基于应用特征的 PaaS 弹性资源管理机制

魏 豪 周抒睿 张 锐 杨 挺 王千祥
(北京大学信息科学技术学院软件所 高可信软件教育部重点实验室 北京 100871)

摘 要 如何在保障服务质量的前提下,尽可能地节省服务器资源,是云计算环境中弹性资源管理面临的主要问题之一.目前关于云计算环境中弹性机制的研究多数集中在 IaaS 层,且较少考虑不同应用的具体特征.文中提出了一种基于应用特征的 PaaS 弹性资源管理机制 AFERM(Application Feature based Elastic Resource Management mechanism),主要贡献包括:(1)定义了包含资源开销信息与请求率变化信息的应用特征;(2)设计了一种基于应用执行单元的请求分配机制,在请求数量较大时,将它们划分给多个资源开销相对固定的应用执行单元分别处理;(3)基于应用特征将不同类型的应用搭配部署.作者在自主开发的 PaaS 平台 PAE(Peking university Application Engine)上验证了该机制的可行性和有效性.实验证明,AFERM 能够在保证服务质量的前提下,有效地减少服务器的使用:与对比策略相比,AFERM 平均可以减少 28.3%的虚拟机占用,最多可以减少 50%.

关键词 云计算;PaaS;弹性;应用特征
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.00223

Application Feature Based Elastic Resource Management Mechanism on PaaS

WEI Hao ZHOU Shu-Rui ZHANG Rui YANG Ting WANG Qian-Xiang
(*Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

Abstract How to save various resources and keep service quality for clients is one of the main challenges faced by the elastic resource management of cloud computing platforms. Existing elastic management approaches focus mainly on IaaS, and consider few of the features of different applications. In this paper, we presented an application feature based elastic resource management mechanism (AFERM) on PaaS. The benefits of AFERM include: (1) defining application feature which is composed of changing pattern of client requests and resource consumptions of the application; (2) proposing an execution unit based requests distribution approach, which divides large number of requests and sends them to multiple execution units with fixed resource consumptions; and (3) deploying applications onto virtual machines based on their features. We evaluated the feasibility and validity of AFERM on our own PaaS, PAE (Peking university Application Engine), and found that AFERM can effectively reduce the number of servers, while keeping the service quality. Compared with the exiting approaches, AFERM can reduce 28.3% of virtual machines on average, with the highest of 50%.

Keywords cloud computing; PaaS; elasticity; application feature

收稿日期:2014-05-12;在线出版日期:2015-05-19.本课题得到国家"八六三"高技术研究发展计划项目基金(2013AA01A213)、国家"九七三"重点基础研究发展规划项目基金(2011CB302604)和国家自然科学基金(61421091)资助.**魏 豪**,男,1986 年生,博士研究生,主要研究方向为云计算平台软件.E-mail:whasic@gmail.com.**周抒睿**,女,1989 年生,硕士研究生,主要研究方向为云计算平台软件.**张 锐**,男,1989 年生,硕士研究生,主要研究方向为云计算平台软件.**杨 挺**,男,1989 年生,硕士研究生,主要研究方向为云计算平台软件.**王千祥**,男,1970 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为软件工程、系统软件.

1 引 言

PaaS 作为云计算[1]的三种平台交付模型之一，支持用户基于 PaaS 开发应用，并将应用程序部署到 PaaS 平台上. PaaS 平台的用户不必关心网络、硬件服务器、操作系统、数据库管理系统、应用服务器等应用的运行环境细节，就可以配置应用的托管环境，并部署所开发的应用. 目前互联网上已经出现了许多典型的 PaaS 平台，例如 Google 的 GAE (Google App Engine)、SalesForce 的 Force.com、Microsoft 的 Azure、VMware 的 CloudFoundry 等. 国内相关产业界也在这个领域进行了尝试，例如新浪的 SAE、百度的 BAE 等.

对于云计算服务的提供者来说，他们的主要任务是保证用户的服务质量，并尽可能地减少系统资源开销. 采取弹性的资源管理机制，可以在保证服务级别协议(Service Level Agreement)的前提下，有效地提高系统的资源利用率，从而减少运维成本[2-4].

