一种基于分层多代理的云计算负载均衡方法

陶晓玲[1,2],韦 毅[2],王 勇[2,3]
(1. 桂林电子科技大学认知无线电与信息处理省部共建教育部重点实验室,广西桂林 541004;
2. 桂林电子科技大学广西高校云计算与复杂系统重点实验室,广西桂林 541004;
3. 桂林电子科技大学广西可信软件重点实验室,广西桂林 541004)

摘 要: 针对现有云计算系统中负载均衡方法的不足,借鉴系统逻辑分层和多代理的思想,提出一种基于分层多代理的云计算负载均衡方法.通过对云计算平台逻辑分层,在任务代理层设置任务监控代理和任务子代理,根据用户任务的差异性,采用基于任务优先级和QoS目标约束的调度策略协同完成任务调度;在资源代理层设置资源监控代理和资源子代理,考虑物理节点的异构性,采用基于启发式贪婪的资源分配策略协同完成虚拟机到物理节点的映射.通过评估对比仿真实验,结果表明该方法在任务调度效率、任务完成时间、截止时间违背率和负载均衡度方面表现更优,多代理有效地分担了中心管理节点的管理负载,使云计算平台的任务处理能力、资源利用率及鲁棒性均得到了进一步的提升.

关键词: 负载均衡; 云计算平台; 分层; 多代理
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2016)09-2106-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.09.013

A Load Balancing Method Based on Hierarchy and Multi-agent for Cloud Computing Platform

TAO Xiao-ling[1,2], WEI Yi[2], WANG Yong[2,3]
(1. *Key Laboratory of Cognitive Radio and Information Processing, Guilin University of Electronic Technology, Guilin, Guangxi* 541004, *China*;
2. *Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin, Guangxi* 541004, *China*;
3. *Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, Guangxi* 541004, *China*)

Abstract: To overcome the shortcoming of the load balancing method in cloud computing, a load balancing method for cloud computing platform is proposed, which is inspired by the system logic hierarchical and multi-agent theory. By making use of the idea of logical layer of cloud computing platform, on the one hand, task monitoring agents and task sub-agents are set up in the task agent layer. According to the diversity of users' tasks, the method completes the task scheduling based on the task priority and QoS constrained; on the other hand, resource monitoring agents and resource sub-agents are set up in resource agent layer. Considering the heterogeneity of physical nodes, the method maps the virtual machine to the physical nodes by the resource allocation policy based on the greedy heuristic. Implementation results show that the new method outperforms the others in terms of efficiency of task scheduling, makespan of task, task time-constraint violation time and load balancing. Moreover, after multi-agent effectively shares the management load of the center management node, task processing ability, resource utilization rate and robustness of the cloud computing platform are significantly improved.

Key words: load balancing; cloud computing platform; hierarchy; multi-agent

收稿日期:2016-02-05;修回日期:2016-04-13;责任编辑:梅志强
基金项目:国家自然科学基金(No.61163058,No.61363006);广西可信软件重点实验室主任基金(No.KX201306);广西高校云计算与复杂系统重点实验室开放课题(No.14104)

1 引言

云计算技术的核心问题就是平台的资源管理,其目标就是通过资源的合理调度,使云计算平台能够高效地处理用户任务.在云计算环境中,面对海量的任务需求时,会出现资源分配不合理的情况.有些节点的资源长时间闲置,有些节点因为其资源频繁被使用、负载过重,各个节点负载的不均衡严重影响了云计算平台的任务处理能力[1].因此,负载均衡是云计算平台必须具备的关键机制.合理的负载均衡框架和方法是提高云计算平台运行效率的重要因素[2].

云计算平台架构大多采用主从模式的管理方式,即中心管理节点作为主节点,平台的其它节点作为从节点,由主节点集中对从节点进行管理.早期的负载均衡策略均部署在云计算平台的中心管理节点上,采用这种方式的方法被称为集中式负载均衡方法.这类方法[3,4]具有部署简单、效率高、便于维护等优点,适合规模较小的云计算平台.然而,集中式方式下,中心管理节点会成为平台的性能瓶颈,尤其在用户任务需求量大、从节点数量多的情况下中心管理节点极易瘫痪(也就是单点失效)[5].

鉴于集中式负载均衡方法的缺陷,分布式负载均衡方法得到了许多研究者的关注.此类方法没有中心控制节点,每个节点都参与资源的调度与分配,具有较好的扩展性及鲁棒性.Bittencourt 等[6]提出一种基于博弈论的分布式负载均衡算法,网络节点之间通过博弈相互转嫁负载使自己的负载最小化,从而使系统达到纳什均衡来实现最优的负载均衡.因为分布式负载均衡策略缺乏中心控制,云计算平台中各节点之间的通信及负载信息的共享等交互机制是实现系统负载均衡目标的关键.Randles 等[5]研究比较了三种分布式的负载均衡算法(蜜蜂觅食算法,随机行走算法以及主动聚类算法),其中蜜蜂觅食算法采用“广告牌”机制,主动聚类算法使用服务发现系统,而随机行走算法则用路由表创建行走的网络图来实现节点间的交互和资源分配,最终达到系统整体上的负载均衡.

