计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20151123
54(2):446-456,2017

云计算中资源延迟感知的实时任务调度方法

陈黄科　祝江汉　朱晓敏　马满好　张振仕
(国防科学技术大学信息系统工程重点实验室　长沙　410073)
(hkchen@nudt.edu.cn)

Resource-Delay-Aware Scheduling for Real-Time Tasks in Clouds

Chen Huangke, Zhu Jianghan, Zhu Xiaomin, Ma Manhao, and Zhang Zhenshi
(*Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha* 410073)

Abstract　Green cloud computing has become a central issue, and dynamical consolidation of virtual machines (VMs) and turning off the idle hosts show promising ways to reduce the energy consumption for cloud data centers. When the workload of the cloud platform increases rapidly, more hosts will be started on and more VMs will be deployed to provide more available resources. However, the time overheads of turning on hosts and starting VMs will delay the start time of tasks, which may violate the deadlines of real-time tasks. To address this issue, three novel startup-time-aware policies are developed to mitigate the impact of machine startup time on timing requirements of real-time tasks. Based on the startup-time-aware policies, we propose an algorithm called STARS to schedule real-time tasks and resources, such making a good trade-off between the schedulibility of real-time tasks and energy saving. Lastly, we conduct simulation experiments to compare STARS with two existing algorithms in the context of Google's workload trace, and the experimental results show that STARS outperforms those algorithms with respect to guarantee ratio, energy saving and resource utilization.

Key words　cloud computing; virtualization; scheduling; real-time tasks; energy-efficient; startup time

摘　要　绿色云计算已经成为一个研究焦点,动态整合虚拟机和关闭空闲主机是极具潜力的途径可降低云计算数据中心的能耗.当云平台的负载迅速增加时,系统需要启动更多的主机和创建更多的虚拟机来扩展可用资源.然而,启动主机和创建虚拟机需要一定的时间开销,使得紧急任务难以及时开始,从而延误了截止期.为了解决以上问题,首先提出具有机器启动时间感知的虚拟机扩展策略,以缓解机器启动时间冲击实时任务的时效性要求.基于该策略,设计算法 STARS 来调度实时任务和资源,以在保障任务时效性与节能 2 方面进行权横.最后,使用 Google 的负载数据进行模拟实验,比较算法 STARS 与其他 2 个算法的性能.实验结果表明,在保障任务时效性、节能和资源利用率方面,算法 STARS 优于对比算法.

关键词　云计算;虚拟化;调度;实时任务;节能;机器启动时间

中图法分类号　TP393

为了迎接急剧增长的计算服务需求,云计算数据中心部署的主机规模急剧增长.一个数据中心的主机数量就高达几万台,甚至几十万台[1].运行这些主机,云服务系统需要消耗大量的电能.据统计,从

收稿日期:2015-12-21;**修回日期**:2016-09-06

基金项目:国家自然科学基金项目(61572511, 71271213);国防科学技术大学科研计划项目(ZK16-03-09)
This work was supported by the National Natural Science Foundation of China (61572511, 71271213) and the Scientific Research Project of National University of Defense Technology (ZK16-03-09).

2005—2010 年全球数据中心的能耗提高 56%,占全球总能耗的 1.5%[2].对企业而言,高能耗就意味着高成本.另外,高能耗对生态环境产生较大的负面影响,因为使用煤矿发电会向空气中释放大量的废气[3].云计算数据中心的高能耗问题已经引起各界的极大关注,并成为学术界研究的热点.

大量的研究[4-6]表明,在云计算数据中心提高活跃主机的资源利用效率和减少电能消耗的有效途径是:在云计算数据中心负载下降时,动态整合虚拟机到尽可能少的主机上,然后关闭空闲的主机,以减少能量消耗.由于主机处于完全空闲的状态,功耗仍然是它最大功耗的 50%以上[6],关闭空闲的主机就意味着节约大量的电能消耗.

但是,这类资源整合方法带来了另一个挑战性问题:当云计算数据中心的负载突增时,伸展虚拟机的过程中,创建新虚拟机或者先开启关闭的主机然后再创建虚拟机[4]都需要一定的时间开销,使得某些任务不能及时开始,从而延误了它们的截止期.例如,一个在 0 s 到达的新任务,它的执行时间是 5 s,假设它的截止期是执行时间的 5 倍(即 25 s).启动主机的时间大概为 30 s,创建 1 台虚拟机的时间近似于启动一个操作系统的时间,大概也是 30 s.当该任务到达后再创建新虚拟机,很明显,该任务的截止期将被延误.

为了解决以上问题,本文首先提出了具有机器启动时间感知的虚拟机扩展策略,该策略在每台活跃主机上放置 1 台应急虚拟机,然后借助虚拟机 CPU 资源可以动态伸缩的能力,当某些实时任务的截止期得不到满足时,动态扩大应急虚拟机的 CPU 能力来服务这些任务,当这些任务完成以后,应急虚拟机将释放 CPU 资源,以减少能耗.另外,本文将以上策略集成到滚动优化中,形成调度算法 STARS,该算法在每个新任务到达时,为云计算数据中心生成新的任务和虚拟机调度方案.最后,通过仿真实验对本文的调度算法的有效性进行验证.

1 相关工作

为了解决云数据中心的高能耗问题,目前科研人员已经提出大量能耗感知的调度方法.其中具有巨大潜在价值的是:将主机虚拟化成多个独立的虚拟机来同时运行多个任务,以提高主机的资源利用率和减少使用主机的数量;另外,当数据中心的负载下降时,将虚拟机整合到尽可能少的主机上,然后关闭空闲的主机,以进一步减少能量消耗.