目前的弹性资源管理机制研究主要集中在 IaaS 层，并且较少考虑平台上应用的特性. 例如：AGILE[5] 通过预测应用请求率，保证为虚拟机的增加及启动提供充分的时间，并通过快速复制虚拟机的方法保证应用的服务质量. 但是，该方法主要考虑的对象是计算密集型应用，也就是 CPU 密集型应用，没有考虑应用其他方面的特性. Gandhi 等人[6] 提出了一种动态的容量管理方法来减少运行的服务器，从而节省资源. 但是他们并没有考虑应用对多种资源的消耗以及应用在资源开销上表现出的不同特性.

PaaS 平台上的 Web 应用通常是由不同的人员开发、部署的. 往往这些应用对各种资源的使用情况差异较大，而且接受的请求往往随时间表现出不同的特性. 针对这种情形，本文提出了一种基于应用特征的 PaaS 弹性资源管理机制(Application Feature based Elastic Resource Management mechanism, AFERM). AFERM 综合考虑 PaaS 平台的资源使用情况和 Web 应用的特征，在保障服务质量的前提下尽可能地减少系统的资源开销.

本文的主要贡献包括：

(1) 定义了面向 PaaS 平台的应用特征. 通过挖掘应用的访问日志，找出应用请求率随时间的变化规律，以实现对该应用请求率的预测. 结合对应用所在系统环境的监测记录分析，建立起涵盖应用请求率变化情况以及各项资源开销等指标的应用特征模型.

(2) 提出了一种基于应用执行单元的请求分配机制，在请求数量较大时，将它们划分给多个资源开销相对固定的应用执行单元分别处理，有效地减少了弹性操作的频率，降低了弹性操作带来的额外开销.

(3) 提出了基于应用特征的搭配部署方案，将多种在资源开销方面有不同特征的应用部署在一起，均衡、充分地使用服务器资源，提高了资源利用率.

本文第 2 节回顾与本文相关的研究工作；第 3 节介绍应用特征及其获取方法；第 4 节从应用执行单元、弹性操作时机和弹性操作方案等方面对 AFERM 进行详细介绍；第 5 节通过实验对本文提出的方法进行评估；第 6 节总结全文并给出下一步的研究方向.

2 相关工作

近年来，人们对云平台的弹性资源策略进行了大量的研究. 为了实现云平台的弹性机制，需要根据应用的资源需求变化情况及时调整应用的部署方案. 本节将简略介绍与该问题相关的研究工作的进展情况.

已有的研究工作大多基于应用的请求率来优化资源的供给方案. 其中有的通过实时的请求率变化情况，判断当前应用的运行情况，动态地增减系统资源[6]；有的通过预测应用的请求率或资源占用情况来达到提前供给资源、保证服务质量的目的[2,7-12]；还有一些工作同时依据预测和实时响应来为应用提供弹性资源[13-14].

Gandhi 等人[6] 提出了一种动态的容量管理方法，可以显著地减少正在运行的服务器的数量，以节省资源和能源. 该方法的重点在于实时地计算系统的负载，并保守地进行服务器的关闭，从而为爆发的负载留出余量. 参照该方法提出的基于实时负载进行动态资源管理的思路，我们设计了一个基于实时请求率的弹性策略，并与我们本文提出的弹性资源方法进行了实验对比.

Gandhi 等人[13] 在他们的另外一个工作中，采用了反馈与预测结合的资源提供方法，将服务器在逻辑上划分为两部分，一部分用来处理预测模块的基础负载，另一部分用来处理实时的多余负载. 他们

的两个工作都仅考虑了服务器的计算能力，并没有考虑多种资源的消耗和应用的特性.

SOPRA 框架[7]通过对应用请求率的预测，预先为应用分配相应的资源，以保证服务质量，优化资源配置，达到节省能源、提高资源利用率的目的. 但是，该方法提出的调整策略只是开启关闭服务器，这个调整力度对于 PaaS 平台上的 Web 应用来说过大；同时该方法只关注一种有固定逻辑的网购应用，不具备通用性.