近年来,有些研究者引入代理执行虚拟机迁移操作,实现云计算系统的动态负载管理.文献[7]利用移动代理管理混合云中所有的资源和监控系统行为,并且协商所有的活动,目的是为了实现私有云和公有云之间自动的、智能的服务迁移.文献[8]通过多代理基于开放知识架构构建交互模型,用于完成虚拟机迁移的工作流.文献[9]将负载均衡协议和能量感知聚合协议赋予多代理,协同完成虚拟机的动态迁移,实现云数据中心的分布式负载管理.以上文献仅仅专注于将代理用于云计算系统的资源分配方面,而忽略了任务调度对系统负载均衡的影响.本文从云计算平台的体系结构特点出发,将平台逻辑分层,并基于多代理分布式、协同完成云计算平台的任务调度与资源分配,从而实现整个平台的负载均衡.

2 相关工作

根据云计算平台的特点,其负载均衡方法主要从任务调度和资源分配两个方面进行研究.现有的任务调度策略通常分为两类:静态任务调度和动态任务调度.经典的静态任务调度方法有:顺序调度、先到先服务调度、短任务优先调度、均衡调度、贪婪调度等.这些静态的任务调度策略是根据任务分配时当前的虚拟机负载情况,加上一些先验知识进行调度,运行过程中任务不能重新分配.此类方法的优点是实现比较简单、负载均衡所需的系统额外开销小,适用于待处理任务规模较大,且任务需求接近的场合.但是由于静态任务调度方法不能实时动态地监测虚拟机的负载变化,可扩展性和自适应能力极差.

随着云用户规模的逐渐庞大,用户任务的种类和需求愈趋复杂,静态任务调度也愈趋不能满足实际需要,国内外专家和学者纷纷转向了对动态任务调度方法的研究,通常他们改进传统的一些启发式算法来解决任务调度问题.Su 等[10]提出一种包含两级启发式策略的节约成本的任务调度算法,首先将任务队列表示成一个有向无环图,通过任务的特定要求计算各个任务的优先级,然后基于帕累托最优的理论找出任务到虚拟机的映射策略,实验表明该方法能够有效地减小任务开销.Zuo 等[11]提出了一种截止时间约束下基于粒子群优化的自适应学习任务调度方法,通过自适应调整参数,设计特殊的粒子群拓扑以及采用多个粒子群协同优化的方式避免粒子群优化陷入局部最优解,从而提高了方法的鲁棒性.Shojafar 等[12]提出一种将模糊理论和遗传算法相结合的启发式任务调度方法,对任务总体执行效率进行优化,实验表明该方法能够改善任务执行时间、开销和云计算平台的平均负载均衡度.以上这些动态任务调度方法能够针对虚拟机的动态变化的负载情况进行任务调度,但是由于需要实时监视虚拟机的负载状况,因此会额外增加云计算平台中心管理节点的开销,再加上复杂的任务调度策略,中心管理节点往往出现过载,不适用于用户任务规模较大的情况.

目前对资源分配方法的研究主要集中在如何针对物理节点的运行状况动态地部署和迁移虚拟机,使得云计算系统的计算资源和网络带宽资源得到有效利用,并致力于减小系统的整体能耗.Cho 等[13]结合蚁群优化和粒子群优化算法提出一种混合启发式虚拟机调

度算法,该算法无需获取额外任务信息,仅通过历史信息去预测新任务的动态环境需求. Piao 等[14]提出一种网络感知的虚拟机部署和迁移方法,用以减少虚拟机之间的数据传输时间消耗,改善云计算系统整体性能,然而这种方法可能导致物理节点资源利用率降低. Sonneck 等[15]提出一种基于关系感知的虚拟机迁移技术,该技术通过监视虚拟机间的亲和度,使用一种分布式交换算法进行虚拟机迁移,最小化物理节点间的通信开销,以实现负载均衡. Shrivastava 等[16]评估了运行在不同物理节点上的虚拟机所执行的任务间的关系,将具有任务强关联的虚拟机迁移至同一物理节点上,以减小节点间的网络开销,提升云计算平台的执行效率,但此方法没有考虑到平台的成本开销以及负载均衡问题. Zhao 等[17]提出了一种启发式自适应多目标优化的虚拟机迁移算法,该算法基于改进遗传算法和帕累托最优解,以减小云计算中心的能耗并实现负载均衡.

代理技术被有效应用于跨域多个异构云计算平台的任务移植和互操作[18]以及云服务组合[19]中. 有学者也针对该技术用于云计算系统的虚拟机迁移方面进行了研究. Fan 等[7]提出了一种基于代理的智能服务迁移框架,通过设计原型系统和评估分析,表明该框架可以有效应用在混合云中. Anderson[8]等描述了一种基于 Lightweight Coordination Calculus (LCC)系统的多代理虚拟机管理模型,用于规范云数据中心内部以及中心之间虚拟机的迁移行为. Gutierrez-Garcia 等[9]将协同多代理用于虚拟机的动态迁移,以分布式的方式均衡和整合异构负载,并最终通过实验证明了该方法的有效性. 以上基于代理的虚拟机迁移策略还存在一些问题,如代理部署过多会增加节点资源开销;虚拟机迁移反而会增加系统负载等问题.

3 分层多代理的整体框架及设计

针对现有云计算系统中负载均衡方法的不足,借鉴系统逻辑分层和多代理的思想,本文提出了一种基于分层多代理的云计算负载均衡方法. 首先逻辑上将云计算平台进行分层,不同层之间由多个代理共同承担平台管理节点的工作. 其次通过多代理之间相互交流、协同工作,能够并行高效地处理多个云用户任务. 最后根据各个物理节点的负载情况进行资源分配,从整体上实现了云计算平台的负载均衡,提高平台的资源利用率,并增强平台的鲁棒性. 基于分层多代理的云计算负载整体架构如图 1 所示.