比如,Li 等人[7]针对目前大部分调度算法根据任务的平均运行时间进行调度,很难同时最小化任务的完成时间和能耗的不足,建立了随机任务的运行时间和能耗模型,并提出一个能耗感知的调度算法.Mei 等人[8]针对异构系统中基于任务复制的调度算法过度复制任务造成资源浪费的问题,提出了一个新的能耗感知调度算法来最小化任务的复制量,从而最小化应用的完成时间和能量消耗.Ma 等人[9]针对异构系统的高能耗问题,为带截止期的独立任务设计了能量高效的调度算法.Beloglazov 等人[6]提出基于双阈值的虚拟机动态配置策略,根据主机的实时资源利用率对虚拟机进行整合,并将空闲节点转入睡眠模式.Zhu 等人[4]提出了一种实时任务的节能调度算法 EARH,同时提出了资源动态增加与缩减策略.Xiao 等人[10]设计了一个资源管理方法,借助虚拟化技术动态分配数据中心的资源,同时优化使用主机的数量,以支持绿色计算.马艳等人[11]提出了网格依赖任务的能耗有效调度算法,来优化应用完成时间的同时降低能耗.魏亮等人[12]设计一种面向云计算基础设施基于工作负载预测的整合调度算法,以减少主机使用量、虚拟机迁移次数和资源利用率.何炎祥等人[13]为提高资源和能源的有效利用率,提出了一种迭代式多目标分配优化方法,从能源消耗和资源的均衡使用度 2 个方面出发,利用可交换类指令重排优化和寄存器重分配优化,对总线和存储系统的绿色指标进行改进.Hsu 等人[14]提出一个能耗感知的任务整合方法来限制 CPU 的资源利用率低于预设的阈值,以最小化系统的能量消耗.周景才等人[15]基于用户行为特征,动态调整云计算系统的资源分配策略.Corradi 等人[16]提出一种云管理平台,以优化虚拟机整合的 3 个性能:功耗、主机和网络资源.丁有伟等人[17]针对异构集群下 I/O 密集型的大数据处理任务,提出一种新的能量高效算法,以减少各个节点因为等待而造成的能量浪费.李强等人[18]使用多目标遗传算法来求解虚拟机的放置方案.殷小龙等人[19]改进 NSGA II 来求解虚拟机调度问题,以均衡系统的负载、提高任务执行效率和降低能耗.肖鹏等人[20]针对高性能计算领域的高能耗问题,设计了一种“最小能耗路径”的数据密集型工作流算法.Li 等人[21]针对任务的执行时间具有随机性的特点,提出一个随机任务调度算法,来同时满足任务的时效性和能耗约束.

但是,这些调度方法存在如下问题:当云计算

数据中心的负载突增时,在伸展虚拟机的过程中,启动主机和创建虚拟机需要一定的时间开销,使得某些任务不能及时开始,从而延误了截止期.本文针对该问题,提出一种机器启动时间感知的虚拟机伸展策略,并将其集成到滚动优化中,以提高云计算数据中心保障实时任务时效性的能力,同时降低系统的能量消耗.

2 问题描述

2.1 调度框架

本文研究的虚拟云服务系统包括一个主机集合 $H=\{h_1,h_2,\cdots,h_m\}$,其中 m 表示主机的数量.这些主机为用户的计算服务提供硬件平台.任意一台主机 $h_j\in H$ 可描述为 $h_j=(f_j^{\max},m_j,n_j,p_j^{\max})$,其中 $f_j^{\max},m_j,n_j,p_j^{\max}$ 分别表示主机的最大 CPU 频率(MHz)、内存(GB)、带宽(Mb/s)和最大功耗(W).

每台主机都可容纳一个虚拟机集合 $VM_j=\{vm_{j1},vm_{j2},\cdots,vm_{jm'}\}\cup\{lvm_j\}$,其中下标 m' 表示主机 h_j 上非应急虚拟机的个数;主机 h_j 上的第 k 台虚拟机表示为 $vm_{jk}\in VM_j$,$1\leqslant k\leqslant m'$;lvm_j 为主机 h_j 上唯一的 1 台应急虚拟机.对于每台虚拟机 vm_{jk},我们分别使用 f_{jk},m_{jk},n_{jk} 表示分配给该虚拟机的 CPU 频率、内存和网络带宽.本文的调度模型如图 1 所示.

图 1 中,调度层的主要部件为:一个滚动窗口(rolling horizon)、资源监控器(resource monitor)、任务调度分析器(schedulability analyzer)和资源调配器(resource adjuster).滚动窗口容纳新到达任务和等待调度的任务;资源监控器监控系统的状态,为任务调度提供底层资源的信息支撑;任务调度分析器负载调度滚动窗口中的任务到虚拟机上;当云计算数据中心的负载发生变化时,资源调配器将触发来动态伸缩资源,包括虚拟机(VM)和主机(host).另外,每个虚拟机上只放置一个正在执行的任务(ET).

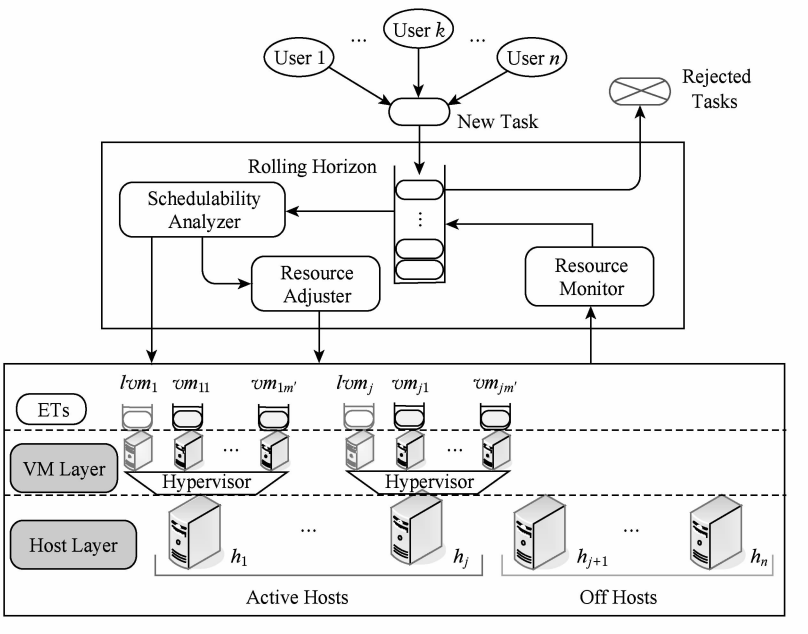


Fig. 1 The scheduling architecture

图 1 调度架构

2.2 任务模型

对于云服务系统,用户提交的应用具有高动态和随机性.本文针对的应用是实时、非周期和独立的任务,表示为 $T=\{t_1,t_2,\cdots,t_n\}$.这些任务的到达时间和截止期只有在任务到达之后才获得.对于一个任务 $t_i\in T$,可以表示为 $t_i=(a_i,l_i,d_i)$,这里 a_i,l_i,d_i 分别表示任务 t_i 的到达时间、长度和截止期.由于虚拟机 CPU 处理能力的异构性,假设 $et_{i,jk}$ 为任务 t_i 在虚拟机 vm_{jk} 上的执行时间,如式(1)所示[4,22]:

$$et_{i,jk}=l_i/f_{jk}. \tag{1}$$

2.3 主机的能耗模型

类似文献[4,22],本文主要考虑主机 CPU 的能耗.主机 CPU 的能耗分为 2 部分:空闲能耗(p_j^{idle})和活跃能耗(p_j^{active}).活跃能耗与电压的平方和工作频率成正比.主机 CPU 的活跃能耗 p_j^{active} 可以表示为

$$p_j^{\text{active}} \propto v_j^2 \times f_j, \tag{2}$$

其中,v_j 和 f_j 分别表示主机的电压和频率.