Gong 等人[2]使用信号处理技术来获取应用资源占用相关特征，然后通过马尔可夫链预测应用未来的资源开销，他们只是针对 CPU 等单一资源使用情况进行了考虑，忽略了多种资源的搭配使用问题.

Krioukov 等人[8]的 NapSAC 工具通过分析监测数据获得应用的请求率，通过预测请求率的方法来预先为应用分配运行资源，提前增加或删减服务器池里的服务器. 该工作比较了多种预测算法，如 Last Arrival 算法、滑动窗口平均(Moving Window Average，MWA)算法、指数加权平均法和线性回归的预测算法等，并得出结论：MWA 算法和线性回归算法的预测效果最好，比始终保持服务器开启的 AlwaysOn 方法节省资源. 但是，在被对比的 AlwaysOn 方法中，作者无法知道请求率的峰值何时出现，因此该对比结果说服力较弱.

另外的一些工作关注了系统资源的快速分配方面，降低资源实时分配的时间和性能开销，从而快速对负载的变化进行响应，以提高资源利用率.

Xiao 等人[15]在他们的另外一项工作中，设计并实现了一个可对资源进行自动伸缩的 IaaS 平台，该方法只适用于 CPU 密集型应用，对于 PaaS 上大量的在资源开销方面不同的应用而言，该方法没有充分考虑它们的差异. 针对 CPU 密集型应用，该方法提出了一种基于 CPU 占用情况的弹性策略，系统依据 CPU 开销情况执行弹性操作. 我们参照该方法实现了一个基于 CPU 占用情况的弹性策略，并与我们本文提出的方法进行了实验对比.

Gong 等人[16]的工作对应用进行了基于向量的特性描述，但在其实际系统的实验验证里，并没有对 CPU、I/O 等多种资源的消耗情况同时进行考虑，其应用部署策略也没有采取资源互补的搭配部署方式.

Qin 等人[17]的工作关注了分布式系统上的 I/O 密集型应用，分析了 I/O 密集型应用的特点和原因，提出了针对 I/O 密集型应用的两种调度算法. 但是该方法对资源的搭配使用考虑的不够，如果系统的 I/O 资源不是性能的瓶颈的话，也很难证明该方法的有效性.

综上所述，目前的相关研究工作较少有对平台上应用的特性进行综合考虑，它们或者忽视了应用通常是对多种资源进行竞争，只局限地考虑了单一资源，或者在实施弹性资源分配方案时没有对应用在资源使用方面的差异加以考虑，又或者忽视了应用请求随时间变化体现出的重要规律. 在本文中，我们提出了一种基于应用特征的 PaaS 弹性资源管理机制，综合考虑了 PaaS 平台的特性和 Web 应用的特征，在保证应用服务质量的前提下，尽可能地提高平台资源利用率、降低服务器使用数量.

3　应用特征获取技术

本节首先解释什么是应用特征，并给出其形式化描述，然后介绍应用特征获取方法.

3.1　什么是应用特征

部署在 PaaS 平台上的不同应用，可能由于用户需求的不同、业务逻辑的不同、以及开发者实现方式的差别，在资源占用方面往往存在很大的差异. 例如，一个在线报表数据处理系统对报表数据进行计算，往往会占用较多的 CPU 资源，而一个以静态数据为主的个人博客系统，则需要更多的磁盘读写时间来读写文章、图片等数据. 此外，应用对不同种类资源的需求与该应用请求情况的关系也表现出了不同的特征，例如，一个装载在 Tomcat 中的应用在启动阶段占用的内存逐步增加，但启动完毕后其内存占用便基本维持一个稳定的数值上. 而它的 CPU 占用或 I/O 开销会随请求率变化而改变. 换言之，此应用的内存占用几乎不受请求率影响，而 CPU 占用及 I/O 开销与请求率具有较强的相关性.