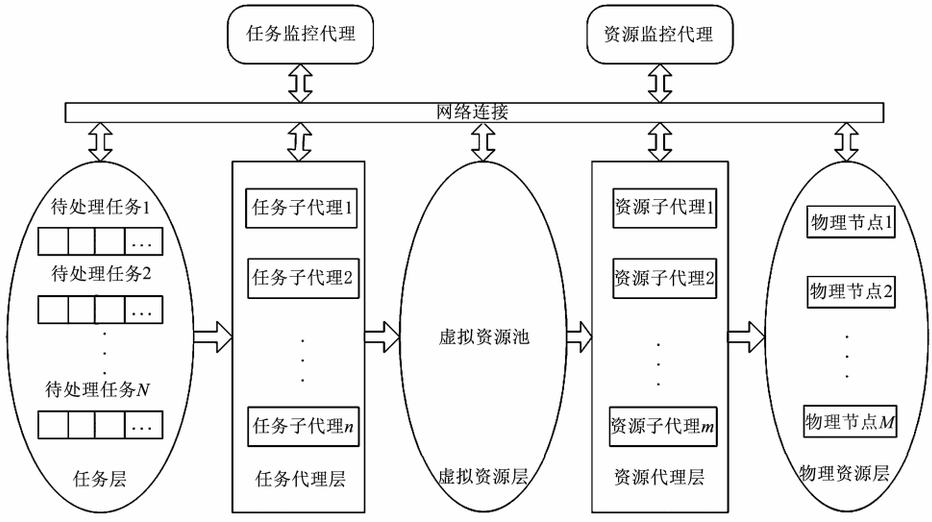

图1 基于分层多代理的云计算负载均衡整体架构

从图 1 可知,云计算平台逻辑划分为:任务层、任务代理层、虚拟资源层、资源代理层和物理资源层,并设置任务监控代理、任务子代理、资源监控代理和资源子代理共同负责管理云计算平台. 任务层中有用户提交的 $1 \sim N$ 个待处理任务,物理资源层有 $1 \sim M$ 个物理节点,物理资源层的各个节点资源被虚拟化成虚拟资源信息,整合至虚拟资源池,构成虚拟资源层. 任务代理层由 1 个任务监控代理及 $1 \sim n$ 个任务子代理构成,负责任务层与虚拟资源层之间的任务调度. 资源代理层由 1 个资源监控代理及 $1 \sim m$ 个资源子代理构成,负责虚拟资源层与物理资源层之间的资源分配.

3.1 任务调度设计

任务调度是根据任务的需求,采用合理的策略,完成待处理任务到合适的虚拟机上的映射. 本文将任务调度模式设计为主从结构,即由一个任务监控代理指导多个任务子代理协同完成任务调度.

任务监控代理的功能是监视任务层的当前任务信息、各任务子代理的当前负载信息、虚拟资源池当前的虚拟资源信息,并定期向所有任务子代理通报当前虚拟资源信息;将任务层中各待处理任务分别赋予优先级,分析任务需求并定义待处理任务的类型;根据任务子代理选择策略,将待处理任务分发给合适的任务子代理.

任务子代理的功能是接收任务监控代理分发的任务,根据任务信息计算所需的虚拟资源需求;向资源监控代理提交虚拟资源需求,等待资源子代理部署虚拟机;将待处理任务调度到创建好的虚拟机上执行;从相应的资源子代理处得到任务执行信息,报告给任务监控代理;在任务执行完毕时向相应的资源子代理发送虚拟机撤销请求.

3.2 资源分配设计

为了保证云计算平台中物理节点的资源得到有效利用,需要实时把握各个物理节点的负载情况,使节点资源均衡地分配给虚拟机,实现负载均衡,需要一系列的监控及资源分配机制来完成. 本文的资源分配方式设计为主从结构,即由一个资源监控代理指导多个资源子代理协同完成资源分配.

资源监控代理的功能是收集并定期向所有资源子代理通报各物理节点当前负载信息;监视各资源子代理当前的负载信息;将物理节点的物理资源虚拟化成虚拟资源信息并发送到虚拟资源层的虚拟资源池;根据任务子代理发来的虚拟资源需求来划分虚拟资源池中的虚拟资源;根据资源子代理选择策略,选择某个资源子代理部署相应的虚拟机.

资源子代理的功能是接收资源监控代理发来的当前物理节点负载信息和虚拟资源需求;根据物理节点的当前负载,选出满足虚拟资源需求的物理节点,标记为可行物理节点;根据虚拟资源需求队列,依次在可行物理节点上部署虚拟机;启动当前虚拟机并监视该虚拟机运行状态;从该虚拟机收集当前任务执行信息,反馈给创建该虚拟机的任务子代理;根据任务子代理发送的虚拟机撤销请求,撤销相应的虚拟机并释放所部署的物理节点资源;将该虚拟机撤销信息发送给资源监控代理.

4　调度策略

4.1　任务调度策略

作为一种按需服务平台,良好的服务质量(QoS)保证是评价云计算平台性能的一项重要指标. 本文选取用户普遍关心的任务截止时间违背率,即实际完成时间超越其截止时间底线的任务占比作为任务调度的 QoS 目标约束要求,并根据任务的截止时间来定义待处理任务的优先级.

