第44卷第7期 中国科学技术大学学报 Vol. 44, No. 7
2014年7月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jul. 2014

文章编号:0253-2778(2014)07-0590-09

一种新型的云任务调度算法研究

周发超[1],王志坚[1,2],叶 枫[1,2]

(1.河海大学计算机与信息学院,江苏南京 211100;2.南京航空航天大学计算机科学与技术学院,江苏南京 210016)

摘要:云计算具有弹性、保证服务质量和按需的资源配置模型等特征,通常用于处理大批量的计算任务,因此任务调度策略对资源使用效率起着至关重要的作用.考虑到任务的数量和到达服务器的时间不确定性,并且用户对任务的执行往往有一定的期望(如任务优先级、执行时间等),如何合理地分配计算资源,最大程度满足用户的服务质量需求是一个值得研究的问题.为此,提出了一种新型的云环境下QoS-aware服务质量感知的任务调度算法(QTS),该算法结合贪心算法的思想,并加入了任务完成满意度模型作为任务调度的评价依据.通过扩展CloudSim仿真平台进行实验,将QTS与RR调度、Max-Min和Min-Min调度比较,结果表明,QTS是一种有效的任务调度算法.

关键词:云计算;任务调度;服务质量感知;CloudSim

中图分类号:TP311 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2014.07.008

引用格式:Zhou Fachao, Wang Zhijian, Ye Feng. Research of a novel cloud task scheduling algorithm[J]. Journal of University of Science and Technology of China, 2014,44(7):590-598.
周发超,王志坚,叶枫. 一种新型的云任务调度算法研究[J]. 中国科学技术大学学报,2014,44(7):590-598.

Research of a novel cloud task scheduling algorithm

ZHOU Fachao[1], WANG Zhijian[1,2], YE Feng[1,2]

(1. *College of Computer and Information, Hohai University, Nanjing* 211100, *China*;
2. *College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016, *China*)

Abstract: With its flexibility, guaranteed quality of service and on-demand features such as resource allocation model, cloud computing is often used to handle large computing tasks, so efficient task scheduling strategies of cloud computing play a vital role. Given the uncertainty in the number of tasks and the time of arrival at the server, and the fact that, users tend to have certain expectations (such as task priority, execution time, etc.) for the implementation of the tasks, reasonable allocation of computing resources for task scheduling to satisfy users' QoS requirements is of great importance. A novel QoS-aware task scheduling mechanism (QTS) was proposed, this scheduling mechanism can best meet the user's QoS requirements. By comparing QTS with RR, Max-Min and Min-Min scheduling policies by CloudSim simulation, it was found that QTS is a more effective task scheduling mechanism.

Key words: cloud computing; task scheduling; QoS-aware; CloudSim

收稿日期:2014-03-21;**修回日期**:2014-06-15

基金项目:中国高技术研究发展(863)计划(2008505611),江苏省水利科技项目(2013025),河海大学中央高校基本科研项目(2009B21614)资助.

作者简介:周发超,男,1988年生,硕士生. 研究方向:云计算、数据挖掘. E-mail: 790428547@qq.com

通讯作者:王志坚,博士/教授. E-mail: zhjwang@hhu.edu.cn

0　引言

云计算[1-2]是一种能够将动态可伸缩的虚拟化资源通过 Internet 以服务的方式按需提供给用户使用的计算模式，是分布式计算、并行计算和网格计算等的跃升发展. 云计算具有弹性、有保证的服务质量、按需的资源配置模型(resource provisioning model)等特征，其提供的服务主要可以分为基础设施及服务(IaaS)、平台及服务(PaaS)以及软件及服务(SaaS)三类. 用户可以低代价获取云提供商提供的高质量服务，从而避免购买昂贵的硬件设备和维护成本.

当前，对云计算的研究主要集中在虚拟化、大数据处理、资源管理、任务调度和云安全等方面. 其中，任务调度的研究重点是如何在计算任务数量庞大的情况下，将任务合理地调度到合适的资源上执行. 任务调度算法的优劣直接影响到云计算资源的使用效率及用户的评价. 当前，已有的研究[3-16]分别从资源利用率、最大完工时间、负载均衡、执行费用及满足用户 QoS 等角度开展了研究，但是所提出的算法实现都是“静态”的，即在执行任务调度之前，所有任务均已到达云端服务器，而实际的情况则是用户所要执行的任务未必是一次性提交到云端的，而是呈现一种“不定时、多批次、任务数目不等，任务长度不一”的动态特点. 很显然，当前在任务调度上的研究均未能考虑到这一实际情况. 基于此，本文提出了一种新型的云环境下 QoS-aware 任务调度算法，该调度算法能够在任务动态到达云端情况下发挥出良好的调度效果，并通过扩展 CloudSim[17]开源仿真平台对本文所提调度算法进行了仿真对比验证.

1　相关工作

分布式计算环境中，有大量的实现算法应用于将任务映射和调度到相关的计算资源上去执行，在此基础上，很多云计算中的任务调度算法也被陆续提了出来. 从优化任务的完工时间角度去考虑，有研究[3-9]在传统算法上加以改进，在任务的完工时间上作不同程度的优化. 如文献[3]给出了一种改进的 Min-Min 任务调度算法，该算法执行任务调度能一定程度减少任务最大完工时间；Guo 等[4]设计的一种“双适应度的遗传算法”，其基于传统的遗传算法，以任务的最大完工时间最短作为调度策略的优化目标，同时将任务的平均执行时间作为算法优劣的评价标准，可以达到较好的任务调度结果；Li 等[5]提出了一种基于蚁群优化的任务调度算法，算法以负载均衡作为调度依据，以任务的最大完工时间作为衡量资源处理任务的标准，其产生的调度在缩短最大完工时间上有较好的表现；史少峰等[6]研究了一种基于动态规划的云计算任务调度策略，该算法以运行时间最短作为优化目标，最后通过与 Max-Min，Min-Min 调度算法相比，在一定数量规模下，任务完成时间会减少. 虽然这些算法比起传统的算法都能达到更优的结果，但是算法均未考虑任务的动态性，对任务的多批次、规模波动情况也没有考虑，特别是极端的两种情况下：规模很小或者规模过大，所以算法难免有局限性.