本文主要关注与资源占用相关的 Web 应用特征：(1)资源消耗特征，即应用实例的各类资源开销随请求率的变化情况；(2)请求率变化特征，即应用实例接收到的请求率随时间的变化特性. 请求率的变化通常会影响到应用对系统资源的需求情况.

应用的资源消耗特征主要通过对资源使用情况的监测得到. 通过对较长时间内监测的数据进行统计分析，可以归纳出应用资源消耗的特征.

应用的请求率变化特征可以通过对其访问日志

分析得到.通过周期性地从日志中采样统计应用在某段时间内的平均请求率,可以归纳出应用的请求率变化特征,并进一步基于请求率变化特征对应用将来的请求率变化情况进行预测.

3.2 应用特征的形式化描述

3.2.1 资源消耗特征的形式化描述

运行在 PaaS 平台上的应用会消耗系统资源.这些资源主要包括 CPU 时间、磁盘时间、内存占用等.这些资源的消耗量往往随应用的请求率有规律地变化,我们将应用资源消耗随请求率的变化规律定义为应用的资源消耗特征 RC(Resource Consumption).具体地,我们将 RC 表示为应用资源消耗与其请求率之间的函数关系:

$$RC=\{C_1(r),\cdots,C_n(r)\},$$

其中 n 代表资源种类数,$C_i(r)(1\leq i\leq n)$代表第 i 种资源消耗随请求率 r 的变化函数,特别地,规定 $C_1(r)$、$C_2(r)$和 $C_3(r)$分别表示 CPU 时间、磁盘时间和内存空间的占用情况.

如图 1 即为一个具体应用 A 的资源消耗随请求率的变化情况.图中横轴表示请求率,单位为"请求数每秒(req/s)";纵轴表示资源消耗情况,均转换为百分数的形式(单个虚拟机上各资源总量均表示为 100%).可以看出应用 A 的 CPU、I/O、内存等资源占用率与请求率存在明显线性关系(实验中我们使用了固态硬盘,如果是普通硬盘的话磁盘时间占用率与请求率之间可能是非线性关系),并可拟合得到具体的线性函数.因此根据定义,应用 A 的资源消耗特征表示为

$$RC=\{1.02r+9.98,0.38r+3.64,6\}.$$

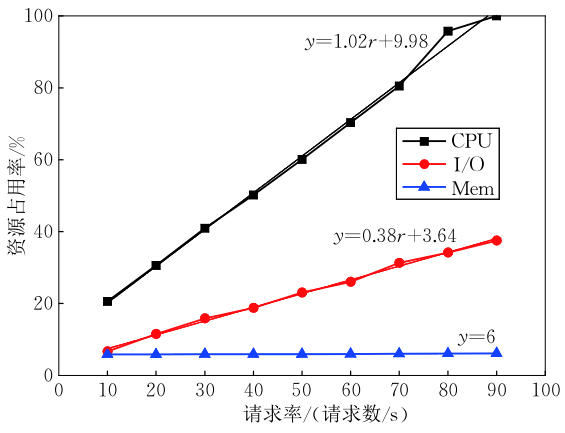


图 1 一个具体应用 A 的资源消耗特征

为了将应用的资源消耗特征方便地应用到平台环境中,我们还对平台系统资源的使用情况进行了形式化描述.

通常一个 PaaS 平台由多台物理机构成,而每台物理机又划分为若干虚拟机.因此,我们可以将 PaaS 视为多个虚拟机的集合.本文中,我们假设所有虚拟机都是同构的,它们具有相同种类相同性能的系统资源.每个虚拟机都有其系统资源向量:

$$\boldsymbol{SR}=\{R_1,\cdots,R_n\},$$

其中 $R_i(1\leq i\leq n)$代表该虚拟机第 i 种资源的使用情况.

3.2.2 请求率变化特征的形式化描述

应用的请求率变化特征 RV(Request Variability)是应用在各个单位时间内的平均请求率随时间的变化规律.我们通过傅里叶变换分析和描述应用的请求率变化规律.