令 P 为待处理任务的优先级,其计算公式如下:

$$P = T_{\text{deadline}} - T_{\text{arrival}} \tag{1}$$

其中 T_{deadline} 为任务的截止时间、T_{arrival} 为任务到达时间,优先级 P 越小则表明该任务优先级越高.

考虑到用户任务的差异性,将待处理任务根据任务需求进行分类,并根据任务类型进行调度,任务类型定义方式如下:

令 t_i 为任务集合 $T = \{t_1, t_2, \cdots, t_i, \cdots, t_n\}$ $(1 \leqslant i \leqslant n)$ 中第 i 个待处理任务,t_i 的任务信息 $\text{ts}_i = \{\text{ts}_{i\_\text{len}}, \text{ts}_{i\_\text{ram}}, \text{ts}_{i\_\text{net}}\}$,其中 ts_{i_len} 为任务长度,用 MI 表示,ts_{i_ram} 为任务所需内存,用 MB 表示,ts_{i_net} 为任务所需带宽,用 bps 表示. 则 ts_i 所需计算能力 ts_{i_cpu} 计算公式如下:

$$\text{ts}_{i\_\text{cpu}} = \frac{\text{ts}_{i\_\text{len}}}{P} \tag{2}$$

综合任务集合中所有任务信息,计算待处理任务所需平均 CPU 资源 $\overline{\text{ts}_{\text{cpu}}}$、平均内存资源 $\overline{\text{ts}_{\text{ram}}}$、平均带宽资源 $\overline{\text{ts}_{\text{net}}}$:

$$\overline{\text{ts}_{\text{cpu}}} = \frac{1}{N}\sum_{i=1}^{N} \text{ts}_{i\_\text{cpu}} \tag{3}$$

$$\overline{\text{ts}_{\text{ram}}} = \frac{1}{N}\sum_{i=1}^{N} \text{ts}_{i\_\text{ram}} \tag{4}$$

$$\overline{\text{ts}_{\text{net}}} = \frac{1}{N}\sum_{i=1}^{N} \text{ts}_{i\_\text{net}} \tag{5}$$

对比各项任务信息,得到 t_i 的任务类型 $\text{ts}_{i\_\text{Type}}$:

$$\text{ts}_{i\_\text{Type}} = \begin{cases} 0, & \text{ts}_{i\_\text{cpu}} \geqslant \overline{\text{ts}_{\text{cpu}}} \\ 1, & \text{ts}_{i\_\text{cpu}} < \overline{\text{ts}_{\text{cpu}}} \cap \text{ts}_{i\_\text{ram}} \geqslant \overline{\text{ts}_{\text{ram}}} \\ 2, & \text{ts}_{i\_\text{ram}} < \overline{\text{ts}_{\text{ram}}} \cap \text{ts}_{i\_\text{net}} \geqslant \overline{\text{ts}_{\text{net}}} \\ 3, & \text{other} \end{cases} \tag{6}$$

其中 0 表示 CPU 密集型,1 表示内存密集型,2 表示网络带宽密集型,3 表示其他类型.

由于云计算平台所承载的任务规模庞大,为避免单个任务代理承担过多的调度任务造成节点过载,采用一个任务监控代理与多个任务子代理协同工作. 为保证多个任务子代理的负载均衡,任务监控代理需要时刻监视各个任务子代理当前的负载,选择负载最低的任务子代理承担当前待处理任务. 令 L_j 为第 j 个任务子代理当前的负载,其计算公式如下:

$$\begin{aligned} L_j = {} & w_1 \times \text{rs}_{j\_\text{cpu}} + w_2 \times \text{rs}_{j\_\text{ram}} + w_3 \times \text{rs}_{j\_\text{net}} \\ & + w_4 \times \text{rs}_{j\_\text{ql}} + w_5 \times \text{rs}_{j\_\text{rt}} \end{aligned} \tag{7}$$

其中第 j 个任务子代理中的各项参数:rs_{j_cpu} 表示可用 CPU 资源、rs_{j_ram} 表示可用内存资源、rs_{j_net} 表示可用网络带宽资源、rs_{j_ql} 表示任务队列、rs_{j_rt} 表示响应时间,各参数权重 w_l 满足约束条件:

$$\sum_{l=1}^{5} w_l = 1 \quad w_l \in [0,1] \tag{8}$$

为保证 QoS 目标约束要求,任务应尽可能在截止时间内完成,因此任务对应的虚拟机创建时其 CPU 资源是首要考虑的指标,其次考虑内存和带宽资源,任务 i 的虚拟机资源信息应满足以下约束条件:

$$\mathrm{ts}_{i\_\mathrm{len}}/(P-t)\leqslant \mathrm{vm}_{i\_\mathrm{cpu}} \tag{9}$$

$$\mathrm{ts}_{i\_\mathrm{ram}}\leqslant \mathrm{vm}_{i\_\mathrm{ram}} \tag{10}$$

$$\mathrm{ts}_{i\_\mathrm{net}}\leqslant \mathrm{vm}_{i\_\mathrm{net}} \tag{11}$$

其中 t 为任务在队列中的等待时间,$\mathrm{vm}_{i\_\mathrm{cpu}}$、$\mathrm{vm}_{i\_\mathrm{ram}}$、$\mathrm{vm}_{i_\mathrm{net}}$分别为所创建虚拟机的 CPU、内存、网络带宽资源. 创建的虚拟机类型 $\mathrm{vm}_{i_\mathrm{Type}}$和任务类型保持一致:

$$\mathrm{vm}_{i\_\mathrm{Type}}=\mathrm{ts}_{i\_\mathrm{Type}} \tag{12}$$

4.2 资源分配策略

相关工作的文献分析中已指出,基于代理的虚拟机迁移可能会增加系统负载,因此本文在资源分配中不考虑虚拟机迁移问题,着重从虚拟机部署方面研究资源分配策略,资源分配由资源监控代理和资源子代理协同完成.