另外,由于 CPU 的电压与频率是线性关系,那么主机的活跃功率可表示为

$$p_j^{\text{active}} \propto f_j^3. \tag{3}$$

假设 s_j 为主机静态功耗的比例,$p_j^{\max}$ 为主机的最大功耗,主机的功耗可表示为

$$p_j = p_j^{\text{idle}} + p_j^{\text{active}} = s_j \times p_j^{\max} \times y_j^t + \frac{(1-s_j) \times p_j^{\max}}{(f_j^{\max})^3} \times f_j^3, \tag{4}$$

其中,y_j^t 表示主机 h_j 在时刻 t 的状态,如果主机处于活跃状态,则 $y_j^t=1$;否则 $y_j^t=0$.

假设执行任务集合 T 的开始时刻和结束时刻分别是 st 和 et,主机 h_j 的总能耗 tec_j 表示为

$$tec_j = \int_{st}^{et} (s_j \times p_j^{\max} \times y_j^t + \frac{(1-s_j) \times p_j^{\max}}{(f_j^{\max})^3} \times f_j^3) \mathrm{d}t. \tag{5}$$

那么,处理完任务集合 T,云数据中心 m 台主机的总能耗 tec 可以表示为

$$tec = \sum_{j=1}^{m} tec_j. \tag{6}$$

2.4 数学模型

根据以上描述,本文调度问题的数学模型可以总结如下:

$$\max \sum_{k=1}^{|H|} \sum_{j=1}^{|VM_k|} \sum_{i=1}^{|T|} x_{i,jk} / |T|;$$

$$\min \ tec = \sum_{j=1}^{m} tec_j$$

$$\text{s.t.} \begin{cases} ft_{i,jk} \times x_{i,jk} \leqslant d_i, \ \forall t_i \in T; \\ \sum_{k=1}^{|H|} \sum_{j=1}^{|VM_k|} x_{i,jk} \leqslant 1, \ \forall t_i \in T; \\ f_k^{\max} - \sum_{j=1}^{|VM_k|} c_{jk} \geqslant 0, \ \forall h_k \in H; \\ r_k - \sum_{j=1}^{|VM_k|} r_{jk} \geqslant 0, \ \forall h_k \in H; \\ n_k - \sum_{j=1}^{|VM_k|} n_{jk} \geqslant 0, \ \forall h_k \in H; \\ x_{i,jk} \in \{1,0\}; \end{cases} \tag{7}$$

其中,$x_{i,jk}$ 表示任务 t_i 与虚拟机 vm_{jk} 的映射关系,如果任务 t_i 映射到虚拟机 vm_{jk} 上,则 $x_{i,jk}=1$;否则,$x_{i,jk}=0$.

该模型首先最大化任务的完成率,然后最小化数据中心的能量消耗.第 1 个约束条件表示任务需要在其截止期内完成才满足用户要求的服务质量;第 2 个约束条件表示任务是不可分割的,1 个任务最多只能在 1 台虚拟机上执行.第 3~5 个约束分别表示在任何时刻主机分配给虚拟机的 CPU 能力、内存和带宽不能超过主机相应的资源能力.

3 调度算法 STARS

本节首先提出一个具有启动时间感知的虚拟机扩展策略,这里的启动时间包括启动主机和创建虚拟机的时间,然后将以上策略集成到滚动优化中,形成算法 STARS,用于调度实时任务、主机和虚拟机.

3.1 机器启动时间感知的虚拟机扩展策略

为了缓解启动主机和创建虚拟机的时间开销对任务截止期的影响,本节提出以下策略:在每台活跃主机上放置 1 台应急虚拟机,该虚拟机处于完全空闲状态时,占用主机少量的内存(比如 512 MB)和可以忽略不计的 CPU 资源.当需要使用应急虚拟机时,使用以下 1 个步骤增加应急虚拟机的 CPU 能力.

步骤 1. 如果活跃主机上剩余的 CPU 资源大于应急虚拟机的资源需求,那么,增大主机的频率,然后为应急虚拟机配置 CPU 资源,如图 2 所示.

如图 2①所示,在任务分配到应急虚拟机(lash-up VM)之前,应急虚拟机处于空闲状态,占用主机很少的 CPU 资源,并且主机有足够的剩余 CPU 资源;当准备分配任务到应急虚拟机上时,如图 2②所示,迅速增大主机的工作频率,为应急虚拟机提供 CPU 资源;最后,当应急虚拟机完成任务后,应急虚拟机释放 CPU 资源,如图 2③所示.显然,在主机有足够空闲资源的情况下,步骤 1 能够避免创建新虚拟机的时间开销对实时任务开始时间的延迟,从而提高系统保证实时任务时效性的能力.

步骤 2. 如果步骤 1 不可行,并且主机上存在某些虚拟机的执行任务和等待任务可以忍受应急虚拟机完成任务造成的延迟,那么把那些可以忍受延迟的虚拟机的 CPU 资源暂时转移给应急虚拟机使用,待应急虚拟机完成任务后,释放应急虚拟机的 CPU 资源,并返还给其他虚拟机.图 3 为步骤 2 的示意图.

如图 3①所示,主机的 CPU 资源大部分已经被分配给虚拟机 1(VM1)和虚拟机 2(VM2),主机没有足够的 CPU 资源供应急虚拟机使用.假设虚拟机 2 上的执行任务和等待任务能够忍受应急虚拟机执行任务造成的延迟,如图 3②所示,将虚拟机 2 的 CPU 资源转移给应急虚拟机.待应急虚拟机完成任务以后,把 CPU 资源返还给虚拟机 2,如图 3③所示.