傅里叶变换在声学、电信、电力系统、信号处理等诸多领域都有广泛应用,它将时间序列数据转换为频率序列数据,以便了解序列的频率构成,从而能够把握序列的主要特征,方便对其进行进一步地处理.具体地,可将时间序列 x_t 展开为傅里叶级数:

$$x_t=\sum_{j=1}^{k}[a_j\cos(2\pi f_j t)+b_j\sin(2\pi f_j t)]+A_0,$$

其中 k 为周期分量个数,A_0 为误差项,f_j 为频率,a_j、$b_j(1\leq j\leq k)$为各频率成分对应的振幅.

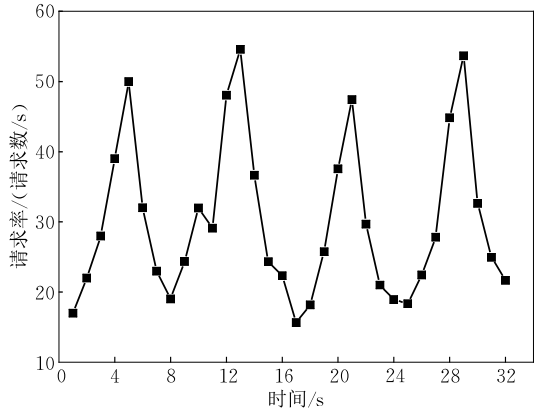
我们借助傅里叶变换分析应用的请求序列,了解它的主要频率成份及其对应振幅,从而获取应用的请求率变化规律.具体地,应用的请求率变化特征 RV 表示为对其请求序列进行时频变换后的主要频率成分的频率及振幅构成的元组的集合:

$$RV=\{(f_j,a_j,b_j)\mid j\in[1,m]\},$$

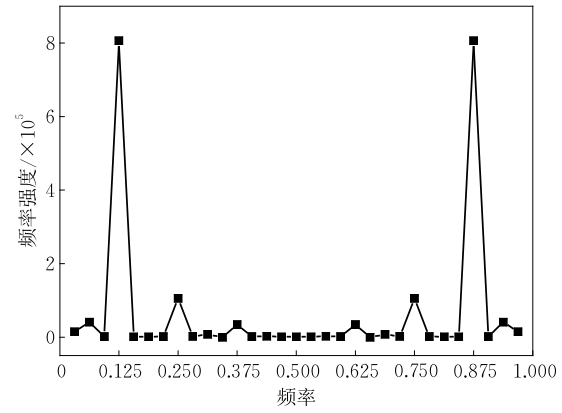
其中 m 为请求序列主要频率成分的个数.

如图 2(a)展示了一个具体应用 B 在一段时间内的请求率随时间变化的情况,图中横轴表示时间,纵轴表示请求率,单位为"请求数每秒(req/s)".对其进行傅里叶变换得到图 2(b)所示的频谱图,图中横轴表示频率,纵轴表示频率强度,由于离散傅里叶变换的共轭对称性,我们只需关注图中前半部分结果.由图 2(b)可知强度较大的频率有 0.125、0.25 和 0.375,它们构成了应用 B 请求率变化序列的主要频率成分.可以得到它们的振幅分别为(−218,−54)、(77,26)和(−42,−19),因此根据定义,应用 B 的请求率变化特征表示为

$$RV=\{(0.125,-218,-54),(0.25,77,26),(0.375,-42,-9)\}.$$



(a) 请求率变化



(b) 频谱图

图 2　一个具体应用 B 的请求率变化及频谱图

基于应用请求率变化特征，我们可以对应用的请求情况进行预测，从而能够预先地对其进行资源调配，在保证服务质量的前提下提高系统资源利用率.

3.3　应用特征的获取

一个新应用被部署到 PaaS 平台时，必须经过一个监测和日志分析的过程，才能让平台获取到其应用特征，从而更好地指导应用的部署和资源分配. 我们将应用的部署划分为单独部署和搭配部署两个阶段：(1) 当应用被初次部署到云平台时，需将其单独部署到一个虚拟机上，对虚拟机和应用进行监测，并分析应用访问日志，待该应用运行状态稳定，获取其应用特征；(2) 基于应用特征将该应用与其他应用搭配部署到服务器上，并适时进行弹性操作，以实现优化. 在部署搭配阶段，我们还会继续通过监测和日志分析，实时更新应用特征.

