种经典调度算法.最后,分析了响应时间的方差,证实了本文算法的稳定性.通过对系统状态估计的敏感性分析,本文所提出的算法也表现出较好的鲁棒性.

2 相关工作

对于任务调度问题,近年来已经有较多研究,但这个问题似乎远没有到达终点.这些算法主要分为两类:一类是在编译时确定何时何处来处理任务的静态调度,另一类是在运行时进行决策的动态调度.如前所述,静态调度发生在编译时.有关任务和系统状态的知识需要预先获悉.但由于多用户网络应用中存在的不确定性,这些知识仅在运行时才能确定,因此,传统的静态调度无法适用本文研究的问题.

动态调度中,系统的状态只有当任务运行时才能确定.随着网络计算的出现,如云计算等,动态调度成为研究热点.多用户网络应用的调度属于动态调度的范畴.动态调度包括两个阶段:系统状态估计和决策制订.分布式系统的动态调度技术主要被分为:发送者发起,接收者发起,和对称发起三类[7].在发送者发起算法中[8,9],过载的节点向负载较轻的节点转移任务;在接收者发起算法中[10,11],负载较轻的节点向负载重的节点发出任务迁移请求;最后一类,发送者和接收者能够对称地发起负载迁移请求[12,13].实际上,这些技术都试图在处理单元之间达到负载均衡.Makespan是并行分布式计算中一个主要性能指标,它反映的是系统的整体性能.而在多用户网络应用中,用户期望他们的请求能够被迅速处理.因此,本文关注如何进行任务分配以使用户得到快速响应.

在上述调度算法中,节点既是处理单元也是任务调度器,分配任务给自己或其它节点,这种调度方式可能会增加调度开销. 集中式或者分布式调度则可以弥补这种性能损失. 本文采用的是集中式调度方法,选取其中一个处理节点用于负责任务的调度.

在分布式系统中任务调度的一个难点是资源的动态性质[14]. 下列方法常被用于适应这种动态性,估计系统的状态. (1)利用第三方软件组件的实时信息,如 GIS (Grid Information Service)是网格中一种软件组件[15],用来维护计算网格中用户、软件、服务和硬件的信息,一旦收到请求便返回相应的信息. (2)性能预测. 一般而言,静态算法在调度时依赖于性能评估. 预测可以基于历史记录[16]或者负载建模[17,18]. 本文第一个算法即通过建立排队模型进行预测. (3)再调度. 基于当前最新的资源状态,再调度技术改变以前的调度决策[19,20]. 上述三种方法,各有优缺:方法(1)需要额外的通信开销;方法(2)很难保证一个简单的算法能够提供较高的预测精度;方法(3)需要改变系统基础结构以支持任务迁移.

本文引入机器学习技术主动学习动态性质而不是仅仅像上述方法一样获取动态信息,该学习方法仅涉及少量的通信. 在上述的动态调度算法中,每一个任务调度的时间复杂度至少与处理器的数量成正比,这可能导致任务的实时性需求无法满足,而本文的概率型调度仅具有常量的时间复杂度,是一个快速调度算法. 此外,以前的工作通常没有考虑调度的稳定性,而这一性质在动态环境下是十分重要的. 另外,动态调度还取决于系统状态估计. 在这两方面,本文的概率型调度算法表现出较高的稳定性和对状态估计的容错能力,这是本文与以往工作的重要不同之处.

3　排队模型

在描述排队模型之前,首先分析一下响应时间和 Makespan 之间的关系.

3.1　响应时间与 Makespan

响应时间指系统中某个任务从提交到完成所经历的时间[21],而 Makespan 指从第一个任务到达至最后一个任务完成之间的时间间隔,用于表示完成所有到达任务的总时间. 这两个定义有不同的含义. Makespan 用于测量一个系统并行化应用的并行化程度. 而响应时间是为每一个用户服务的瞬时速率. 一般而言,较大的响应时间导致较长的 Makespan,但有时在特定的环境下,不得不适当延长 Makespan,减少任务的等待时间,使得响应时间降低. 下面的示例说明即使在同样的 Makespan 下,不同的调度方案可以得到不同的响应时间.

假设一个简单分布式系统具有 3 个处理单元 PU_1, PU_2 和 PU_3. 系统接收 t_1 和 t_2 两类任务. t_1 类型任务在 PU_1,PU_2 和 PU_3 执行所耗费的时间分别是 6,9,9;t_2 类型任务在 PU_1,PU_2 和 PU_3 执行所耗费的时间分别是 3,5,6. 用 $\tau_i^j\tau_i^j$ 表示到达的任务,其中 i 代表任务类型,j 代表按时间先后到达序列号. 从开始时刻起,四个任务 $\{\tau_1^1,\tau_2^2,\tau_2^3,\tau_2^4\}$ 依次到达,他们的到达时刻分别是 0,1,2 和 5. 图 1 给出了两个具有最短 Makespan 9 的调度方案,但是在图 1(a)中响应时间为(6 + 6 + 5 + 4)/4 = 5.25. 任务 τ_2^4 在执行之前需要等待一个时隙. 如果将任务 τ_1^1 从 PU_1 迁移到 PU_3 上,得到的调度方案如图 1(b)所示,其响应时间为(9 + 5 + 3 + 3)/4 = 5. 从这个例子中可以发现,如果能够预测在任务 τ_1^1 到达后有大量的 t_2 类型任务到达,则可以将性能更好的计算资源预留给后续任务从而减小响应时间. 即牺牲当前的任务,增加其响应时间为后续的任务服务.