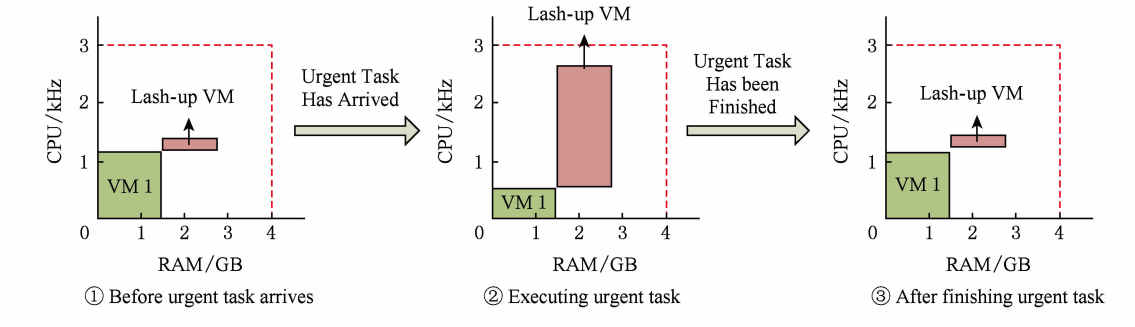


Fig. 2 An example of Step1

图 2 步骤 1 的示意图

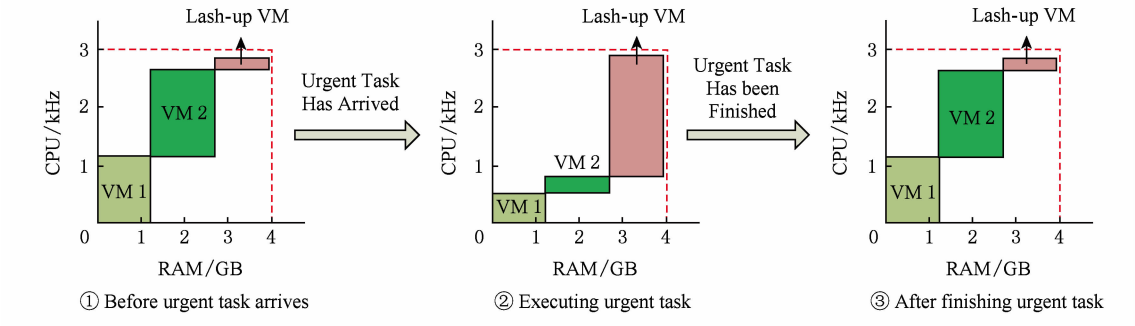


Fig. 3 An example of Step2

图 3 步骤 2 的示意图

步骤 3. 如果以上 2 个步骤都不可行,并且主机上存在某些虚拟机,能够忍受启动主机后迁移它们到其他主机的时间延迟,那么,把这些虚拟机的 CPU 资源转移给应急虚拟机使用. 同时开启 1 台关闭的主机,待关闭主机开启后,再将那些能够容忍延迟的虚拟机,迁移到刚启动的主机上,然后恢复它们的 CPU 资源供给. 图 4 为步骤 3 的示意图.

如图 4①所示,主机 1 没有足够的剩余资源给

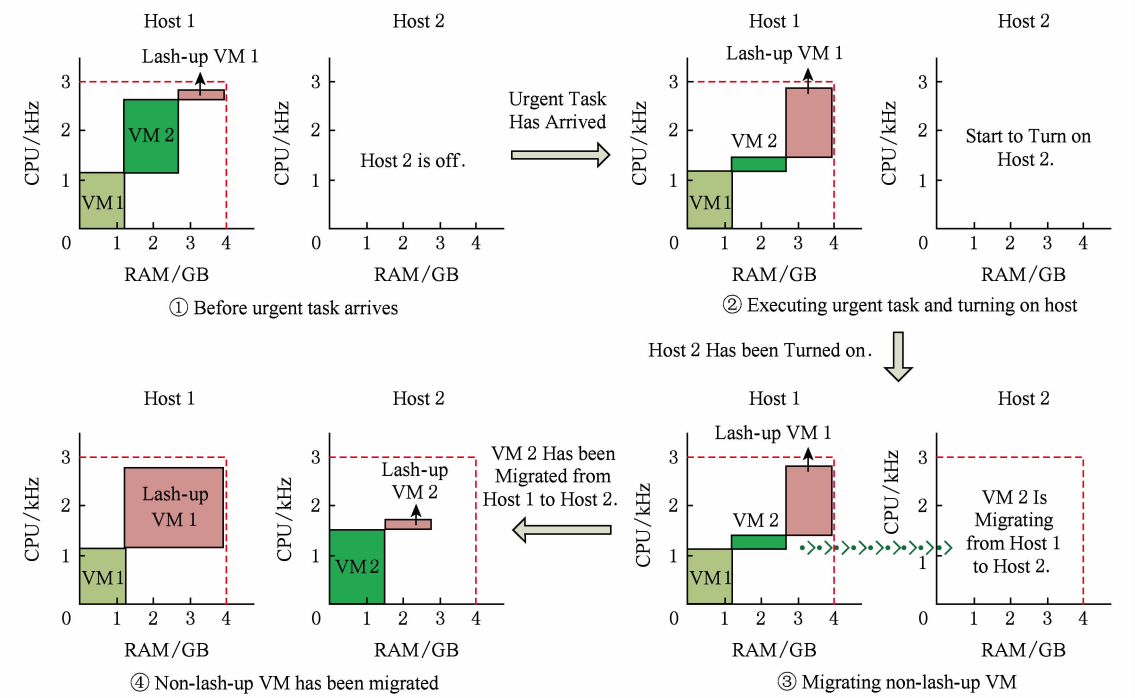


Fig. 4 An example of Step3

图 4 步骤 3 的示意图

应急虚拟机 1 使用,同时,应急虚拟机 1 需要运行的时间比较长,主机 1 上的其他虚拟机都不能忍受运行应急虚拟机带来的延迟. 如果虚拟机 2 能够忍受启动主机和迁移虚拟机的时间开销,那么把虚拟机 2 的 CPU 资源转移给应急虚拟机 1 使用,同时开启 1 台关闭的主机,如图 4②所示. 待关闭的主机启动后,把虚拟机 2 从主机 1 迁移到主机 2,如图 4③所示;待虚拟机 2 成功迁移到主机 2 后,主机 2 为虚拟机 2 供给 CPU 资源,同时在主机 2 上创建 1 台应急虚拟机,如图 4④所示.

3.2 算法设计

本节将启动时间感知的扩展策略集成到滚动优化,形成具有启动时间感知能力的调度算法 STARS,来实现实时任务和资源的调度. 在算法 STARS 中,所有未开始执行的任务都在滚动窗口 RH 中等待,直到准备被执行时,才被从 RH 中传输到虚拟机. 当新任务到达时,算法 STARS 将重新调度滚动窗口中的所有等待任务和新到达任务,如算法 1 所示.

算法 1. STARS:启动时间感知的调度算法.

输入:实时任务集合 T,可用虚拟机集合 $vmSet=\bigcup_{j=1}^{|H|}VM_j$,主机集合 H;

输出:每个任务与虚拟机的映射,虚拟机与主机的映射.

① $RH\leftarrow\varnothing$; /* 滚动窗口中的任务集合 */
② while 新任务 $t_i\in T$ 到达 do
③ 删除 RH 中等待任务与虚拟机的映射关系,并更新虚拟机的就绪时间;
④ 将任务 t_i 加到 RH;
⑤ 根据任务截止期对 RH 中所有任务进行非递减排序;
⑥ for 任务 $t_i\in RH$ do
⑦ 调用函数 $ScheduleTask()$ 调度任务 t_i;
⑧ end for
⑨ end while

算法 1 中,当新任务到达时,滚动窗口中所有任务的调度方案将被删除,并且更新系统中每台虚拟机的就绪时间(算法 1 中行③),就绪时间是指虚拟机完成所有分配给它的任务的时间;然后,把新到达的任务加入到滚动窗口 RH(行④);接着,对滚动窗口中所有任务按照它们截止期进行非降序排序(行⑤);最后,调用函数 $ScheduleTask()$ 为滚动窗口中的每个任务生成调度方案(行⑥~⑧).