3.3.1　资源消耗特征的获取

在单独部署阶段，系统对运行在独立虚拟机上的应用进行监测，记录其各项资源的消耗情况，这里我们借鉴了 Shao 等人[18] 的研究工作. 经过一段时间(具体时间长度可根据应用被访问的频度进行设定，访问频繁的应用时间较短，反之则相反)的运行，应用运行状态趋于稳定，我们由监测得到的应用的各类资源的消耗情况，拟合出其随请求率变化的变化函数，从而归纳得到应用的资源消耗特征 RC.

随着请求率的增加，应用所在虚拟机会有某项系统资源消耗首先超过阈值(通常设为单个虚拟机上该资源总量的 80%)，我们定义应用此时的请求率为应用的请求率上限，记为 R_{ul}. 例如，图 1 中应用 A 请求率达到 70 req/s 的情况下，CPU 开销会首先达到 80%的阈值，因此可得该应用 R_{ul} 为 70 req/s. R_{ul} 将为搭配部署阶段应用执行单元的确定提供依据.

实验表明(详见 5.1 节)，应用在单独部署阶段表现出的资源消耗特征可以应用到搭配部署阶段.

3.3.2　请求率变化特征的获取

日志分析是一种对应用特征进行学习和挖掘的有效手段[19-20]. 因此我们通过分析应用在运行过程中产生的访问日志，来获取其请求率变化特征. 在单独部署阶段，系统通过日志记录应用的访问情况，待应用运行状态趋于稳定，对应用访问日志进行分类和统计，得到单位时间内应用的平均请求率. 然后我们对请求率序列进行傅里叶变换，得到其主要频率成分对应的频率、振幅等，从而获取到应用的请求率变化特征 RV. 基于 RV，我们对应用请求率进行预测.

根据请求率变化特征反映出的应用的历史请求信息，系统可以预测应用未来的请求情况，从而能够预先地为应用进行资源调配，在保证服务质量的前提下提高资源利用率.

应用本身的请求特征对预测精度有较大影响，如果请求率变化周期性好、规律明显，那么预测精度自然较高，反之则相反. 因此，预测请求率的方法更适用于请求率变化规律明显的应用.

4　基于应用特征的 PaaS 弹性机制

基于在单独部署阶段获取到的应用特征，我们以执行单元为单位，将应用搭配部署在 PaaS 平台上，并根据一定的触发条件，在应用运行过程中进行弹性操作——扩展或伸缩，动态地为其分配所需资源、灵活地调整其部署. 本文提出的弹性资源管理机制包括 3 个要素：

(1) 执行单元

执行单元是被限定了接收到的最高请求率的应用实例. 我们以执行单元为单位，对应用进行部署及

后续弹性操作,能够减少弹性操作次数,降低弹性操作带来的额外开销.

(2)弹性操作时机

新应用进入搭配部署阶段,就可能触发扩展操作.在应用运行过程中,我们预测其下一时段请求率,并对应用及其运行环境实时监测,获取平台整体的资源使用情况和应用响应时间等一系列监测数据.当预测结果或监测数据满足特定条件时,触发相应的扩展或伸缩操作.

(3)弹性操作方案

由(2)中预测结果或监测得到的平台服务器运行状态、应用响应时间等确定了具体的弹性操作类型后,根据应用的资源占用特征以及应用的请求状况,确定应用执行单元增删位置等具体操作细节,并通过搭配部署的方式,实施弹性操作方案.