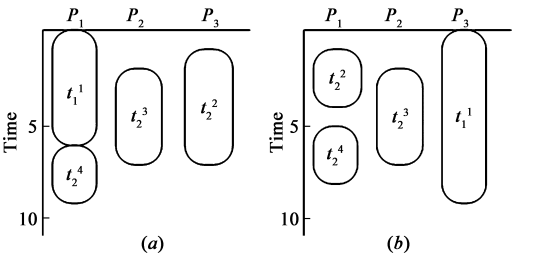


图1　示例:响应时间和Makespan对比

3.2　形式化描述

根据分布式系统中多用户网络应用的信息流,基于排队论建立了一个简单的模型,如图 2 所示.

根据任务的大小进行分类,调度器负责将来自用户的任务分配给系统中的各处理单元. 到达的任务首先被压入调度器的到达队列,然后调度器处理队列中所有的任务,每一次从到达队列提取一个任务,分配给某处理单元并将其放入处理单元的本地队列中. 处理单元以先进先出(FIFO)的方式处理本地队列中的任务. 这些处理单元是异构的. 在多用户网络应用环境中,用户期盼得到快速响应. 因此,调度问题即为:将任务映射到能够给用户较快响应的处理单元.

分布式系统能够处理 m 类任务,用户提交的任务类型集合为 $\{t_1,t_2,\cdots,t_m\}$. 假设任务到达过程为泊松

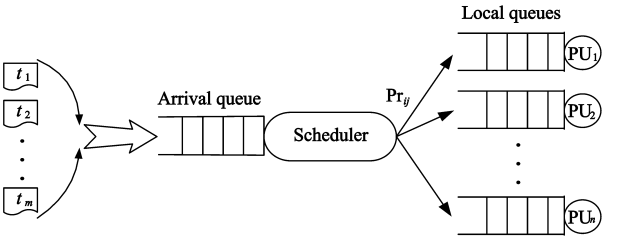


图2　任务调度的排队模型

(Poisson)流,λ 是任务到达率. 定理 1 证明了本地队列的到达过程也是一个泊松流,两个连续到达任务的时间间隔服从指数分布,排队论中 M/G/1 模型被应用于本地队列. 类型为 t_i 的任务在处理单元 PU_j 上的服务率用 μ_{ij} 表示. 图 2 中 $\Pr_{ij}$ 表示类型为 t_i 的任务调度到 PU_j 上概率. 为了简化,PU_j 的服务率视为 μ_{ij} 的均值,如下:

$$\lambda_j = \sum_{i=1}^{m} \lambda_{ij} = \sum_{i=1}^{m} P^{ij}\lambda \tag{1}$$

PU_j 上的任务到达率 λ_j 定义如下(见定理 1):

$$\lambda_j = \sum_{i=1}^{m} \lambda_{ij} = \sum_{i=1}^{m} \rho_{ij}\lambda = \sum_{i=1}^{m} \Pr_{ij}\rho_i\lambda \tag{2}$$

其中,λ_{ij} 是 PU_j 上类型为 t_i 的任务到达率,ρ_{ij} 是 PU_j 的本地队列中任务类型为 t_i 的概率. ρ_i 是到达队列中任务类型为 t_i 的概率.

引理 1 当 $m=1$ 且 $\mu \geqslant \lambda$(μ 是调度器的调度速率),则本地队列的任务到达过程为泊松流;设唯一的任务类型为 t_i,其到达率为 $\lambda_{ij}=\Pr_{ij}\lambda_i^t$,其中 λ_i^t 是类型为 t_i 的任务到达率.

证明 对于条件 $m=1$,如果到达过程是一个到达率为 λ 的泊松流,则类型为 t_i 的任务到达过程是一个到达率 $\lambda_i^t=\lambda$ 的泊松流. 在图 2 的模型中,$\mu \geqslant \lambda$ 意味着由于系统具有较高的调度率,到达队列总是保持为空. 无论任务何时到达,总是能得到立即调度. 到达队列中两个连续任务的到达时间间隔用随机变量 ζ 表示,服从指数分布,即 $\zeta \sim \exp(\lambda_i^t)$. PU_j 本地队列中两个连续任务的到达时间间隔用随机变量 T 表示. 假设从上次分配到 PU_j,第 l 个任务再次被分配给 PU_j,这个时间间隔 $T=\Pr_{ij}(1-\Pr_{ij})^{n-1}\cdot l\zeta$. 一般有,

$$T = \sum_{n=1}^{\infty} T = \sum_{n=1}^{\infty} \Pr_{ij}(1-\Pr_{ij})^{n-1}\cdot n\xi = \frac{1}{\Pr_{ij}}\xi$$

由于 $\zeta \sim \exp(\lambda_i^t)$,事件 $T<x$ 的概率为

$$P(T<x) = P\left(\frac{1}{\Pr_{ij}}\xi < x\right) = \begin{cases} 1-\mathrm{e}^{-\Pr_{ij}\lambda_i^t x} & ,\ x>0 \\ 0 & ,\ x<0 \end{cases}$$

以上概率分布表明 PU_j 上的任务到达过程也是一个泊松流,且到达率为 $\Pr_{ij}\lambda_i^t$.

证毕.

定理 1 PU_j 的本地队列上任务到达过程为泊松流,且到达率为 λ_j.

证明 类型为 t_i 的任务到达过程是一个到达率 $\lambda_i^t=\rho_i\lambda$ 的泊松流,其中 ρ_i 表示当前到达任务的类型为 t_i 的概率. 实质上,PU_j 的本地队列上的到达过程是各种类型的任务流汇聚而成,由引理 1 可知,每一种类型的任务流是泊松流,根据泊松分布的可加性,汇集任务流也是一个泊松流,且其到达率为各个任务流到达率之和. 根据引理 1,有

$$\lambda_j = \sum_i \Pr_{ij}\lambda_i^t = \sum_i \Pr_{ij}\rho_i\lambda = \sum_i \rho_{ij}\lambda$$

证毕.