文献[5-6，10-11]考虑了服务器的负载均衡，对任务的调度进行优化. 例如，Wang 等[10]提出了一种基于 CPU，Memory 和 VMs 的多维任务调度算法，该算法可以根据任务需求和资源负载进行任务的分配，但是并未考虑资源请求的动态性；文献[11]提出了一种基于负载均衡的双级别任务调度算法，算法的一级调度考虑任务到虚拟机的调度，二级调度实现虚拟机到主机的调度，实验结果表明，该算法在资源负载上有更优的表现，但是该算法没有考虑任务的动态性，也没有考虑单个任务长度变化很大的问题.

同样，从用户 QoS 请求角度，也有文献给出了相应的算法研究，如文献[12-14]. Jung 等提出的一种基于用户 QoS 的 CloudSim 改进模型，考虑了用户对任务调度优先级的 QoS 要求，产生的调度结果可以很好地满足用户的请求，但是考虑的任务规模较小，而且没考虑多批次任务不定时到达云端情况下，后来批次中高优先级的任务对先于该批次到达的任务中低优先级任务的影响，虽然有很好的调度效果，却由于考虑不足难免有局限性. 从资源利用率、任务执行成本考虑，也已有许多研究[9-12，15-16]. 特别地，文献[16]给出了一种基于带宽感知的任务调度算法，它将网络的带宽限制考虑到任务的调度中去，算法产生的调度结果与不考虑带宽限制的算法相比能够表现出较好的性能，但是同样未考虑任务的动态性特点；朱宗斌等[9]考虑时间和成本的约束实现了一种基于改进的 GA 云计算任务调度算法，此算法产生的调度结果可以使得执行任务产生的成本较小，但并没有考虑任务规模变化大和多批次任务不定时到达云端的情况.

上述算法在执行任务调度上虽各有侧重,但实现都是静态的,与实际的云计算中任务调度环境存在很大差别,不符合实际情况下任务"多批次、不定时、规模不一、大小不等"的动态特点,而且以上算法都没有考虑任务的多核并发处理特性.基于现有算法的局限性,本文充分考虑任务的动态特点,并且加入用户对任务优先级及用户期望的任务完成时间两项 QoS 要求,提出了一种新型的 QoS-aware 任务调度算法,通过与 RR、Max-Min 和 Min-Min 等调度策略相比较,验证了本文所提算法在任务动态环境下的有效性.

2 模型建立

与现有研究中提到的任务调度算法不同,本文提出的任务调度算法考虑了任务的动态特征,同时加入了用户对所提交任务的期望完成时间和期望任务执行优先级两项指标,即本文所希望解决的调度问题是如何能尽最大可能在用户期望的完成时间内完成任务的执行以及当用户任务到达云端如何按照用户期望的执行优先级进行调度.

为了简化真实世界的复杂性,建立一个解决问题更加有效的模型,本文作如下假设:①任务之间是独立运行的,不存在相互依赖性;②任务在虚拟机中的执行时间是可以根据 CPU 执行速度度量的;③高优先级的任务不能抢占当前运行的低优先级任务所占资源.基于以上设定,将对考虑的调度指标分别建模.为了模型描述的方便,做如下定义:

定义 2.1 task_{ij} 表示第 i 批次到达云端服务器的任务序列中的第 j 个任务,其中 $i,j>0$.第 i 批次到达的所有任务用集合 Task_i 表示:

$$\text{Task}_i=\{\text{task}_{ij}\mid j>0\} \tag{1}$$

定义 2.2 vm_k 表示某一用户使用的虚拟机集合中的第 k 个虚拟机,其中 $k\geqslant 0$.用 VM 表示该用户拥有的所有虚拟机.

$$\text{VM}=\{\text{vm}_k\mid k\geqslant 0\} \tag{2}$$

Task_i 到达 vm_k 将进入任务队列等待被调度执行,执行任务调度操作的机制称为任务调度器,定义如下.

定义 2.3 任务调度器:负责调度 Task_i 中的任务到 VM 中执行.

任务调度器负责定时收集虚拟机负载信息、统计每一台虚拟机中任务的执行情况和根据具体算法机制调度任务到虚拟机中执行.

为了方便描述模型,对任务和虚拟机的相关参数给出如下定义:

定义 2.4 task_{ij} 的指令长度为 l_{ij},任务大小为 s_{ij},并行处理需要的 CPU 核数为 n_{ij}^{task}.

定义 2.5 mips_k 表示 vm_k 中的一个 PE 每秒钟所能处理的信息条数,mbps_k 表示 vm_k 的带宽.

定义 2.6 vm_k 中所有 CPU 核数为 n_k^{vm},$n_k^{\text{vm}}>0$.

这样,任务调度的延迟时间和任务的执行时间可分别定义如下.