函数 $ScheduleTask()$ 的伪代码如算法 2 所示.

算法 2. 函数 $ScheduleTask()$.

输入:任务 t_i,可用虚拟机集合 $vmSet=\bigcup_{j=1}^{|H|}VM_j$;

输出:任务 t_i 与虚拟机的映射.

① $selVM\leftarrow$NULL; $minFT\leftarrow+\infty$;
② for 系统中每台虚拟机 vm_{jk} do
③ 计算任务 t_i 在虚拟机 vm_{jk} 上的完成时间 $ft_{i,jk}$;
④ if $ft_{i,jk}\leqslant d_i$ & $ft_{i,jk}<minFT$ then
⑤ $selVM\leftarrow vm_{jk}$; $minFT\leftarrow ft_{i,jk}$;
⑥ end if
⑦ end for
⑧ if $selVM\neq$NULL then
⑨ 将任务 t_i 映射到虚拟机 $selVM$;
⑩ else /* 已有的虚拟机不能满足任务的时效性要求 */
⑪ $selVM\leftarrow ScaleUpResource()$; /* 增加虚拟机 */
⑫ if $selVM\neq$NULL then
⑬ 分配任务 t_i 到虚拟机 $selVM$ 上运行;
⑭ else /* 扩展虚拟机不成功,使用应急虚拟机 */
⑮ $selVM\leftarrow FindLashUpVM()$;
⑯ if $selVM\neq$NULL then /* 找到应急虚拟机 */
⑰ 分配任务 t_i 到虚拟机 $selVM$ 上运行;
⑱ else /* 任务没能调度成功 */
⑲ 拒绝任务 t_i;
⑳ end if
㉑ end if
㉒ end if

如算法 2 所示,函数使用 3 个递进的步骤把任务映射虚拟机上. 步骤 1:在已经开启的虚拟机中为该任务寻找 1 台目标虚拟机(算法 2 中行②~⑦),该目标虚拟机能够满足任务的时效性要求,并且使得任务的完成时间最小(行④). 如果步骤 1 不能够成功调度任务,那么转入步骤 2:调用虚拟机伸展策略(即函数 $ScaleUpResource()$,算法 4)为任务增加新虚拟机(行⑩). 如果以上 2 个步骤都不能成功调度任务,那么执行步骤 3:调用函数 $FindLashUpVM()$ 来为该任务寻找 1 台应急虚拟机(行⑮). 如果存在应急虚拟机能够满足该任务的时效性要求(行⑯),那么将该任务放置到应急虚拟机上执行(行⑰),否则,拒绝执行该任务(行⑲).

函数 *FindLashUpVM*()的伪代码如算法 3 所示.

算法 3. 函数 *FindLashUpVM*().

输入:任务 t_i, 应急虚拟机集合 $lvmSet=\bigcup_{j=1}^{|H|} lvm_j$;

输出:返回选出的应急虚拟机.

① 确定应急虚拟机满足任务 t_i 截止期的最小 CPU 频率需求;
② for 系统中有空闲应急虚拟机 vm_j 的活跃主机 h_j do
③ if 主机 h_j 剩余的 CPU 频率能够满足应急虚拟机 vm_j then
④ 提高主机 h_j 的工作频率来提高虚拟机 vm_j 的频率;
⑤ 返回应急虚拟机 vm_j;
⑥ end if
⑦ end for
⑧ for 系统中有空闲应急虚拟机 vm_j 的活跃主机 h_j do
⑨ if 主机 h_j 上某些虚拟机能够忍受完成任务 t_i 的延迟 then
⑩ 转移这些虚拟机的 CPU 频率给应急虚拟机 vm_j;
⑪ 返回应急虚拟机 vm_j;
⑫ end if
⑬ end for
⑭ for 系统中有空闲应急虚拟机 vm_j 的活跃主机 h_j do
⑮ if 主机 h_j 上某些虚拟机能够忍受迁移的延迟 then
⑯ 转移这些虚拟机的 CPU 频率给应急虚拟机 vm_j;
⑰ 启动 1 台关闭的主机,然后迁移这些虚拟机;
⑱ 返回应急虚拟机 vm_j;
⑲ end if
⑳ end for

如算法 3 所示,函数 *FindLashUpVM*()使用了 3.1 节中的 3 个策略为紧急任务快速扩展虚拟机. 策略 1:如果存在某台活跃主机上剩余的 CPU 资源大于应急虚拟机的资源需求,那么,增大主机的频率,然后为应急虚拟机配置 CPU 资源来完成该任务(算法 3 中行②～⑦). 如果策略 1 不可行,转入策略 2:如果存在某台主机,它的虚拟机集合中存在某些虚拟机的执行任务和等待任务可以忍受应急虚拟机完成该任务造成的延迟,那么,把那些可以忍受延迟的虚拟机的 CPU 资源暂时转移给应急虚拟机完成该任务(行⑧～⑬). 如果之前 2 个策略都不可行,转入策略 3:如果存在某台主机,它的虚拟机集合中存在某些虚拟机,能够忍受启动主机然后迁移它们到其他主机的时间延迟,那么,把这些虚拟机的 CPU 资源转移给应急虚拟机来完成应急任务. 同时开启 1 台关闭的主机,待关闭主机开启后,再将那些能够容忍延迟的虚拟机,迁移到刚启动的主机上,然后恢复它们的 CPU 资源供给(行⑭～⑳).

函数 *ScaleUpResource*()的伪代码如算法 4 所示.

算法 4. 函数 *ScaleUpResource*().

输入:任务 t_i,虚拟机模板集合 $S=\{s_1, s_2, \cdots, s_m\}$,主机集合 H;

输出:返回增加的新虚拟机.