本地队列是一个典型的排队系统. 基于排队论中的结论[22],可以得到以下信息,如表 1 所示.

表 1 排队模型中使用的符号

符号	含义
m	任务类型数
n	处理单元数
t_i	第 i 种任务类型
PU_j	第 j 个处理单元
λ	系统的任务到达率
λ_{ij}	PU_j 上类型为 t_i 的任务到达率
λ_j	PU_j 上任务到达率
M	调度器的调度服务率
μ_{ij}	PU_j 上类型为 t_i 的任务服务率
μ_j	PU_j 上任务服务率
Pr_{ij}	将类型为 t_i 的任务分配给 PU_j 的概率
ρ_i	当前到达任务类型为 t_i 的概率
ρ_{ij}	P_j 本地队列中任务类型为 t_i 的概率
L_s^{arr}	到达队列的平均队列长度
L_s^{locj}	P_j 本地队列的平均队列长度
L_q^{arr}	到达队列的平均等待长度
L_q^{locj}	P_j 本地队列的平均等待长度
T_q^{arr}	到达队列的平均等待时间
T_q^{locj}	P_j 本地队列的平均等待时间
T_s^{arr}	到达队列的平均逗留时间
T_s^{locj}	P_j 本地队列的平均逗留时间
T_q	系统的平均等待时间
T_s	系统的平均逗留时间

(1)平均队列长度 L_s:队列中任务总数,包括正在被服务的任务.

对于 PU_j 上的本地队列 $L_s^{loc_j}=\dfrac{\lambda_j}{\mu_j-\lambda_j}$

对于调度器上的到达队列 $L_s^{arr}=\dfrac{\lambda}{\mu-\lambda}$

(2)平均等待长度 L_q:队列中等待服务的任务数目.

对于 PU_j 上的本地队列 $L_q^{loc_j}=\dfrac{\lambda_j^2}{\mu_j(\mu_j-\lambda_j)}$

对于调度器上的到达队列 $L_q^{arr}=\dfrac{\lambda^2}{\mu(\mu-\lambda)}$

(3)平均等待时间 T_q:任务接受服务之前的等待时间. T_q 由两部分组成:到达队列和本地队列中的平均等待时间.

$$T_q = T_q^{arr} + \frac{1}{n}\sum_j T_q^{loc_j} = \frac{\lambda}{\mu(\mu-\lambda)} + \frac{1}{n}\sum_j \frac{\lambda_j}{\mu_j(\mu_j-\lambda_j)} \tag{3}$$

(4)平均逗留时间 T_s:任务在系统中耗费的总时间,包括服务时间. T_s 由两部分组成:到达队列和本地队列中的平均逗留时间.

$$T_s = T_s^{arr} + \frac{1}{n}\sum_j T_s^{loc_j} = \frac{1}{\mu-\lambda} + \frac{1}{n}\sum_j \frac{1}{\mu_j-\lambda_j} \quad (4)$$

在这个调度模型中,如果调度器能够迅速的进行任务调度,即$\mu \geqslant \lambda$,则 T_q 和 T_s 中的第一项可以忽略. 第 4 节中提出的概率调度算法是这一类算法,能在常量时间内完成调度,从而使条件$\mu \geqslant \lambda$ 得到满足. 使用排队论,不确定性通过概率参数被消除,基于这些概率分布,可以获得最佳的调度方案. 在下一节中,将详细描述这一算法.

4　概率型调度算法

在上述模型下,本文提出了两个概率型调度算法,达到最小化任务的响应时间的目的. 首先,将该调度问题定义为一个非线性规划问题;但由于受到一些与实际的前提不相符的条件约束,接下来使用 *MDP* 对调度问题进行定义,并提出了相应的基于学习的调度算法.

4.1　基于非线性规划的算法

对于多用户网络应用,用户期望得到快速响应. 因为分配是一个随机事件,所以采用期望响应时间作为目标函数. 在上节调度模型下,类型为 t_i 的任务平均响应时间定义如下:

$$RT_{avg} = T_s \approx \frac{1}{n}\sum_j \frac{1}{\mu_j-\lambda_j} = \frac{1}{n}\sum_j \frac{1}{\mu_j - \sum_i \Pr_{ij}\rho^i\lambda} \quad (5)$$

因为采用概率型调度机制,调度器能快速的完成任务的分配,到达队列中的逗留时间相比于本地队列的等待时间可以忽略不计. 因此,式(5)中的近似是合理的.

调度算法试图找到使得平均响应时间最短的一组最优分配概率 $\Pr_{ij}^*$. 因此,可将调度问题进行如下定义:

$$\begin{aligned} &Min \quad RT_{avg} \\ &\text{s.t.} \quad 0 \leqslant \Pr_{ij} \leqslant 1 \\ &\sum_j \Pr_{ij} = 1 \end{aligned} \quad (6)$$

对于式(6)中的非线性规划问题,有许多求解方法. 本文中采用 MATLAB 工具箱中的"fmincon"求解.

上述的排队模型和基于非线性规划的算法给出了一个求解动态任务调度的方法,但同时该方法预置了一些假设:任务到达过程认为是泊松过程,即两个连续任务的到达时间间隔服从指数分布. 然而,这一假设在实际环境中可能并不成立. 因此,在本小节中,针对不同的任务到达过程对算法性能进行了分析.