定义 2.7 task_{ij} 被调度器调度到 vm_k 中任务队列的过程产生的传输时间延迟定义为 delay_{ij}^k,则 delay_{ij}^k 表示为:

$$\text{delay}_{ij}^k=\frac{s_{ij}}{\text{mbps}_k} \tag{3}$$

定义 2.8 task_{ij} 被调度到 vm_k 中运行所需要的执行时间为 t_{ij}^k,t_{ij}^k 表示为:

$$t_{ij}^k=\begin{cases}\dfrac{l_{ij}}{\text{mips}_k}, & n_{ij}^{\text{task}}\leqslant n_k^{\text{vm}}\\ +\infty, & n_{ij}^{\text{task}}>n_k^{\text{vm}}\end{cases} \tag{4}$$

式(4)表示,如果 task_{ij} 需要的 CPU 核数不大于 vm_k 拥有的 CPU 核数,由于任务是独占资源的,所以运行时间是可以预估出来的,但是如果 task_{ij} 需要的 CPU 核数大于 vm_k 拥有的 CPU 核数,若该任务被不合理地调度到 vm_k 中执行将陷入等待状态,永远得不到执行,也即执行时间趋于正无穷大.

2.1 任务期望完成时间模型

在任务执行之前,用户会对任务的最后完成时间有一个期望值,也即希望任务在自定义的完成时间内完成,定义为用户期望完成时间.

定义 2.9 f_{ij}^{exp} 表示 task_{ij} 的用户期望完成时间.

在实际的执行过程中,任务的完成时间并不一定等于用户设定的期望完成时间,引入变量 f_{ij}^{fact} 表示 task_{ij} 的实际完成时间.

定义 2.10 f_{ij}^{fact} 表示 task_{ij} 的实际完成时间.

为了描述用户对期望完成时间的满意程度,需要引入一个度量函数,为此给出如下定义:

定义 2.11 期望完成时间满意度:某个任务的期望完成时间与实际完成时间之差,引入符号 Sat_{ij} 表示对 task_{ij} 的满意度描述,可形式化表示为:

$$\text{Sat}_{ij}=\begin{cases}1, & f_{ij}^{\text{exp}}\geqslant f_{ij}^{\text{fact}}\\ 1-\dfrac{f_{ij}^{\text{fact}}-f_{ij}^{\text{exp}}}{f_{ij}^{\text{fact}}}, & f_{ij}^{\text{exp}}<f_{ij}^{\text{fact}}\end{cases} \tag{5}$$

式(6)表示,当某个任务的完成时间小于期望时间时,满意度为 1,而当任务实际完成时间大于用户期望时间,则满意度随二者间差值的变大逐渐降低,并趋近于 0.

在真实环境下,调度任务 task_{ij} 到 vm_k 中执行之前,无法确定任务的准确完成时间,为最大限度满足用户对任务期望完成时间的 QoS 请求,需要建立一种预测机制,即通过预测任务的完成时间来估计任务调度的期望完成时间满意度,为任务调度算法提供调度依据.

定义 2.12　用变量 Ewait_{ij}^{k} 表示 task_{ij} 被调度到虚拟机 vm_k 中的预测等待时间.

由式(3)和(4)可以得到 task_{ij} 调度到虚拟机 vm_k 中执行的预测完成时间为:

$$\text{Efini}_{ij}^{k} = \text{delay}_{ij}^{k} + \text{Ewait}_{ij}^{k} + t_{ij}^{k} \tag{6}$$

Ewait_{ij}^{k} 的值为虚拟机 vm_k 的任务队列中排在 task_{ij} 之前的所有任务执行完所需要的时间,由于每个任务的执行时间是可预测的,则 Ewait_{ij}^{k} 的值是可预测的.

则由式(5)和(6)可得预测期望时间满意度为:

$$\text{Esat}_{ij}^{k} = \begin{cases} 1, & f_{ij}^{\text{exp}} \geqslant \text{Efini}_{ij}^{k} \\ 1 - \dfrac{\text{Efini}_{ij}^{k} - f_{ij}^{\text{exp}}}{\text{Efini}_{ij}^{k}}, & f_{ij}^{\text{exp}} < \text{Efini}_{ij}^{k} \end{cases} \tag{7}$$

2.2　任务优先级模型

任务优先级代表用户对所提交任务执行顺序的一种期望,在云端主要起到两个作用:决定同时提交的一批任务中哪个任务优先调度;高优先级的任务在调度到具体虚拟机后,将优先于该虚拟机中待执行队列中低优先级的任务执行.

如果提交的任务在执行过程中优先级不发生变化,则很可能出现后来的任务优先级过高从而使先前提交的低优先级的任务得不到执行,使得任务陷入"饥饿"状态.为了保证低优先级的任务也可以得到执行,本文采用动态优先级,即在任务执行的过程中,随时间的推移低优先级的任务会逐渐提高优先级,从而使得自己可以被调度执行.

在该任务优先级模型中,设定任务优先级最高为 1,任务优先级越低,其数值越大,例如,优先级为 1 的任务优先级高于优先级为 2 的任务.对用户为 task_{ij} 所设定的优先级,可做如下定义:

定义 2.13　p_{ij} 表示用户为任务 task_{ij} 所设定的静态优先级.

静态优先级的解释为:用户自定义的任务优先级,并且任务处于等待状态的过程中优先级是不会变化的,但是为了保证低优先级的任务不会"饿死",需要为任务添加一个动态优先级属性.