由于虚拟机根据任务类型分类,因此资源子代理也相应地被分为 CPU 密集型、内存密集型、网络带宽密集型以及其他类型. 为保证多个资源子代理的负载均衡,资源监控代理需要时刻监视各个资源子代理当前的负载,选择负载最低的资源子代理负责部署虚拟机. 资源子代理的选择策略与任务子代理的选择策略一致,此处不再赘述.

云计算平台作为一种低成本的可伸缩性平台,其物理节点资源异构多样,通常表现为节点的计算能力、存储能力、网络传输能力均有不同,因此本文按节点资源能力把物理节点的类型划分为 CPU 密集型、内存密集型、网络带宽密集型以及其他类型. 由于不同时间段任务层中待处理任务的类型分布动态变化,因此需要采用节点类型动态分布调整策略,通过资源监控代理调整物理节点的类型,策略描述如下:

根据当前时刻待处理任务及物理节点信息,令 $\mathrm{TN}_{\mathrm{cpu}}$为 CPU 密集型任务的数量,$\mathrm{TN}_{\mathrm{ram}}$为内存密集型任务的数量,$\mathrm{TN}_{\mathrm{net}}$为网络带宽密集型任务的数量,$\mathrm{HN}_{\mathrm{cpu}}$为 CPU 密集型物理节点的数量,$\mathrm{HN}_{\mathrm{ram}}$为内存密集型物理节点的数量,$\mathrm{HN}_{\mathrm{net}}$为网络带宽密集型物理节点的数量,预先设定动态调整触发值 $\theta\in[0,1]$,一旦满足:

$$\left|\frac{\mathrm{TN}_I}{\mathrm{TN}_{\mathrm{cpu}}+\mathrm{TN}_{\mathrm{ram}}+\mathrm{TN}_{\mathrm{net}}}-\frac{\mathrm{HN}_I}{\mathrm{HN}_{\mathrm{cpu}}+\mathrm{HN}_{\mathrm{ram}}+\mathrm{HN}_{\mathrm{net}}}\right|\leqslant\theta,\quad I\in\{\mathrm{cpu},\mathrm{ram},\mathrm{net}\} \tag{13}$$

则资源监控代理对物理节点的类型进行重新调整.

令 λ_{cpu}、λ_{ram}、λ_{net}分别为 CPU、内存、网络带宽调整阈值,其计算公式如下:

$$\lambda_{\mathrm{cpu}}=\frac{\mathrm{TN}_{\mathrm{cpu}}}{\mathrm{TN}_{\mathrm{cpu}}+\mathrm{TN}_{\mathrm{ram}}+\mathrm{TN}_{\mathrm{net}}}*\sum_{k=1}^{M}H_{k\_\mathrm{cpu}} \tag{14}$$

$$\lambda_{\mathrm{ram}}=\frac{\mathrm{TN}_{\mathrm{ram}}}{\mathrm{TN}_{\mathrm{cpu}}+\mathrm{TN}_{\mathrm{ram}}+\mathrm{TN}_{\mathrm{net}}}*\sum_{k=1}^{M}H_{k\_\mathrm{ram}} \tag{5}$$

$$\lambda_{\mathrm{net}}=\frac{\mathrm{TN}_{\mathrm{net}}}{\mathrm{TN}_{\mathrm{cpu}}+\mathrm{TN}_{\mathrm{ram}}+\mathrm{TN}_{\mathrm{net}}}*\sum_{k=1}^{M}H_{k\_\mathrm{net}} \tag{6}$$

其中 H_{k_cpu}为物理节点 k 当前的可用 CPU,用 MIPS 表示,H_{k_ram}为物理节点 k 当前的可用内存,用 MB 表示,H_{k_net}为物理节点 k 当前的可用网络带宽,用 bps 表示.

物理节点 k 当前的可用资源与资源调整阈值相比较,得到该节点的类型 H_{k_Type}:

$$H_{k\_\mathrm{Type}}=\begin{cases}0, & \lambda_{\mathrm{cpu}}\leqslant H_{k\_\mathrm{cpu}}\\ 1, & \lambda_{\mathrm{ram}}\leqslant H_{k\_\mathrm{ram}}\\ 2, & \lambda_{\mathrm{net}}\leqslant H_{k\_\mathrm{net}}\\ 3, & \mathrm{other}\end{cases} \tag{17}$$

其中 0 表示 CPU 密集型,1 表示内存密集型,2 表示网络带宽密集型,3 表示其他类型,同一个节点可以同时属于多种类型.

在物理节点上部署虚拟机需要考虑以下几个问题:(1)部署虚拟机的先后顺序;(2)将虚拟机部署到哪些物理节点上;(3)如何保证各物理节点的负载均衡. 由于资源子代理中待部署虚拟机依照任务优先级信息形成队列,因此可以遵循先进先出的部署顺序,基于启发式贪婪算法完成虚拟机到物理节点的映射. 以 CPU 密集型资源子代理为例,启发式贪婪算法描述见算法 1.