首先,通过上述模型和算法得到一组调度概率,然后在四种不同任务到达过程下分析基于以上调度概率的任务平均响应时间. 对于四种不同的任务到达过程,假设其任务到达时间间隔分布服从指数分布、常数、均匀和卡方分布,其期望均是任务到达率的倒数 1/λ. 其中常数分布指连续任务到达时间间隔是常量 1/λ. 假设本实验模拟的分布式系统具有 3 个处理单元,其服务率分别是 50,35,40. 任务的响应时间包括等待时间和执行时间. 实验中分析了每 100 个任务的平均响应时间,结果如图 3 所示. 指数分布的平均响应时间最低,因为指数分布的模型和算法的假设前提相吻合,而卡方分布的平均响应时间最长. 由此可见,当实际与假设条件发生偏离时,基于非线性规划的调度算法的性能会下降,并且随着到达率的增加,性能下降更快.

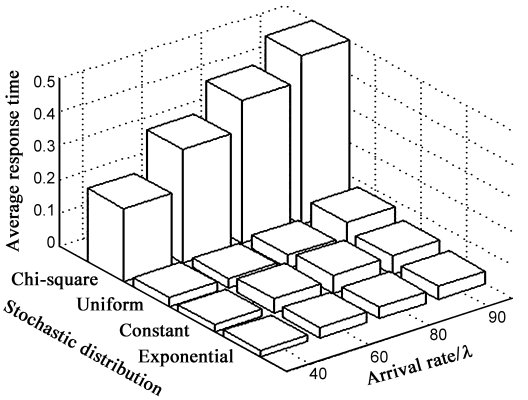


图3　各种随机到达过程下的响应时间

任务到达过程存在不确定性,在网络分布式计算系统中,任务的执行时间通常不确定. 接下来,将分析当状态估计不正确时,算法的鲁棒性. 实验中仍然假设包含 3 个处理单元的分布式系统,其真正的服务率分别是 50,35,40. 对到达率是 60 和 80 进行了两组实验,当处理单元的状态,这里即服务率,估计发生错误时,响应时间变化如图 4 所示. *X* 坐标轴的四个点分别代表服务率估计正确,低估 5 个单位,高估 5 个单位和高估 10 个单位. 从图中可以看出,无论何种估计错误发生,平均响应时间都发生增长. 且在实验中,低估相比于同幅度的高估产生更差的结果.

在本小节分析了当任务到达过程、处理单元状态估计发生错误时,提出的基于非线性规划算法的鲁棒

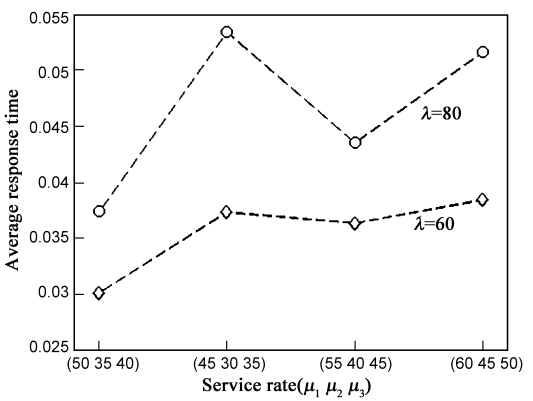


图4　服务率估计错误时的响应时间

性,实验结果表明该方法对调度的参数较为敏感. 针对该问题,下一小节给出了一个基于学习的调度算法,它能自适应这些不确定性,且不需要提前预测,在第 5 节将对这两个算法的有效性进行分析.

4.2 基于强化学习的算法

从上述分析可知,基于非线性规划的算法具有鲁棒性方面的不足. 此外,在排队模型式(1)中,对所有任务,使用平均服务率,这样的简化抽象实质上将所有任务认为是同样大小,这种与实际不相符的假设会给性能带来一定影响. 为了消除以上问题,本小节提出了一个在线学习算法,该算法能学习任务到达模式以及适应不同任务大小的服务率.

4.2.1 调度问题 MDP 定义

如前所述,对于多用户网络应用调度问题,任务随机到达调度器. 必须等所有任务到达或者预知任务到达过程,否则无法找到最优的分配方案. 即,当前任务的分配取决于后续到达任务的分配. 当前任务最短响应时间的分配可能会延长后续任务的响应时间,从而分布式系统的平均响应时间增大. 基于以上观点,多用户网络应用的调度成为一个动态规划问题,一个分配方案的好坏必须要待所有任务全部分配完才能知晓.

前面的分配对后继的分配具有一定的影响. 而当前任务的分配决策仅和当前的任务类型有关,只需知道任务类型,便可知该任务可在哪些单元上处理. 实质上,每一个分配可看作是一个决策制订阶段. 因此,一个完整的调度方案包含多个这样的阶段. 在这样的描述下,调度成为一个 Markov 决策过程(MDP)[23]. 下文将用四元组 $<S,B,\Gamma,R>$ 将调度问题定义为一个 MDP 问题.

(1) S 是状态集,相当于任务类型集,$S=\{t_1,t_2,\cdots,t_m\}$;

(2) B 是分配行为集,其由任务类型和处理单元对组成,$B=\{<t_i;PU_j>\mid t_i\in S;1\leqslant j\leqslant n\}$;

(3) Γ 是状态转换函数:$S\times B\rightarrow PD(S)$,其中 $PD(S)$ 是当前状态为 s 时,选择行为 b 后转移到新状态的概率分布;

(4) R 是报酬函数:$S\times B\rightarrow R$,其中 $r\in R$ 表示在状态 s 选择行为 b 时的立即回报. 回到本问题,R 是响应时间的函数.

将调度作为一个动态规划问题,其目标函数是折扣累积回报,定义如下:

$$\psi=r_\theta+\gamma r_{\theta+1}+\gamma^2 r_{\theta+2}+\cdots=\sum_{i=0}^{\infty}\gamma^i r_{\theta+i} \tag{7}$$

其中 γ 是折扣因子. 式(7)中,期望回报的计算只有当所有任务完成一系列的分配行为后才能获得,当前行为的效用取决于未来的行为. 因此,折扣累积回报 ψ 依赖于状态转移函数 Γ,即任务到达过程. 然而,提前获取有关到达任务的知识是不现实的. 假设任务到达是一个泊松过程也将带来一定的负面影响,如 4.1.1 节所分析. 下一小节将使用一种模型无关的强化学习方法——Q 学习,用来解决以上问题.