定义 2.14　$p_{ij}^{k,\Delta t}$ 表示任务 task_{ij} 被分配到虚拟机 vm_k 中,处于等待执行状态的优先级.其中,Δt 为任务在等待队列中的等待时间,$\Delta t \geqslant 0$.

$$p_{ij}^{k,\Delta t} = \begin{cases} p_{ij}, & 0 < \Delta t \leqslant f_{ij}^{\text{exp}} \\ p_{ij} - \left\lfloor \dfrac{\Delta t - f_{ij}^{\text{exp}}}{f_{ij}^{\text{exp}}} \right\rfloor - 1, & f_{ij}^{\text{exp}} < \Delta t \leqslant (p_{ij} - 1) \times f_{ij}^{\text{exp}} \\ 1, & \Delta t > (p_{ij} - 1) \times f_{ij}^{\text{exp}} \end{cases} \tag{8}$$

当等待时间 Δt 不大于用户期望完成时间与实际在 vm_k 中执行时间之差时,由于任务的等待时间没有超出任务的期望完成时间范围,所以不需要提高其优先级;当等待时间 Δt 超过了任务期望完成时间,则优先级根据等待时间与期望完成时间的差值 $\Delta t - f_{ij}^{\text{exp}}$ 和期望完成时间 f_{ij}^{exp} 之间的倍数关系提升任务执行的优先级,当等待时间 Δt 大于 $(p_{ij}-1)\times f_{ij}^{\text{exp}}$ 时候,任务的优先级始终为 1,也即最高优先级.

3　任务调度算法

由于任务的动态性,仅采用遗传算法、蚁群算法等启发式算法进行调度难以实现大幅优化的目的.本文通过结合贪心算法的思想,并根据本文所需考虑的调度指标加入了任务完成时间满意度模型作为任务调度的评价依据,实现了一种新型的 QoS-aware 任务调度算法 QTS.

任务调度的抽象工作流程如图 1 所示.在整个调度过程中,用户会不定时地提交一批任务,云端无法事先预知提交时间,同时任务的数量也未知,任务通过网络到达服务器端,然后任务调度器进行这批任务的调度.从用户的角度讲,云服务器端的服务器可以分为三层,最上层为任务调度器,负责定时收集所属用户的任务执行信息,并通过这些反馈信息及用户提交的任务信息进行新一批任务的调度;中间层则为所属用户的虚拟机,这些虚拟机只专属于一个用户;最底层为基础设施——分布式数据中心集群.由图 1 可以看出,一个用户对应一个所属的"虚拟机集群",并对应一个"专属的"任务调度器.

用户的任务到达云端后加入任务调度器的任务调度队列,然后任务调度器执行任务的调度.基于任务的动态性特点,QTS 利用了 Min-Min 算法的思

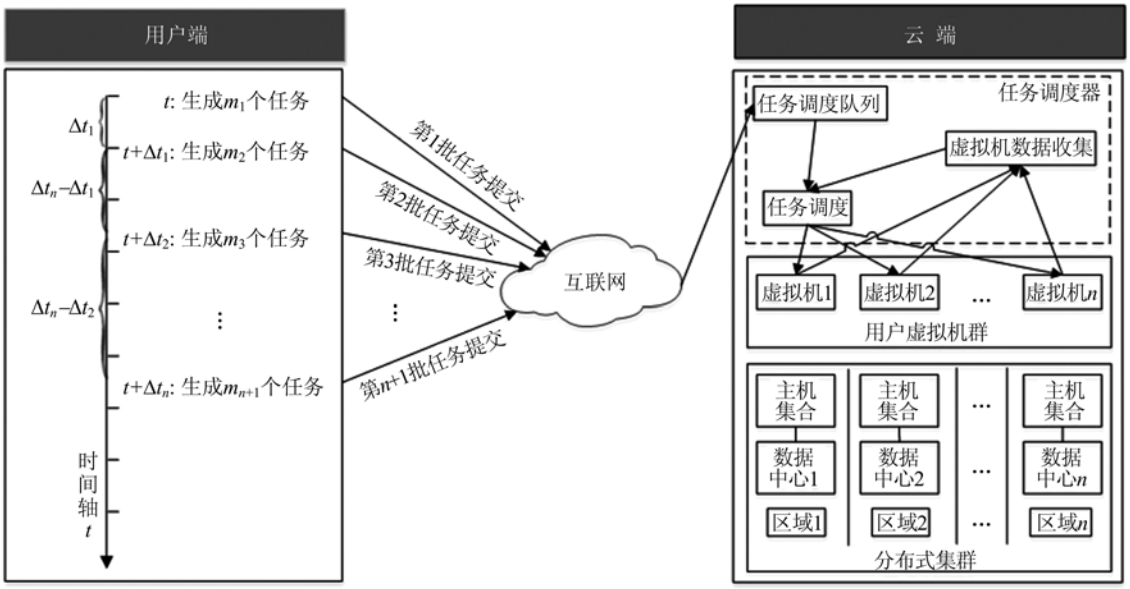

图 1 任务调度抽象工作流图

Fig. 1 Abstract task scheduling workflow diagram

想实现任务的调度,具体的调度过程如下:

虚拟机数据收集模块定期收集所属用户虚拟机的任务执行信息,包括已经执行完成的任务、正在执行的任务以及处于等待状态任务的信息,在按照下述步骤执行任务调度的过程中,最关键的是根据收集的信息计算 Ewait_{ij}^{k} 的值. 为了更好地实施调度,任务调度器根据收集的虚拟机信息为每一台虚拟机 vm_k 创建一个任务执行队列快照,在调度任务时,通过对该快照实施等同于虚拟机工作情况的"演化",从而求得某任务若是在该虚拟机中运行需要的延迟等待时间.