① $activeHostlist \leftarrow$all the active hosts in the system;
② 选择一个虚拟机模板 $s_u \in S$,在考虑创建虚拟机延迟的情况下,既能满足任务的时效性要求又具有最小的 CPU 频率需求;
③ $minFit \leftarrow +\infty$; $destinationHost \leftarrow$NULL;
④ for 活跃主机 $h_j \in activeHostlist$ do
⑤ if 主机 h_j 能够容纳虚拟机 vm^{s_u} then
⑥ 在主机 h_j 上创建虚拟机 vm^{s_u},并标识为 $vm_{jk}^{s_i}$;
⑦ 返回虚拟机 $vm_{jk}^{s_i}$;
⑧ end if
⑨ end for
/* 第 1 个策略不能创建新虚拟机,使用第 2 个策略 */
⑩ 选择一个虚拟机模板 $s_u \in S$,在考虑启动主机和创建虚拟机延迟的情况下,既能满足任务的时效性要求又具有最小的 CPU 频率需求;
⑪ $offHostlist \leftarrow$all the off hosts in the system;
⑫ for 关闭主机 $h_j \in offHostlist$ do
⑬ if 主机 h_j 能够容纳虚拟机 vm^{s_u} then
⑭ 启动主机 h_j,然后创建虚拟机 vm^{s_u},并标识为 $vm_{jk}^{s_i}$;
⑮ 返回虚拟机 $vm_{jk}^{s_i}$;
⑯ end if
⑰ end for

如算法4所示,函数 *ScaleUpResource*()使用了2个策略,来为云服务系统增加新的虚拟机.策略1是在活跃主机上创建新虚拟机(算法4中行①～⑨).如果策略1可行,那么在目标主机上创建新虚拟机(行⑤～⑧).否则,转入策略2,该策略开启1台关闭的主机,然后创建新虚拟机(行⑩～⑰).策略2的具体流程如下:首先,选择1个能够在任务截止期内完成该任务的虚拟机模板,同时考虑开启主机和创建虚拟机对该任务截止期的影响;然后,从关闭主机的队列中,选出1台能够容纳该虚拟机模板的主机(行⑬～⑯),来创建虚拟机.

当云服务系统中负载下降,并且存在某些虚拟机长时间空闲时,本文使用文献[4]中的资源收缩策略来收缩虚拟机和主机的规模,以减少系统的能耗.

3.3 时间复杂度分析

定理1. 算法STARS调度一个任务集合 T 的时间复杂度为 $O(|T|\times N_w\times(N_{vm}+N_H))$,其中,$|T|$ 表示任务集合 T 中的任务数量;N_w 表示等待任务的数量;N_{vm} 和 N_H 分别表示虚拟机和主机的数量.

证明. 计算1个任务在所有虚拟机上的开始和结束时间的时间复杂度为 $O(N_{vm})$(见算法2中行②～⑦).算法2中调用函数 *ScaleUpResource*()增加1台虚拟机的时间复杂度为 $O(N_H)$.分析如下,在算法4中,为虚拟机模板寻找1台合适的启动主机的时间复杂度为 $O(N_a)$(见算法4中行①～⑨),其中 N_a 表示启动主机的数量;为虚拟机模板寻找1台关闭的主机的时间复杂度为 $O(N_o)$(见算法4中行⑩～⑰),其中 N_o 表示关闭主机的数量.那么,算法4的时间复杂度为 $O(N_a+N_o)=O(N_H)$,即函数 *ScaleUpResource*()的时间复杂度为 $O(N_H)$.另外,算法2调用函数 *FindLashUpVM*()为任务寻找1台应急虚拟机的时间复杂度为 $O(N_a)$.那么,算法STARS调用函数 *ScheduleTask*()调度1个任务的时间复杂度为 $O(N_{vm}+N_H+N_a)=O(N_{vm}+N_H)$,由于 $N_a\leqslant N_H$.因此,算法STARS调度任务集合的时间复杂度为 $O(|T|)O(N_w)O(N_{vm}+N_H)=O(|T|N_w(N_{vm}+N_H))$. 证毕.

4 算法STARS的性能评估

为了检验算法STARS的有效性,本节通过仿真实验,将算法STARS的性能与其他2个算法进行比较,对比算法是EARH[4]和Lowest-DVFS[22].

算法EARH与算法STARS的区别在于:1)EARH在任务调度算法中没有启动时间感知的策略;2)EARH考虑运行主机在最优频率上进行节能.

算法Lowest-DVFS在满足实时任务截止期要求的条件下,把主机的工作频率调整到最低.该算法没有使用虚拟机整合策略来节能.

另外,本节选用以下2个性能指标来比较不同算法的性能.

1) 任务完成率(guarantee ratio, GR).任务在截止期内完成的比例.

2) 总能量消耗(total energy consumption,TEC).执行完一个任务集合,云服务系统中所有主机消耗的总能量.

4.1 实验设置

在本文的实验中,CloudSim[23]仿真平台被选来模拟云服务系统中的基础设施.为了模拟云服务系统中的主机参数,本文使用以下5种真实主机的参数①:PowerEdge R730,Sugon I620-G20,RH2288H V2,Altos R360,Express5800.每种主机的数量都设为2 000台.假设启动主机的时间为30 s.

另外,假设云服务系统中有6个虚拟机模板,虚拟机模板对主机的CPU频率的需求分别有1.0,1.5,2.0,2.5,3.0,3.5(单位GHz).使用虚拟机模板创建虚拟机的时间为30 s.

本节将根据Google云服务系统中真实任务的执行数据②来模拟随机任务.本组实验选择从 $timestamp=1\,468\,890$ 到 $timestamp=1\,559\,030$,共计955 626个任务的执行数据.

由于任务执行数据中没有包含任务周期(cycle)和截止期的详细信息,类似于文献[24],本节使用以下方式设置以上2个参数.

对于任务的周期,本文将使用Google云服务系统中关于任务开始时间、结束时间和主机CPU的平均利用率的数据来计算,如式(8)所示:

$$c_i=(ts_{finish}-ts_{schedule})\times U_{avg}\times C_{CPU}, \tag{8}$$

其中,ts_{finish} 和 $ts_{schedule}$ 分别表示任务开始和结束的时间戳;U_{avg} 表示执行该任务时,主机CPU的平均利用率.以上3个参数可以从Google公开的数据获取.C_{CPU} 表示主机CPU的处理能力,类似于本文其他实验中主机的参数,假设 $C_{CPU}=4.0$ GHz.