4.2.2 自适应算法

调度器在进行分配决策时,需要考虑两个因素:一是各处理单元的负载情况,它决定了任务的执行开销,即行为的立即回报;二是任务到达过程的预测. 因为任务随机出现,尽管当前立即回报很高,但长远来看,系统的平均回报率并不一定如此. 通过获得到达过程信息使得系统长期回报最大化,Q 学习正是一种能够在线学习到达过程,并逐渐适应这两个参数的方法.

根据 Q 学习算法,如果类型为 t_i 的任务到达调度器,行为 $b_{ij}=<t_i,PU_j>\in B$ 的期望回报为:

$$Q(t_i,b_{ij})=(1-\alpha)Q(t_i,b_{ij})+\alpha[R(t_i,b_{ij})+\gamma V(t_k)] \tag{8}$$

其中 $\alpha(0<\alpha\leqslant 1)$ 是学习速率,$V(t_k)$ 是下一个转移状态 t_k 的期望回报. $V(t_k)$ 按照式 $\max_j Q(t_k,b_{kj})$ 进行更新. 立即回报 $R(t_i,b_{ij})$ 是任务 t_i 到达至其在 PU_j 上完成所花费时间的倒数. 根据强化学习理论,学习过程试图最大化 ψ. 此外,Q 学习是一种试错型方法,在算法 1 中使用轮盘赌的方法根据概率 Pr_{ij} 进行行为选择,Pr_{ij} 可以在执行完一个任务时按照下式进行更新:

$$\mathrm{Pr}_{ij}=\frac{Q(t_i,b_{ij})}{\sum_j Q(t_i,b_{ij})} \tag{9}$$

Pr_{ij} 将收敛到一个稳定值.

算法 1 基于强化学习的算法

(1) 初始化
FOR any $t_i\in S$, any $b_{ij}\in B$
$Q(t_i,b_{ij})=0$
FOR any $t_i\in S$
$\mathrm{Pr}_{ij}=\frac{1}{n},1\leqslant j\leqslant n$

(2) 类型为 t_i 的任务分配
IF Explore with probability P_e THEN
Select action b_{ij} randomly by uniform distribution
ELSE
Select action b_{ij} by policy Pr_{ij} using roulette

(3) 更新
Update (Q):
$Q(t_i,b_{ij})=(1-\alpha)Q(t_i,b_{ij})+\alpha[R(t_i,b_{ij})+\gamma V(t_k)]$
Update (V): $V(t_i)=\max_j Q(t_i,b_{ij})$

(4) 学习
FOR all PU_j
$\mathrm{Pr}_{ij}=\frac{Q(t_i,b_{ij})}{\sum_j Q(t_i,b_{ij})}$

(5) 下一次迭代
GOTO (2)

上述算法是一个在线学习算法,当任务被分配并执行完后进行分配概率的学习.

5　实验与结果分析

本节将对提出的两个调度算法进行实验验证. 实验中,任务到达服从随机分布,其服务时间假设服从指数分布. 基于学习的算法的有关参数设置如下:

(1)探索率 $P_e = 0.2$:任务分配以 0.2 的概率进行随机分配,否则将按照分配概率进行分配. 探索机制用于避免局部最优. 该值越大,则收敛越慢;

(2)折扣因子 $\gamma = 0.9$:累积折扣回报表达式的系数;

(3)学习速率 $\alpha = 0.3$:该参数同样对收敛速度也有影响. 它也可以设置为一个单调递减的函数.

另外,在服务率的设置上有所不同. 在排队模型中,将所有任务的平均服务率作为处理单元的服务率,而通常任务具有不同的大小,所以在某处理单元上任务完成所耗费的时间有一个较大变化范围,因此用单一值定义服务率的抽象能力有限. 在基于学习的算法中,为每一个处理单元上各任务均定义相应的服务率,在收敛性实验中的服务率矩阵如表 2 所示,为了便于比较,各处理单元的平均服务率保持和 4.1.1 节中的一致,其它参数保持不变.

表 2　服务率矩阵

Unit	t_1	t_2	t_3	t_4	t_5
PU_1	60	50	40	45	55
PU_2	30	35	30	40	40
PU_3	15	55	50	35	45

本节所有的实验都是基于 PC 进行的模拟仿真,在将来工作中考虑将这些实验移植到真实集群系统上进行.

图 5 给出了收敛性实验的结果,每一种颜色所占的比例代表了将类型为 t_1 的任务分配到三个处理单元的概率. 随着学习的进行,三个分配概率值震荡减小,趋于稳定. 此外,在前几百个迭代周期内,任务以较大概率分配给 PU_1. 造成这一现象的原因是:在初始时,处理单元空闲,而 PU_1 具有最大的处理能力,随着任务不断到达,PU_1 上发生阻塞,这时任务开始向具有次大处理能力的 PU_2 转移.