假设在 t_1 时刻任务调度器对 task_{ij} 实施调度,并试图将该任务调度到 vm_k 中,并且在 task_{ij} 之前,已有 x 个任务已经成功被安排进 vm_k 中等待运行,这 x 个任务的传输耗时为 Times. 在 t_1 时刻 vm_k 的任务执行队列演化快照概念如图 2 所示.

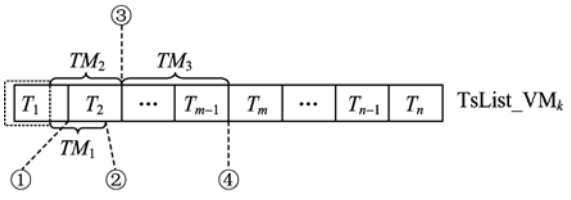

图 2 vm_k 的任务执行队列演化快照

Fig. 2 The task queue evolution snapshot of vm_k

根据式(3),假设 task_{ij} 在 vm_k 中执行的传输延时为 TM_1,此时 vm_k 正在执行 T_1,如图中虚线框中所示. 若不考虑网络传输延时,任务 task_{ij} 被安排执行的位置可能为图 2 中①,②,③和④四个位置. 实际上延时是存在的,这种情况下,由于延迟了 TM_1,当任务进入 vm_k 的执行等待队列时,T_1 已经做完,T_2 也已经执行到图中②的位置,①的位置已经不存在,也即是 task_{ij} 可能的存在位置只有③及其以后的位置. 现假设 task_{ij} 被安排在③处等待执行,则可以得出 $\mathrm{Ewait}_{ij}^{k}=TM_2-TM_1$,由于 $\mathrm{delay}_{ij}^{k}=TM_1$,$t_{ij}^{k}$ 可以根据式(4)得到,那么预测完成时间 $\mathrm{Efini}_{ij}^{k}=\mathrm{delay}_{ij}^{k}+\mathrm{Ewait}_{ij}^{k}+t_{ij}^{k}$ 是可以求得的. 由于在传输之前,传输延时任务 task_{ij} 已经等待了 Times,所以 task_{ij} 若被安排在 vm_k 中运行,其最终的预测完成时间为:$\mathrm{Efini}_{ij}^{k}=\mathrm{delay}_{ij}^{k}+\mathrm{Ewait}_{ij}^{k}+t_{ij}^{k}+\mathrm{Times}$. 为了准确求得 Ewait_{ij}^{k},需要做以下操作:

① 由式(3),根据虚拟机带宽信息和任务自身的大小求得 $\mathrm{delay}_{ij}^{k}=TM_1$;

② 对队列 $\mathrm{TsList\_VM}_k$ 进行演化,使得在安排任务 task_{ij} 时,$\mathrm{TsList\_VM}_k$ 的当前执行位置为图中②指示的地方;

③ 根据 task_{ij} 的优先级和 $\mathrm{TsList\_VM}_k$ 中待执行任务的优先级,对 $\mathrm{TsList\_VM}_k$ 进行演化,找出 task_{ij} 在队列中合适的位置和执行之前需在队列中等待的时间 Ewait_{ij}^{k}.

通过以上三步便可求得 Ewait_{ij}^{k} 和相应的 Efini_{ij}^{k}. 由于虚拟机中的任务执行采用了动态优先级,为了与虚拟机中保持一致,快照的演化同样需要

加入动态优先级特性，在演化过程中，更新动态优先级的时机在每个新任务执行之前. 同样地，可以计算出 $task_{ij}$ 在其他虚拟机中的预测完成时间，然后找出使得 $task_{ij}$ 执行完成满意度最高的虚拟机作为最终的调度目标，并将该任务调度到其中等待运行. 在调度完 $task_{ij}$ 后，不妨假设 $task_{ij}$ 最终被安排到 vm_k 中运行，为不影响后续任务的调度，需要对 vm_k 的任务执行队列快照做如下操作：

① 更新 $TsList\_VM_k$ 总的传输延时：$Times = Times + TM_1$；

② 将 $task_{ij}$ 的基本信息（任务长度，优先级等）封装为任务快照加入到该任务被安排执行的位置，并将快照前段的起始位置截断到图 2 中的②处；

③ 保持其他的虚拟机调度 $task_{ij}$ 之前的快照信息不变.

以上快照演化过程以虚拟机只有一个 PE，任务只需一个 PE 为例. 本文所做的调度算法中，综合考虑了任务需要多核并行处理以及虚拟机有多个 PE 的情况，其演化过程相比于图 2 的情况稍复杂，但是基本原理是一样的，故此处不再说明.

现在假设用户 $User_1$ 拥有 K 台虚拟机，在 t 时刻第 i 批任务 $Task_i$ 到达云端，QTS 任务调度步骤如下：

Step 1　任务调度器将第 i 批任务 $Task_i$ 按照优先级从高到低加入任务调度队列 Q_{tasks}；

Step 2　任务调度模块向虚拟机数据收集模块请求虚拟机执行信息和虚拟机配置信息；

Step 3　虚拟机数据收集模块收到请求后，将立即收集所有虚拟机在 t 时刻的任务执行等待队列信息，以及该虚拟机配置信息，并将收集到的结果发送给任务调度模块；

Step 4　任务调度模块收到数据收集模块返回的信息后，进入下一步骤；

Step 5　任务调度模块从 Q_{tasks} 中取出优先级最高的所有任务放入集合 taskCol；

Step 6　判断集合 taskCol 是否为空，不为空，则循环执行 Step 7～Step 9，否则跳到步骤 Step 10；

Step 7　对于任务集合 taskCol 中的每一个任务 $TempTask_{ij}$，根据 Step 4 收集到的虚拟机相关数据，计算出 $Efini_{ij}^{k}$. 同时计算出任务 $TempTask_{ij}$ 在虚拟机集合 VM 中每一台虚拟机中的预测完成时间，然后根据式(7)计算出相应的预测期望完成时间满意度集合，并从中筛选出满意度最高的 $Esat_{ij}^{k'}$；