算法 1 启发式贪婪算法

输入:待部署的虚拟机列表
$\mathrm{VM}_{\mathrm{cpu}}=\{\mathrm{vm}_1,\mathrm{vm}_2,\cdots,\mathrm{vm}_m\}$,代理所管理的物理节点列表 $H_{\mathrm{cpu}}=\{h_1,h_2,\cdots,h_M\}$
输出:映射规则 $\Omega:\mathrm{VM}\rightarrow H$

```
While(VM_cpu ≠ ∅)
  For each vm = VM_cpu[0]
    Rank H_cpu by the lowest CPU-usage
    For(i = 0; i < H_cpu.size(); i++)
      h = H_cpu[i]
      If vm_cpu ≤ h_cpu & vm_ram ≤ h_ram & vm_net ≤ h_net
        生成虚拟机映射规则 ω: vm→h
        Ω = Ω∪ω
        Break
      Else if
    End for
  End for
End while
Return Ω: VM_cpu→H_cpu
```

算法 1 中 $\mathrm{vm}_{\mathrm{cpu}}$、$\mathrm{vm}_{\mathrm{ram}}$、$\mathrm{vm}_{\mathrm{net}}$分别为待部署虚拟机的 CPU、内存、网络带宽需求,h_{cpu}、h_{ram}、h_{net}分别为被选择物理节点当前的可用 CPU、内存、网络带宽资源. 内存

密集型资源子代理和网络带宽密集型资源子代理的资源分配方式类似. 需要指出的是,其他类型资源子代理采取与 CPU 密集型资源子代理相同的策略,尽可能保证任务能在截止时间内完成.

5　实验与性能分析

5.1　实验环境及评价指标

为了验证本文提出的基于分层多代理的云计算负载均衡方法(LB-HMA)的有效性,采用云计算仿真平台 CloudSim[20] 来模拟云计算实验环境,进行评估对比实验. 实验的参数设置如下:

(1)为了充分模拟云任务规模庞大,种类繁多的特点,向云计算平台提交 50000 个云任务,其任务长度在范围[500,50000]内随机生成,单位 MI;所需内存、带宽参数在合理范围内随机生成;指定任务到达任务池的时间 t_1 在范围[0,120]之间随机生成,任务截止时间 t_2 在时间范围[30,600]之间随机生成,单位为分钟,且为了保证 t_1 和 t_2 之间存在执行任务的时间间隔,随机生成 t_1 和 t_2 满足条件:$t_1 + 15 \leqslant t_2$.

(2)为了充分模拟物理节点性能异构的特点,设置 20 个物理节点,其 CPU 性能在范围[1000,2500]之间随机生成,单位 MIPS;内存、带宽参数在合理范围内随机生成;CPU、内存及网络负载初始值范围设为[0.01,0.1].

(3)设置 10 个任务子代理、10 个资源子代理.

为了测试本文提出的方法在调度效率、任务执行效率以及负载均衡方面的效果,采用以下几种性能评价指标:

(1)$T_{\text{scheduling}}$,调度时间,评价方法调度效率的指标,定义为完成所有任务到物理节点间映射的总耗时,其计算公式如下;

$$T_{\text{scheduling}} = \sum_{i=1}^{N} t_{\text{scheduling}}(\text{ts}_i) \tag{17}$$

其中 $t_{\text{scheduling}}(ts_i)$ 表示第 i 个任务调度的耗时.

(2)Makespan,任务完成时间,反映云计算平台的执行效率,是衡量 QoS 的一个重要指标,其定义为完成所有提交任务需要的时间,其计算公式如下:

$$\text{Makespan} = t_{\text{finish}}(\text{ts}_{\text{last}}) - t_{\text{start}}(\text{ts}_{\text{first}}) \tag{18}$$

其中 $t_{\text{start}}(\text{ts}_{\text{first}})$ 为第一个任务的启动时间,$t_{\text{finish}}(\text{ts}_{\text{last}})$ 为最后一个任务的完成时间.

(3)TTCVR,任务截止时间违背率(Task Time-Constraint Violation Rate),定义为任务的实际完成时间超过其截止时间的任务数占总任务数的百分比,是衡量 QoS 的一个重要指标,其计算公式如下:

$$\text{TTCVR} = \text{TN}_{\text{violation}}/\text{TN} \tag{19}$$

其中 $\text{TN}_{\text{violation}}$ 为违背截止时间的任务数,TN 为任务集合中的任务总数.

(4)LB,负载均衡度,反映负载均衡方法的有效性,其定义为时刻 t 各个节点负载的方差,其计算公式如下:

$$\text{LB}_t = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\text{Load}_i - \text{Load}_{\text{avg}})^2} \tag{20}$$

上式中 m 为节点数量,Load_i 为节点 i 的负载,Load_{avg} 为所有节点平均负载,负载均衡度越小,表示节点之间的负载越均衡.

5.2　性能分析

为了评估本文提出的 LB-HMA 在任务调度方面的性能,采用基于遗传算法的集中式任务调度方法(CTS-GA)(由中心管理节点负责调度)、基于多代理和遗传算法的任务调度方法(TSM-MA-GA)作为对比方法. 实验中用到的遗传算法均迭代 1000 次,三种方法的任务调度时间对比结果如表 1 所示.