① http://www.spec.org/power_ssj2008/results

② http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1

类似于文献[4],假设任务在虚拟机上的执行时间为任务的周期与虚拟机 CPU 频率的比值,如式(9)所示:

$$\mu_{i,jk}=c_i/f_{jk}. \tag{9}$$

本节使用截止期基准(*deadlineBase*)来控制任务的截止期,任务 t_i 的截止期 d_i 的计算公式如式(10)所示:

$$d_i=a_i+(ts_{\text{finsh}}-ts_{\text{schedule}})\times deadlineBase. \tag{10}$$

4.2 任务截止期对性能的影响

为了测试任务截止期对算法性能的影响,本组实验选取任务集中的 350 000 个任务,并将它们的参数 *deadlineBase* 从 1.1 递增到 3.6,步长为 0.5. 对于任务完成率(GR)和总能耗(TEC),算法 STARS, EARH, Lowest-DVFS 的实验结果如图 5 所示:

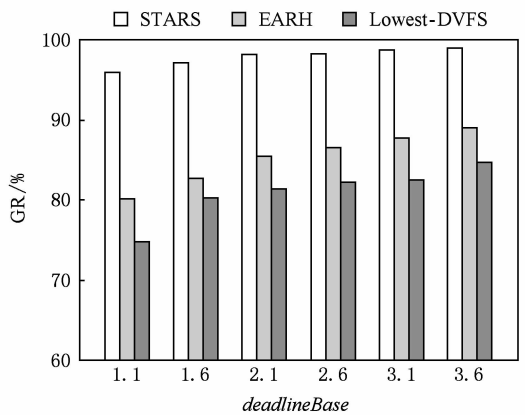


Fig. 5 Performance impact of task deadlines

图 5 任务截止期对性能的影响

从图 5(a)中可以观察到,随着 *deadlineBase* 的增加,3 个算法的任务完成率也随之上升. 这可以解释为:随着 *deadlineBase* 的增加,任务的截止期被延长,任务的时效性要求降低,从而使得更多的任务能够在截止期内完成. 另外,算法 STARS 的任务完成率平均比 EARH 高 13.48%,比 Lowest-DVFS 高 19.10%. 然而,所有算法的任务完成率还是达不到 100%. 这是由于在 Google 的任务集中,存在执行时间很短的任务,即使 *deadlineBase* 很大,这些任务的截止期也很小,不能够忍受启动主机和创建虚拟机造成的延迟. 在 Google 任务集中,执行时间小于 10 s 的任务占总任务的 10%左右,对于这些执行任务很短的任务而言,即使 *deadlineBase* 达到 3.6 时,它们的截止期也小于 36 s. 很明显这些任务的截止期小于启动主机和创建虚拟机的时间. 而算法 STARS 中的应急虚拟机可以及时开始执行这些任务,从而能够保障所有任务的截止期. 本组实验结果表明,具有机器启动时间感知能力的算法 STARS 在保障实时任务时效性方面具有优异的性能.

从图 5(b)中可以观察到,随着 *deadlineBase* 的增加,Lowest-DVFS 减少,EARH 基本不变,STARS 先增后降. 另外,算法 STARS 消耗的能量平均比算法 EARH 少 15.23%,比算法 Lowest-DVFS 少 24.47%. 本组实验结果表明,本文的虚拟机伸缩算法具有高效的节能能力.

4.3 任务数量对性能的影响

本组实验的目的是测试在不同任务规模下不同

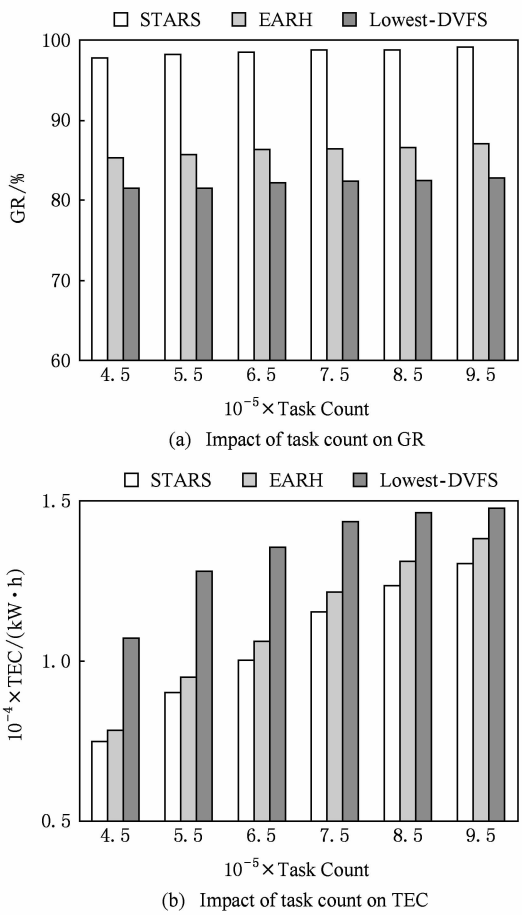


Fig. 6 Performance impact of task count

图 6 任务量对性能的影响

的算法的性能.本组实验将任务量从350 000递增到950 000,步长为200 000,设$deadlineBase=2.1$,实验结果如图6所示.

如图6(a)所示,不管任务规模如何变化,算法STARS,EARH,Lowest-DVFS的任务完成率分别稳定在99.01%,88.66%,82.06%.这是由于扩展虚拟机的过程中,启动主机和创建虚拟机的时间开销,延误了一些截止期较短的任务的截止期.另外,算法STARS的任务完成率平均比算法EARH优11.67%,比算法Lowest-DVFS优20.65%.更多的解释类似于图5(a).

如图6(b)所示,3个算法的能耗都随着任务数量线性增长.这由于在本组试验中,3个算法的任务完成率都基本保持不变如图6(a),当总任务量增长时,云服务系统完成的任务数量也随之线性增长,云服务系统的能耗与它的工作量也是正比关系.另外,算法STARS消耗的能量比算法EARH少14.98%,比算法Lowest-DVFS少24.22%.

5 结束语

本文针对虚拟机扩展过程中,启动主机和创建虚拟机的时间开销影响实时任务时效性的问题,提出了启动时间感知的虚拟机扩展策略,借助单个虚拟机CPU频率可以动态调整的能力,将机器启动时间对新到达实时任务的影响转移到其他能够容忍该延迟的任务.然后,将以上策略集成到滚动优化中,实现对实时任务和资源的动态调度,降低机器启动时间对实时任务时效性的影响.最后,将本文的算法STARS部署到CloudSim模拟平台中.实验结果显示,算法STARS能够有效提高虚拟化云保障实时任务时效性的能力,减少云服务系统的能量消耗.