Step 8　根据 Step 7 中的结果，找出预测满意度最高的那个任务 $TempTask_{ij'}$ 所对应的虚拟机 $vm_{k'}$；

Step 9　将任务 $TempTask_{ij'}$ 映射到虚拟机 $vm_{k'}$ 上，并将该任务的基本信息加入到任务调度模块后得到的虚拟机任务等待队列中，更新该队列使其按照优先级排列，并将该任务从集合 Q_{tasks} 和集合 taskCol 中移除；返回到 Step 6；

Step 10　判断 Q_{tasks} 是否为空，如果不为空，则跳到步骤 Step 5，否则进入下一步.

Step 11　清除虚拟机数据收集模块返回给任务调度模块的信息，任务调度器工作完成，退出调度程序.

对于虚拟机，当收到任务调度器提交的任务后，将会根据式(8)重新计算任务队列中任务动态优先级，并按照该优先级对任务进行重新排序，并且在每个任务完成执行后，从等待队列中取出任务执行之前，同样需要重新计算任务动态优先级，以确保取出的下一个任务动态优先级最高.

4　结果与评估

由于搭建一个真实的云计算环境是一个很大的系统工程，而且在其上进行真实环境下的实验也将会造成一笔很大的花销，因此使用模拟工具是一个行之有效的方法，CloudSim 是一款开源的、支持无缝建模和仿真的可扩展的仿真框架，能对时间、用电量以及传输开销仿真建模，文献[2,17-18]详细介绍了这个云计算环境下的模拟器. 本文提出的一种新型的 QoS-aware 任务调度机制，将通过扩展 CloudSim 仿真平台进行调度算法的仿真与验证.

为了验证所提算法的有效性，本文对 CloudSim 进行了扩展. 首先，为了模拟任务的不定时到达特性，在 CloudSim 的 DataCloudTags 类中加入了任务动态到达云端事件，当模拟任务到达时，将触发该事件，从而使代理类 DataCenterBroker 依据任务调度算法将任务映射到用户所属虚拟机，并提交；为了模拟任务优先级特性，在任务类 CloudLet 中加入了静态和动态优先级属性，静态优先级由用户创建任务时设定，动态优先级为任务处于等待状态时由系统根据等待时间动态调整；为了描述用户对任务的期望完成时间和任务到达云端的时间，在 CloudLet 类中同时扩展出了“期望完成时间”和“到达云端时间”两种属性，期望完成时间由用户创建任务时指

定,到达云端时间根据任务到达云端的时钟值由仿真平台确定;在任务调度之前,需要得到所属用户的虚拟机信息以及每台虚拟机中执行任务队列和等待队列信息,由于虚拟机为用户所创建,则代理中保存了虚拟机基本信息,不用向数据中心请求,而任务执行队列和等待队列信息则是动态变化的,为了在执行任务调度时依据本文算法执行任务调度,需要动态向数据中心发送请求,为此在 DataCloudTags 中增加了请求虚拟机执行队列和任务队列信息的事件及对应的代理 DataCenterBroker 的响应事件,请求事件触发时机为任务到达代理后,执行算法调度前.同时为了实现与 Max-Min、Min-Min 和轮转调度的对比,本文在代理类中还同时扩展实现了除本文外的另外三种对比算法.

实验所构建的仿真环境参数设置如表 1 所示.

表 1 基本实验参数设置列表

Tab. 1 The basic experimental parameters settings list

名 称	值
虚拟机 PE 数	1 或 2 或 4
虚拟机 MIPS	500～1 000
虚拟机带宽	500～1 000 Mbps
虚拟机内存	1～4 GB
虚拟机硬盘	20～100 G

为了更好地验证本文所提任务调度算法 QTS 的真实性和有效性,在实验环境下,对任务到达的批次以及总的到达时间范围进行了量化,具体的量化值将根据实验目标设定,本文实验的主要目标是验证本文任务调度机制 QTS 中加入的动态优先级是否发挥了有效的作用;QTS 与轮转调度、Max-Min 和 Min-Min 算法机制相比,是否有更优的调度表现.

实验 1 为了验证 QTS 中加入的动态优先级的有效性,设定实验环境下任务的参数为:用户每次提交任务规模在1 000个以内,任务需要的 PE 个数 1～2 个随机,任务长度在 500～100 000条指令范围内,任务静态优先级随机生成,期望完成时间随机生成,但与优先级高低呈正相关.模拟用户在 10 个小时内提交 20 批任务,研究随着虚拟机数目的增多,任务完成时间满意度的变化机器对比情况,调度结果对比如图 3 所示.

由图 3 可看出,采用动态优先级策略的调度方案始终优于采用静态优先级的调度方法,这种满意度的差距在虚拟机数目较少的时候表现得尤为明显,由图 3 可以看出,当虚拟机数目为 5 时,采用动态优先级策略比只用静态优先级在满意度上要高出 7.6%,这主要因为虚拟机数目较少时安排给每台虚拟机的任务更多,单位时间内的任务密度更大,从而导致更多的任务处于等待状态,随着等待时间的增加,采用动态优先级策略将会调整等待队列中的任务优先级从而导致任务执行顺序发生变化,使其调度向更优的满意度优化.随着虚拟机数目的增多,任务完成时间满意度都呈上升趋势,但趋势逐渐变缓,二者之间的满意度差距也在逐渐变小,并都趋近于 1,即完全满意程度.至于最终的调度结果是否能达到 1,取决于用户对任务期望完成时间的设定,如果用户设定的期望完成时间过小,以至于其申请的虚拟机性能均无法在期望完成时间内做完该任务,则满意度永远达不到 1.