表 1　不同方法的调度时间(ms)

调度方法	任务数量(个)				
	10000	20000	30000	40000	50000
LB-HMA	74.67	150.77	240.91	369.92	533.45
CTS-GA	24832.6	52121.3	84622.4	116840	150165.1
TSM-MA-GA	1025.31	2091.5	3074.74	4025.58	5078.70

从表 1 可以看出,CTS-GA 由于经过多次迭代,时间复杂度比较高,因此所需的调度时间最长;通过 10 个任务代理协同调度后,TSM-MA-GA 将调度时间缩短至 CTS-GA 的 1/24 左右;由于本文提出的 LB-HMA 的任务调度没有采用复杂的智能算法,而是从任务需求与 QoS 目标约束方面来考虑,因此耗时最小,仅为 MA-GA 的 1/14 左右,任务调度效率高.

为了评估本文提出的 LB-HMA 在任务执行效率方面的性能,选取经典 Min-Min 算法、基于多代理和遗传算法的任务调度方法(TSM-MA-GA)(其资源分配策略采用 CloudSim 默认的方法)作为对比方法,三种方法在不同任务规模下的任务完成时间和任务截止时间违背率的对比结果如图 2 和图 3 所示.

从图 2 中可以看出,经典 Min-Min 算法的任务完成时间最长,而 TSM-MA-GA 以任务完成时间为适应度进行多次迭代优化,其任务完成耗时优于 Min-Min 算法. 同时也可以看出,由于 TSM-MA-GA 的交叉、变异过程具有随机性,因此相对于其他两种方法,TSM-MA-GA 随任务规模的扩大带来的任务完成时间增长不是完全线性,实验结果也反映出了该算法的特性. 由于本文提出的 LB-HMA 在任务调度时根据任务需求对任务进行分类,而且对物理节点也做了分类,既考虑了任务需求的差异性,又考虑了物理节点的异构性,任务调度和资

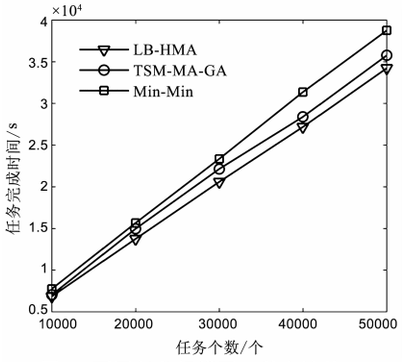

图2 不同任务规模下三种方法的任务完成时间

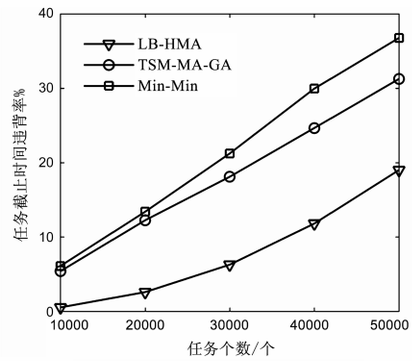

图3 不同任务规模下三种方法的任务截止时间违背率

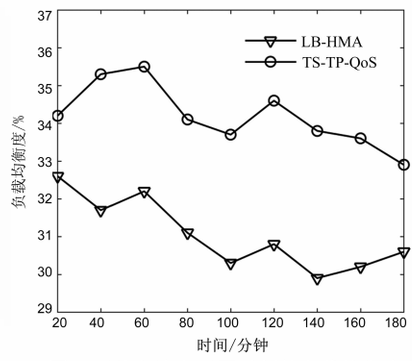

图4 两种方法在不同时刻的负载均衡度

源分配执行更加高效,因此相对于其他两种算法,平均任务完成耗时最低,这也说明 LB-HMA 能够有效地提高云计算平台的资源利用率.

从图3 中可以看出,由于 Min-Min 算法和 TSM-MA-GA 均未考虑 QoS 目标约束,因此任务的截止时间违背率较高,且二者十分接近. 而本文提出的 LB-HMA 根据任务的截止时间对任务进行优先级排序,结合用户任务需求进行任务调度和资源分配,使得任务的截止时间违背率远远小于其他两种算法,保证了 QoS.

为了验证本文提出的 LB-HMA 中资源分配策略的有效性,选取基于任务优先级和 QoS 目标约束的任务调度方法(TS-TP-QoS)作为对比方法,具有如下特点:采用 LB-HMA 的任务调度策略,而资源分配方式采用 CloudSim 默认的方法. 图4 给出了两种方法在不同时刻的负载均衡度对比.

从图4 中可以看出,相较于 TS-TP-QoS,本文提出的 LB-HMA 通过多代理进行资源分配,总体负载均衡度更低,波动更平缓. 这说明 LB-HMA 的各个资源子代理根据任务类型,采用启发式贪婪算法将其对应的虚拟机部署至合适的物理节点上,可以更有效地利用节点的物理资源,而且资源监控代理可以动态地调整节点类型,避免了某些节点长时间处于低负载状态,使得整个云计算平台表现出更好的负载均衡度.

6 结论

本文采用分层多代理解决云计算平台中的负载均衡问题. 通过对云计算平台的逻辑分层,在任务代理层设置任务监控代理和任务子代理,根据用户任务的差异性,采用基于 QoS 目标约束的任务调度策略完成任务调度;在资源代理层设置资源监控代理和资源子代理,考虑物理节点的异构性,着重从虚拟机部署方面研究资源分配策略. 实验表明本文提出的方法既可以保证 QoS,又可以极大地利用各个物理节点的资源,提升了平台的任务处理效率,同时可以有效地分担中心管理节点的管理负载,提高了平台的鲁棒性,增大了平台对海量任务的容纳能力. 此外,方法表现出较低的负载均衡度也表明,它能够均衡地分配平台资源,防止某些物理节点因为负载过重而导致任务处理失败. 下一步我们将把该方法应用于实际云计算平台,并针对实际情况作进一步调整.