参 考 文 献

[1] Chen H, Zhu X, Guo H, et al. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment [J]. Journal of Systems and Software, 2015 (99): 20-35

[2] Koomey J. Growth in data center electricity use 2005 to 2010 [OL]. [2016-08-01]. http://www.missioncriticalmagazine.com/ext/resources/MC/Home/Files/PDFs/Koomey_Data_Center.pdf

[3] Pettey C. Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions [OL]. [2016-08-01]. https://www.gartner.com/newsroom/id/503867,2007,14:2013

[4] Zhu X, Yang L T, Chen H, et al. Real-time tasks oriented energy-aware scheduling in virtualized clouds [J]. IEEE Trans on Cloud Computing, 2014, 2(2): 168-180

[5] Hermenier F, Lorca X, Menaud J M, et al. Entropy: a consolidation manager for clusters [C] //Proc of ACM SIGPLAN/SIGOPS Int Conf on Virtual Execution Environments. New York: ACM, 2009: 41-50

[6] Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing [J]. Future Generation Computer Systems, 2012, 28(5): 755-768

[7] Li K, Tang X, Li K. Energy-efficient stochastic task scheduling on heterogeneous computing systems [J]. IEEE Trans on Parallel and Distributed Systems, 2014, 25(11): 2867-2876

[8] Mei J, Li K, Li K. Energy-aware task scheduling in heterogeneous computing environments [J]. Cluster Computing, 2014, 17(2): 537-550

[9] Ma Y, Gong B, Sugihara R, et al. Energy-efficient deadline scheduling for heterogeneous systems [J]. Journal of Parallel and Distributed Computing, 2012, 72(12): 1725-1740

[10] Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment [J]. IEEE Trans on Parallel and Distributed Systems, 2013, 24(6): 1107-1117

[11] Ma Yan, Gong Bin, Zou Lida. Duplication based energy efficient scheduling for dependent tasks in grid environment [J]. Journal of Computer Research and Development, 2013, 50(2): 420-429 (in Chinese)
(马艳, 龚斌, 邹立达. 网格环境下基于复制的能耗有效依赖任务调度研究[J]. 计算机研究与发展, 2013, 50(2): 420-429)

[12] Wei Liang, Huang Tao, Chen Jianya, et al. Workload prediction-based algorithm for consolidation of virtual machines [J]. Journal of Electronics & Information Technology, 2013, 35(6): 1271-1276 (in Chinese)
(魏亮, 黄韬, 陈建亚, 等. 基于工作负载预测的虚拟机整合算法[J]. 电子与信息学报, 2013, 35(6): 1271-1276)

[13] He Yanxiang, Yu Tao, Chen Yong, et al. Green demands oriented data allocation for embedded systems [J]. Journal of Computer Research and Development, 2015, 52(1): 94-104 (in Chinese)
(何炎祥, 喻涛, 陈勇, 等. 面向嵌入式系统绿色需求的数据分配方法[J]. 计算机研究与发展, 2015, 52(1): 94-104)

[14] Hsu C H, Slagter K D, Chen S C, et al. Optimizing energy consumption with task consolidation in clouds [J]. Information Sciences, 2014, 258: 452-462

[15] Zhou Jingcai, Zhang Huyin, Zha Wenliang, et al. User-aware resource provision policy for cloud computing [J]. Journal of Computer Research and Development, 2014, 51(5): 1108-1119 (in Chinese)

(周景才, 张沪寅, 查文亮, 等. 云计算环境下基于用户行为特征的资源分配策略[J]. 计算机研究与发展, 2014, 51(5): 1108-1119)

[16] Corradi A, Fanelli M, Foschini L. VM consolidation: A real case based on openStack cloud [J]. Future Generation Computer Systems, 2014 (32): 118-127

[17] Ding Youwei, Qin Xiaolin, Liu Liang, et al. An energy efficient algorithm for big data processing in heterogeneous cluster [J]. Journal of Computer Research and Development, 2015, 52(2): 377-390 (in Chinese)
(丁有伟, 秦小麟, 刘亮, 等. 一种异构集群中能量高效的大数据处理算法[J]. 计算机研究与发展, 2015, 52(2): 377-390)

[18] Li Qiang, Hao Qinfen, Xiao Limin, et al. Adaptive management and multi-objective optimization for virtual machine placement in cloud computing [J]. Chinese Journal of Computers, 2011, 34(12): 2253-2264 (in Chinese)
(李强, 郝沁汾, 肖利民, 等. 云计算中虚拟机放置的自适应管理与多目标优化[J]. 计算机学报, 2011, 34(12): 2253-2264)

[19] Yin Xiaolong, Li Jun, Wan Mingxiang. Virtual machines scheduling algorithm based on improved NSGA II in cloud environment [J]. Computer Technology and Development, 2014, 24(8): 71-75 (in Chinese)
(殷小龙, 李君, 万明祥. 云环境下基于改进 NSGA II 的虚拟机调度算法[J]. 计算机技术与发展, 2014, 24(8): 71-75)

[20] Xiao Peng, Hu Zhigang, Qu Xilong. Energy-aware scheduling policy for data-intensive workflow [J]. Journal on Communications, 2015, 36(1): 1-10 (in Chinese)
(肖鹏, 胡志刚, 屈喜龙. 面向数据密集型工作流的能耗感知调度策略[J]. 通信学报, 2015, 36(1): 1-10)

[21] Li K, Tang X, Yin Q. Energy-aware scheduling algorithm for task execution cycles with normal distribution on heterogeneous computing systems [C] //Proc of the 41st Int Conf on Parallel Processing (ICPP). Piscataway, NJ: IEEE, 2012: 40-47

[22] Kim K H, Beloglazov A, Buyya R. Power-aware provisioning of cloud resources for real-time services [C] //Proc of the 7th ACM Int Workshop on Middleware for Grids, Clouds and e-Science. New York: ACM, 2009: 1-7

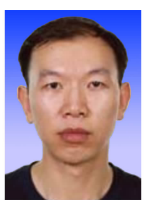
[23] Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms [J]. Software: Practice and Experience, 2011, 41 (1): 23-50

[24] Moreno I S, Garraghan P, Townend P, et al. An approach for characterizing workloads in Google cloud to derive realistic resource utilization models [C] //Proc of the 7th IEEE Int Symp on Service Oriented System Engineering (SOSE). Piscataway, NJ: IEEE, 2013: 49-60

Chen Huangke, born in 1990. Received his BSc and MSc degrees from the College of Information and System Management at National University of Defense Technology (NUDT), China, in 2012 and 2014, respectively. Currently, he is PhD candidate at the same college and university. His main research interests include scheduling, resource management and data security in cloud computing.

Zhu Jianghan, born in 1972. Received his MSc and PhD degrees in management science and technology from NUDT, China, in 2000 and 2004, respectively. His main research interests include combinatorial optimization, cloud services, space-based information systems, and satellite application information link.

Zhu Xiaomin, born in 1979. Received his PhD degree in computer science from Fudan University, Shanghai, China, in 2009. He is currently associate professor in the College of Information Systems and Management at NUDT, China. His main research interests include scheduling and resource management in distributed systems.



Ma Manhao, born in 1974. Received his MSc and PhD degrees in management science and technology from NUDT, China, in 2004 and 2008, respectively. He is currently associate professor in the School of Information Systems and Management at NUDT. His main research interests include combinatorial optimization, and space-based information systems.

Zhang Zhenshi, born in 1979. Received his MSc degree from NUDT, China, in 2010. His main research interests include cognitive computing and decision neuroscience, planning & scheduling, and multiple satellites.