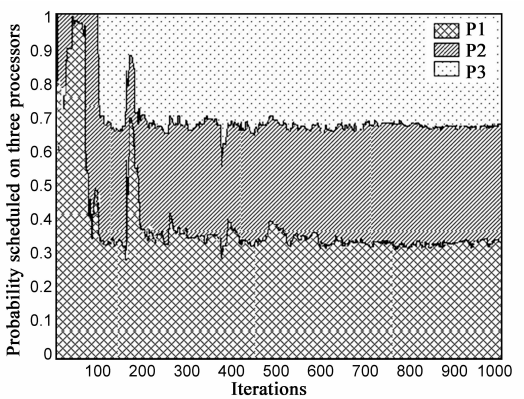


图5　基于学习算法的收敛性

本文下一步工将对基于 Q 学习算法的鲁棒性进行分析. 实验设置参照 4.1.1 节,对四种不同随机分布及四种不同到达率下的平均响应时间进行了记录. 图 6 给出了这组实验的结果,其中蓝色代表基于 Q 学习的算法,而红色代表基于规划的算法. 从图 6 可以得出两个结论:第一,与图 4 中的结果比较,基于规划算法的平均响应时间更长,其原因可能是因为本实验中对不同任务类型的服务率进行了区分,而基于规划的算法将所有任务视为同样大小. 同样的原因,对于指数分布的任务到达间隔,基于学习的算法也表现出优于基于规划算法的性能. 这一原因从表 3 可以得到更好的理解,表 3 给出了两个算法在到达率 $\lambda = 80$ 的分配概率,在实验中各种任务类型出现的概率依次为 0.4,0.2,0.1,0.1. 对于基于规划的算法,当不同类型的任务以相同概率出现,则其分配概率也相同;而由于基于学习的算法考虑了各处理单元上任务大小的影响,分配概率相互之间存在差异. 第二,对于不同的任务到达分布,基于学习算法的平均响应时间相对平稳,相比而言,基于规划算法的性能在非指数分布下降,这与 4.1.1 节的实验结果一致. 由此可见,基于学习的调度算法对任务到达过程具有更强的适应能力.

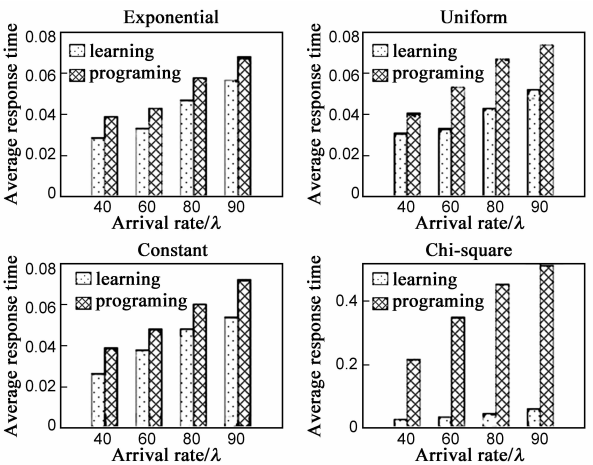


图6　基于学习调度算法鲁棒性分析

表 3　本文两个算法的分配概率示例

	Pr_{ij}	t_1	t_2	t_3	t_4	t_5
PU_1	Learning	0.3587	0.3541	0.3403	0.3327	0.3484
	Programming	0.4934	0.4134	0.3733	0.3733	0.4134
PU_2	Learning	0.3215	0.3253	0.3244	0.3346	0.3281
	Programming	0.2052	0.2693	0.3013	0.3013	0.2693
PU_3	Learning	0.3198	0.3207	0.3353	0.3327	0.3235
	Programming	0.3014	0.3174	0.3254	0.3251	0.3174

下面的实验将本文提出的算法与其它经典调度算法进行了比较. 动态调度有两种模式:一种称为批模式,

在该模式下,当一批任务到达后才启动调度机制,Min-Min,Max-Min,和 Suffrage 是这种模式三个经典算法. 对于 Min-Min 算法,一批任务的最早完成时间(ECT)均被计算,然后总是挑选具有最小 ECT 的映射作为当前的一次调度,直至所有任务调度完. 而对于 Max-Min 算法,则每次挑选具有最大 ECT 的映射. Suffrage 算法则是挑选具有最大损失值的映射. 一般,损失值可定义为任务最大和次大 ECT 之间的差值. 批的大小可以以任务的数量或者时间周期作为划分标准. 显然,批模式下只有当达到批的规模,任务才开始进行调度,先到达的任务不得不等待,这将造成响应时间的增加. 此外,由于需要计算所有任务处理单元对的 ECT,这类算法的时间复杂度渐近于批大小与处理单元数目之积. 基于上述的原因使得批模式算法不能满足实时性的需求,因此在线模式被提出. 这类算法立刻对到达的任务根据最小执行时间(MET)或者 ECT 进行分配,MET 或者 ECT 越小,则越可能分配给该处理单元. 所有这些启发式算法试图减少响应时间,但忽略了负载均衡问题,从而可能使得平均响应时间增加. 本文提出的两个概率型调度算法属于在线模式,这组实验将对本文算法,以及 Min-Min,Max-Min,Suffrage 和一个基于 ECT 的简单在线算法的性能展开分析.

本文中,平均响应时间被用来作为评估调度算法的主要指标,然而,在动态环境中,随机性导致了平均响应时间振荡不稳定的现象. 在任何任务到达过程下人们总是希望调度算法总是能够产生一个较好的调度方案. 如果将平均响应时间看作是一个随机变量,期望平均响应时间的方差能够尽量小. 因此,在动态调度中,方差也是一个重要的评价指标. 出于这样的考虑,在进行性能分析时,不仅考虑平均响应时间,也对其方差变化进行了分析.