参考文献

[1] LD D B, Krishna P V. Honey bee behavior inspired load balancing of tasks in cloud computing environments[J]. Applied Soft Computing, 2013, 13(5): 2292 – 2303.

[2] Cho K M, Tsai P W, Tsai C W, et al. A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing[J]. Neural Computing and Applications, 2015, 26(6): 1297 – 1309.

[3] Bhadani A, Chaudhary S. Performance evaluation of web servers using central load balancing policy over virtual machines on cloud[A]. Proceedings of the Third Annual ACM Bangalore Conference[C]. Bangalore: ACM, 2010. 1 – 4

[4] Dover Z, Gordon S, Hildred T. The Technical Architecture of Red Hat Enterprise Virtualization Environments—Edition 1. Red Hat Enterprise Virtualization 3. 2—Technical Reference Guide[EB/OL]. https://access. redhat. com/documentation/en-US/Red_ Hat_ Enterprise_ Virtualization/3. 2/pdf/Technical_ Reference_ Guide/Red_ Hat_ Enterprise_ Virtualization-3. 2-Technical_ Reference_ Guide-en-US. pdf. 2015-10-25.

[5] Randles M, Lamb D, Taleb-Bendiab A. A comparative study into distributed load balancing algorithms for cloud computing[A]. Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on[C]. Perth: IEEE, 2010. 551 – 556.

[6] Bittencourt L F, Miyazawa F K, Vignatti A L. Distributed

load balancing algorithms for heterogeneous players in asynchronous networks[J]. Journal of Universal Computer Science,2012,18(20):2771 - 2797.
[7] Fan C T, Wang W J, Chang Y S. Agent-based service migration framework in hybrid cloud[A]. High Performance Computing and Communications (HPCC),2011 IEEE 13th International Conference on[C]. Ji′nan: IEEE, 2011. 887 - 892.
[8] Anderson P, Bijani S, Herry H. Multi-agent Virtual Machine Management Using the Lightweight Coordination Calculus[M]. Germany: Springer Berlin Heidelberg, 2013. 123 - 142.
[9] Gutierrez-Garcia J O, Ramirez-Nafarrate A. Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines[J]. Services Computing, IEEE Transactions on, 2015, 8(6): 916 - 929.
[10] Su S, Li J, Huang Q, et al. Cost-efficient task scheduling for executing large programs in the cloud[J]. Parallel Computing, 2013, 39(4): 177 - 188.
[11] Zuo X, Zhang G, Tan W. Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud[J]. Automation Science and Engineering, IEEE Transactions on, 2014, 11(2): 564 - 573.
[12] Shojafar M, Javanmardi S, Abolfazli S, et al. FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method[J]. Cluster Computing, 2015, 18(2): 845 - 845.
[13] Cho K M, Tsai P W, Tsai C W, et al. A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing[J]. Neural Computing and Applications, 2015, 26(6): 1297 - 1309.
[14] Piao J T, Yan J. A network-aware virtual machine placement and migration approach in cloud computing[A]. Grid and Cooperative Computing (GCC), 2010 9th International Conference on[C]. Nanjing: IEEE, 2010. 87 - 92.
[15] Sonnek J, Greensky J, Reutiman R, et al. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration[A]. Parallel Processing (ICPP), 2010 39th International Conference on[C]. San Diego: IEEE, 2010. 228 - 237.
[16] Shrivastava V, Zerfos P, Lee K W, et al. Application-aware virtual machine migration in data centers[A]. INFOCOM, 2011 Proceedings IEEE[C]. Shanghai: IEEE, 2011. 66 - 70.
[17] Zhao J, Ding Y, Xu G, et al. A location selection policy of live virtual machine migration for power saving and load balancing[J]. The Scientific World Journal, 2013, 2013: 1-17.
[18] Zhang Z, Zhang X. Realization of open cloud computing federation based on mobile agent[A]. Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on[C]. Shanghai: IEEE, 2009. 642 - 646.
[19] Gutierrez-Garcia J O, Sim K M. Self-organizing agents for service composition in cloud computing[A]. Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on[C]. Indianapolis: IEEE, 2010. 59 - 66.
[20] 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 and Experience, 2011, 41(1): 23 - 50.

作者简介

陶晓玲　女,1977 年 11 月出生于浙江省金华市. 2008 年毕业于桂林电子科技大学计算机应用技术专业,硕士学位. 现为桂林电子科技大学信息与通信学院副教授. 主要研究方向为网络安全,云计算和计算智能.
E-mail: txl@guet.edu.cn

韦　毅　男,1988 年 11 月出生于广西壮族自治区百色市. 2015 年毕业于桂林电子科技大学信息与通信工程专业,硕士学位. 现从事云计算和人工智能方面的有关研究.
E-mail: 4128wy@163.com

王　勇　男,1964 年 3 月出生于四川省阆中市. 2005 年毕业于华东理工大学控制理论与控制工程专业,博士学位. 现为桂林电子科技大学计算机与信息安全学院教授. 主要研究方向为计算机网络和云计算.
E-mail: wang@guet.edu.cn