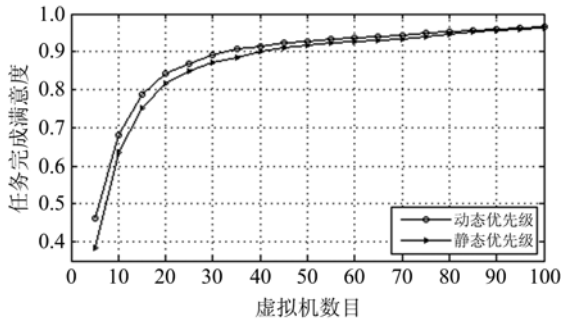

图 3 动态优先级与静态优先级比较

Fig. 3 The comparison of dynamic priority and static priority

实验 2 对比本文任务调度机制、轮转调度、Max-Min 和 Min-Min 算法,研究当任务规模增大时,调度结果对用户期望完成时间的满意程度.设定参数为用户 10 个小时内提交 20 批次任务,初始状态下,每批次任务数目以 50 为最大规模随机产生任务数量,随后的任务规模增大将按照初始值等比例增加,对不同算法进行测试时,采用的数据完全一样,其他参数与实验 1 相同,则任务完成的满意度结果如下图 4 所示.

由图 4 可看出,在不同任务规模下,QTS 在满意度上均比其他调度策略有更优的表现.由于开始情况下任务规模较小,四种调度方案在调度结果上时间满意度都比较高,四种调度方案产生的调度结果满意度比较接近,但是随着任务规模的增大,虽然时间满意度均降低,QTS 与其他三种策略相比,降低幅度更小.与 Min-Min 算法相比,当任务规模变大时候 QTS 调度结果在满意度上比 Min-Min 高出

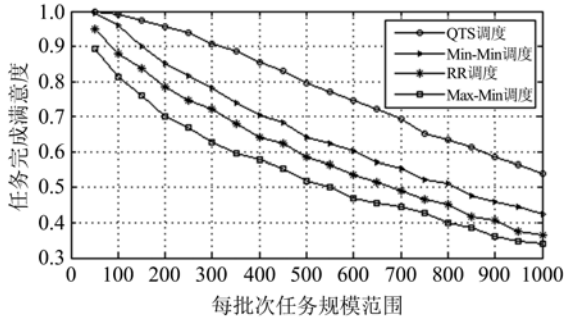

图 4　四种调度方案对比

Fig. 4　The comparison of four scheduling schemes

10 个百分点左右. 从图 4 同样可以发现，Max-Min 的调度结果比轮转调度差，主要是因为 Max-Min 更加适合于任务集是由大批量小任务和少数长任务组成的情况，而本文实验中任务大小是随机的，所以 Max-Min 的表现要差很多.

为了进一步验证 QTS 任务调度机制优于其他三种调度方案，在实验 2 中同时统计出了所有期望时间内完成的任务数目，以下为四种调度方案的统计结果对比，如图 5 所示.

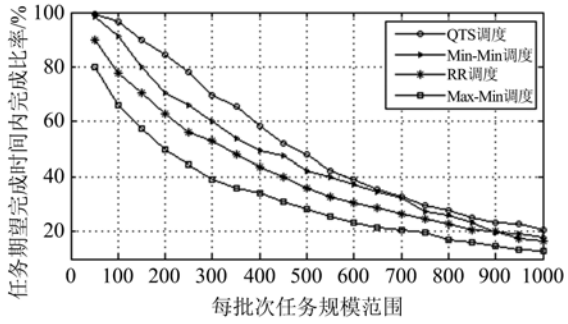

图 5　四种调度方案下“完全满意完成率”比较

Fig. 5　“Fully satisfied completion rate” comparison under the four scheduling schemes

从图 5 可以看出，QTS 产生的任务调度结果中，任务能在期望时间内完成的数目占总数的比率要高于其他三种调度策略. 同样，随着任务规模的增大，任务期望完成时间内完成的比率均在下降，这同样是由于任务密度过大，导致过多的任务处于等待队列中的缘故造成的. 由图 5 可知，当任务规模增到 900 的时候，四种任务调度策略所产生的期望时间内完成任务数目已经很接近，随着任务规模的进一步增大，这种差距将更加缩小.

综上可得，QTS 任务调度机制在任务调度上优于轮转调度、Max-Min 调度以及 Min-Min 调度，QTS 是一种有效的任务调度机制.

5　结论

本文所研究的一种新的任务调度机制，综合考虑了用户对任务的期望完成时间和任务优先级的 QoS 请求，为更好地评价调度结果的优劣以及解决在调度过程中如何映射任务到虚拟机中等问题，为用户 QoS 中的两项指标分别建立了模型，用于调度过程中提供决策依据，引入了任务期望完成时间满意度概念来量化调度结果的优劣. 通过对快照的演化计算任务的执行满意度，为调度提供较优的判断依据；相比于其他任务调度算法，该算法还考虑了任务的多核并发性要求. 实验结果表明，本文所提算法在调度质量上优于轮转、传统的 Max-Min 和 Min-Min 算法.

参考文献(References)

[1] 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)：1 337-1 348.