本文提出的基于规划的算法有两个约束:任务到达时间间隔呈指数分布,不区分各任务类型的规模大小. 图 7、8 检验了这两个约束条件满足和不满足两个条件下算法的性能. 图 7 是假设任务到达服从指数分布且任务大小相同,本文两个算法与其它四个经典算法性能比较结果. 假设中任务大小相同意味着对于所有的任务三个处理单元的服务率分别是 50,35,40. 因为满足了基于规划算法的假设,所以其产生了最优的分配方案,任务响应时间最快. 而基于学习的算法找到的是一个次优解,因此图 7 中基于学习的算法性能低于基于规划的算法. 图中,本文算法和 ECT 算法的平均响应时间随着任务到达率增加而增加,而三个批模式算法趋势相反,其原因是批模式算法只有当一批任务全部到达才进行调度. 当到达任务比较稀疏时,这一现象更严重,因为更多的时间用来等待后续任务的到达. 这一现象在图 8 中同样存在. 图 8 是当任务到达为一般分布且大小不一致时的比较结果,使用不同的任务到达率(见表 2)来表示任务大小的不同. 图 8 中,正如所预料的,当上述条件不满足时,基于规划的算法得到的不是最优解,因此性能下滑,但基于学习的算法仍然表现稳定. 与图 7 相比,各算法的响应时间有所增加,这是因为一些较大任务的服务率缩小(见本节开始部分的表 2). 无论是图 7 还是图 8,本文算法相比于四种经典算法都表现出较好的性能. 另外,不同于前人工作中的性能比较,对平均响应时间的方差同样进行了分析. 任务到达是随机的,因此平均响应时间是一个随机变量. 从图中可以看出基于学习的算法方差较小,性能优于其它算法. 当与前提假设发生偏离时,基于规划算法的方差出现放大. 在实验中,还发现 Min-Min,Max-Min 和 Suffrage 算法具有非常相似的性能,这可能是由于这三个算法需要到达任务呈现某种形式才能体现出各自的优势,例如,如果规模较大的任务不常见时,这时 Max-Min 算法性能可能会有所提升.

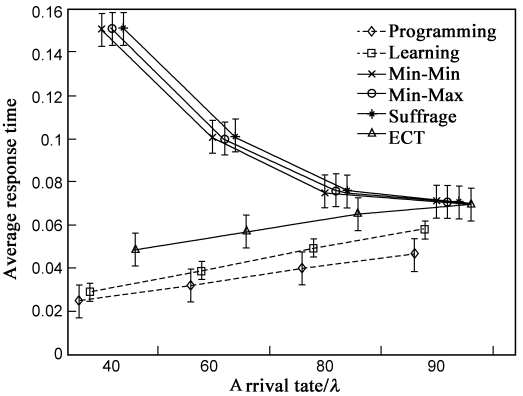


图7 任务按指数分布到达且大小相同时的性能比较

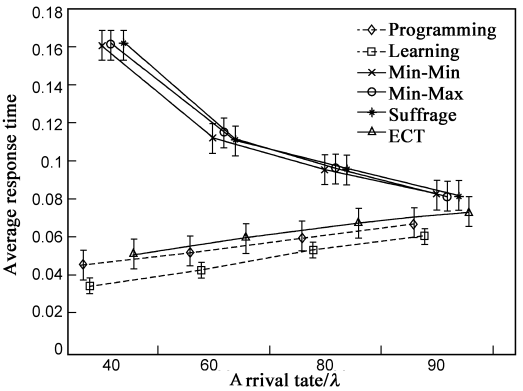


图8 任务按一般随机到达且大小不同时的性能比较

前文提到概率型调度能够有效缓解状态估计不准确的问题,图 9 验证了此结论. 该实验的参数设置与图 8 的实验一致,第一列代表状态估计正确时的平均响应时间,从左至右各列依次代表任务完成时间出现低估

或者高估时的结果. 低估和高估的效果通过调整服务率来达到,比如,↓5 表示每一任务类型的估计服务率下降 5 个单位. 从图 9 可以看出本文提出的算法振荡幅度较小,因此本文的概率型调度算法对状态估计准确度的敏感性较低,从而对状态估计的约束相对宽松.

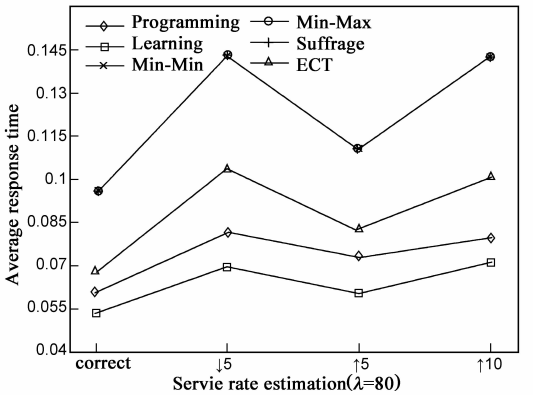


图9　状态估计错误时敏感性分析

6　总结

本文研究了分布式计算中多用户网络应用的任务调度问题,通过对其动态性和不确定性的深入分析,提出了两个概率型调度算法:一是在排队模型下的基于非线性规划算法;一是基于强化学习的算法. 实现了一种快速有效的调度方法. 第一个算法在满足前提条件下能够得到最优的分配概率,第二个算法则具有更强的适应性,能适用各种动态环境.

参考文献

[1] Jiang Y C, Jiang J C. Contextual resource negotiation-based task allocation and load balancing in complex software systems[J]. IEEE Trans Parallel and Distributed Systems, 2009, 20(5): 641 – 653.

[2] Gary M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness[M], New York, NY, USA: W H Freeman and Co, 1979.

[3] Ullman J D. NP-complete scheduling problems[J]. J Computer and Systems Sciences, 1975, 10: 384 – 393.

[4] Gkoutioudi K Z, Karatza H D. Multi-criteria job scheduling in grid using an accelerated genetic algorithm[J]. J Grid Computing, 2012, 10(2): 311 – 323.

[5] Omara F A, Arafa M M. Genetic algorithms for task scheduling problem[J]. J Parallel and Distributed Computing, 2010, 70(1): 13 – 22.

[6] Hu X S, Dick R P. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs[J]. IEEE Trans Very Large Scale Integration Systems, 2011, 19(10): 1884 – 1897.

[7] Chakrabarti P P, Kumar R. Online scheduling of dynamic task graphs with communication and contention for multiprocessors[J]. IEEE Trans on Parallel and Distributed Systems, 2012, 23(1): 138 – 153.

[8] Wen Y, Xu H, Yang J D. A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system[J]. Information Sciences, 2011, 181(3): 567 – 581.

[9] Sinnen O, To A, Kaur M. Contention-aware scheduling with task duplication[J]. J Parallel and Distributed Computing, 2011, 71(1): 77 – 86.

[10] Benoit A, Robert Y. Complexity results for throughput and latency optimization of replicated and data-parallel workflows[J]. Algorithmica, 2010, 57(4): 689 – 724.

[11] 艾丽华,罗四维. 数据网格虚拟机动态存储层析的研究[J]. 电子学报,2010,38(11):2680 – 2685.
AI Li-hua, LUO Si-wei. Research on dynamic storage hierarchy of data grid virtual machine[J]. Acta Electronica Sinica, 2010, 38(11): 2680 – 2685. (in Chinese)

[12] Subrate R, Zomaya A Y, Landfeldt B. Game-theoretic approach for load balancing in computational grids[J]. IEEE Trans on Parallel and Distributed Systems, 2008, 19(1): 66 – 76.

[13] Jiang Y C, Huang Z C. The rich get richer: preferential attachment in the task allocation of cooperative networked multiagent systems with resource caching[J]. IEEE Trans on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2012, 24(5): 1040 – 1052.

[14] Aktaruzzaman M. Literature Review and Survey: Resource Discovery in Computational Grids[R]. Technical Report, School of Computer Science, University of Windsor, Windsor, Ontario, Canada, 2003.

[15] 张伟哲,田志宏,张宏莉,等. 虚拟计算环境中的多机群协同调度算法[J]. 软件学报,2007,18(8):2027 – 2037.
Zhang Weizhe, Tian Zhihong, Zhang Hongli, et al. Multi-cluster co-allocation scheduling algorithms in virtual computing environment[J]. Journal of Software, 2007, 18(8): 2027 – 2037. (in Chinese)

[16] 肖鹏,胡志刚. 截止时间约束下独立网格任务的协同调度模型[J]. 电子学报,2011,39(8):1852 – 1857.
Xiao Peng, Hu Zhigang. Co-scheduling model for independent tasks with deadline constraint in computational grid[J]. Acta Electronica Sinica, 2011, 39(8): 1852 – 1857. (in Chinese)

[17] He L, Jarvis S A, Spooner D P, Bacigalupo D, Tan G, Nudd G R. Scheduling DAG-based applications in multi-cluster environments with background workload using task duplication[J]. International Journal of Computer Mathematics, 2010, 87(11): 2387 – 2397.

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

[19] 张理论, 叶红, 吴建平, 等. 基于最大负载偏移率的并行负载平衡性能分析[J]. 计算机研究与发展, 2010, 47(6): 1125 – 1131.
Zhang Lilun, Ye Hong, Wu Jianping. Parallel load-balancing performance analysis based on maximal ratio of load offset[J]. Journal of Computer Research and Development, 2010, 47(6): 1125 – 1131. (in Chinese)

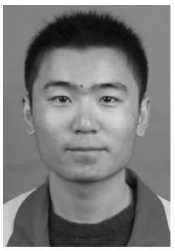
[20] Qin X, Xie T. An availability-aware task scheduling strategy for heterogeneous systems[J]. IEEE Trans on Computers, 2008, 7(57): 188 – 199.

[21] Donald Gross, John F Shortle, James M Thompson, Carl M Harris. Fundamentals of Queueing Theory[M]. Hoboken, NJ: John Wiley & Sons, 2008.

[22] Kaelbling L P, Littman M L, Moore A W. Reinforcement learning: a survey[J]. Journal of Artificial Intelligence Research, 1996, 4: 237 – 285.

[23] 吴小东, 韩建军, 王夭江. 一种基于 VFD 多核系统的硬实时任务节能调度算法[J]. 计算机研究与发展, 2012, 49(5): 1018 – 1027.
Wu Xiaodong, Han Jianjun, Wang Tianjiang. Energy-aware scheduling of hard real-time tasks in VFD-based multi-core systems[J]. Journal of Computer Research and Development, 2012, 49(5): 1018 – 1027. (in Chinese)

作者简介



童 钊 男, 1985 年 2 月生, 湖南岳阳人. 2007 年获得北京理工大学计算机科学专业学士学位, 2014 年获得湖南大学工学博士学位, 现为湖南师范大学数学与计算机科学学院讲师. 主要研究方向为并行与分布式计算、资源管理、任务调度、并行算法设计.
E-mail: tongzhao@hunnu.edu.cn

肖 正 男, 1981 年 4 月生, 湖南怀化人. 2003 年本科毕业于湖南大学通信工程专业, 2009 年获得复旦大学计算机应用技术专业博士学位, 现为湖南大学信息科学与工程学院助理教授. 主要研究方向为并行与分布式计算, 分布式人工智能, 协同计算.
E-mail: zxiao@hnu.edu.cn

李肯立 男, 1971 年 10 月生, 湖南娄底人. 2000 年获得中南大学数学专业硕士学位, 2003 年获得华中科技大学计算机科学博士学位, 现为湖南大学信息科学与工程学院教授. 2004 年至 2005 年在伊利诺伊大学厄巴纳-香槟分校任访问学者. 2013 年获得教育部科技进步二等奖. CCF 高级会员. 主要研究方向为并行计算, 网格计算, 云计算, DNA 计算. E-mail: lkl510@263.net