[2] Wang L Z，Ranjan R，Chen J J et al. Cloud Computing：Methodology，Systems And Applications [M]. Boca Raton：CRC Press，2012.

[3] Liu G，Li J，Xu J C. An improved Min-Min algorithm in cloud computing [C]// Proceedings of the 2012 International Conference of Modern Computer Science and Applications. Berlin，Germany：Springer，2013：47-52.

[4] Guo L Z，Zhao S G，Shen S G，et al. Task scheduling optimization in cloud computing based on heuristic algorithm [J]. Journal of Networks，2012，7(3)：547-553.

[5] Li K，Xu G C，Zhao G Yu，et al. Cloud task scheduling based on load balancing ant colony optimization [C]// Sixth Annual ChinaGrid Conference. Dalian，China：IEEE Press，2011：3-9.

[6] 史少峰，刘宴兵. 基于动态规划的云计算任务调度研究 [J]. 重庆邮电大学学报. 2012，24(6)：687-692.

[7] Cui Y F，Li X M，Dong K W，et al. Cloud computing resource scheduling method research based on improved genetic algorithm [J]. Advanced Materials Research，2011，271：552-557.

[8] Sindhu S，Mukherjee S. Efficient task scheduling algorithms for cloud computing environment [C]// International Conference on High Performance Architecture and Grid Computing. Chandigard，India：Springer，2011：79-83.

[9] 朱宗斌，杜中军. 基于改进 GA 的云计算任务调度算法[J]. 计算机工程与应用. 2013，49(5)：77-80.

[10] Wang L Z, von Laszewski G, Kunze M, et al. Schedule distributed virtual machines in a service oriented environment [C]// Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications. Perth, Australia: IEEE Press, 2010: 230-236.

[11] Fang Y Q, Wang F, Ge J W. A task scheduling algorithm based on load balancing in cloud computing [C]// International Conference on Web Information Systems and Mining. Sanya China: Springer 2010, 6318: 271-277.

[12] Wang J P, Zhu Y L, Feng H Y. A multi-task scheduling method based on ant colony algorithm [J]. Advances in information Sciences and Service Sciences, 2012, 4(11): 185-192.

[13] Rahman M M, Thulasiram R, Graham P. Differential time-shared virtual machine multiplexing for handling QoS variation in clouds [C]//Proceedings of the 1st ACM Multimedia International Workshop on Cloud-Based Multimedia Applications and Services for E-health. Nara, Japan: ACM Press, 2012: 3-8.

[14] Jung J K, Kim N U, Jung S M, et al. Improved CloudSim for simulating QoS-based cloud services [C]// Ubiquitous Information Technologies and Applications. Netherlands: Springer, 2013: 537-545.

[15] 孙瑞锋, 赵政文. 基于云计算的资源调度策略[J]. 航空计算技术, 2010, 40(3): 103-105.

[16] Lin W W, Chen L, Wang J Z, et al. Bandwidth-aware divisible task scheduling for cloud computing [J]. Software: Practice and Experience, 2014, 44(2): 163-174.

[17] Buyya R, Ranjan R, Calheiros R N. Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities [C]// International Conference on High Performance Computing & Simulation. Leipzig, Germany: IEEE Press, 2009: 1-11.

[18] Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms [J]. Software: Practice & Experience, 2011, 41(1): 23-50.

(上接第 589 页)

[16] Hegselmann R, Krause U. Opinion dynamics and bounded confidence: Models, analysis, and simulation [J]. Journal of Artificial Societies and Social Simulation, 2002, 5(3): 1-24.

[17] 张彦超, 刘云, 张海峰, 等. 基于在线社交网络的信息传播模型[J]. 物理学报, 2011, 60(5): 050501(1-7).

[18] Zhang Le, Zhong Qi, Li Zheng. The ISKS-ISK model of crisis information dissemination under time-varying communication rules [J]. Journal of University of Science and Technology of China, 2010, 40(9): 985-990.
张乐, 钟琪, 李政. 时变传播规则下的危机信息 ISKS-ISK 两阶段传播模型及仿真[J]. 中国科学技术大学学报, 2010, 40(9): 985-990.

[19] Liu Xionghong, Zhang Haifeng, Qin Xiaowei, et al. Research of rumor spreading on weighted short message networks[J]. Journal of University of Science and Technology of China, 2012, 42(5): 423-430.
刘星宏, 张海峰, 秦晓卫, 等. 加权短信网络上的谣言传播行为研究[J]. 中国科学技术大学学报, 2012, 42(5): 423-430.

[20] Sudbury A. The proportion of the population never hearing a rumour[J]. Journal of Applied Probability, 1985, 22(2): 443-446.

[21] Zhou J, Liu Z H, Li B W. Influence of network structure on rumor propagation[J]. Physics Letters A, 2007, 368(6): 458-463.

[22] Lü L, Chen D B, Zhou T. The small world yields the most effective information spreading[J]. New Journal of Physics, 2011, 13(12): 123005(1-10).

[23] 顾亦然, 夏玲玲. 在线社交网络中谣言的传播与抑制[J]. 物理学报, 2012, 61(23): 238701(1-7).

[24] 王辉,韩江洪,邓林,等.基于移动社交网络的谣言传播动力学研究[J].物理学报,2013,62(11):110505(1-12).

[25] 钱颖, 张楠, 赵来军, 等. 微博舆情传播规律研究[J]. 情报学报, 2012, 31(12): 1 299-1 304.

[26] 周涛,汪秉宏,韩筱璞,等.社会网络分析及其在舆情和疫情防控中的应用[J].系统工程学报,2010,25(6): 742-754.