4.1 应用执行单元

为了能提供可靠、高效的应用服务,PaaS 平台上应用的服务往往由应用的多个实例共同承担完成,并且通常单个应用实例不会独自占尽虚拟机上的全部资源.具体地,应用实例可实现为"代码加中间件"的形式,即将应用代码与其运行所依托的中间件(如应用服务器等)打包为一个整体,然后部署到虚拟机上运行.平台须将用户的请求根据一定的规则分发给各个应用实例,由各个实例借助所在服务器提供的资源,完成对请求的响应和处理,并将结果返回给用户.

基于轮询的负载平衡是一种较为常见的请求分发方式.它通过轮询,将用户的请求平均地转发给各个应用实例.这种方法实现简单,但在 PaaS 平台的弹性机制中有显著的缺点:由于单位时间内每个应用实例所承担的请求数量不固定,一个应用实例的状态发生改变,可能会影响整个服务器集群上相应应用的所有实例状态,进而引发不必要的弹性操作.

如图 3 所示,假设某应用 C 在 VM1、VM2 和 VM3 三台虚拟机上分别部署了一个实例,某一时刻其接收到的用户请求共需要 6 单位的系统资源来进行处理,基于轮询的负载平衡方法会将这些请求平均分发给 3 个应用实例,因此每个应用实例需提供 2 单位的系统资源.在弹性机制下,由于 VM3 的资源利用率过低,系统会进行收缩操作,将 VM3 关闭,并将它上面的实例承担的请求转发给其他虚拟机上的应用实例.因此,VM1 和 VM2 上的应用实例需共同承担该应用 6 单位的资源需求,即各自提供 3 个单位的资源.如此,两个虚拟机的资源占用都升高了,而由于 VM1 的资源占用过高超过了阈值,又将引发一次扩展操作,需在其他虚拟机上增加应用实例以缓解 VM1 过高的负载压力.上述过程表明,使用基于轮询的负载平衡方法进行请求分发过程中,局部(如本例中 VM3)应用实例状态变化往往会影响平台其他部分应用实例的状态,从而引起弹性操作的"连锁反应",带来更多的额外开销.

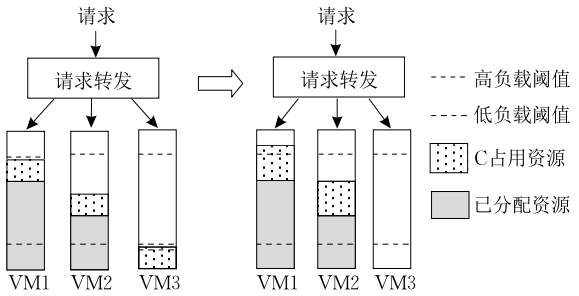


图 3 基于轮询的负载平衡在弹性机制上的应用

为了解决上述问题,我们设计了应用"执行单元"作为应用服务的基本单位,来代替基于轮询的负载平衡方法中接收请求率大小不固定的应用实例.一个应用的执行单元是指限定了最高请求率 R_{max} 的应用实例.如果一个应用执行单元接受的请求率达到 R_{max},则称其处于满负荷状态,否则为不满状态.特别地,我们将接受请求率为零的执行单元称为空执行单元.在每个请求分发周期内,用户请求被分发给应用的若干执行单元承担,且通常只有一个处于不满状态.例如,假设某应用的执行单元最大请求率 R_{max} 设为 60 req/s,某一时刻用户对该应用的请求率达到 150 req/s,则分别分配 60 req/s 的请求给 2 个执行单元使之达到满负荷状态,剩余 30 req/s 的请求分配给 1 个执行单元.

如果应用的全部执行单元都处于满负荷状态仍不足以承载所有的用户请求,则将超出执行单元承载能力的请求推迟至下一分发周期接受处理,如果下一分发周期的请求数仍超过执行单元承载能力,则继续推迟处理,以此类推.上述推迟操作会导致应用平均响应时间增长,而我们的弹性资源管理机制会持续监测平均响应时间,并在其超过特定阈值时增加执行单元以提高承载能力.

基于执行单元的请求分配方式保证了应用大部分实例接收到的请求率都处于较为稳定的状态,各自的请求率不会相互影响,解决了基于轮询的负载平衡方法中各应用实例请求率之间的耦合性,从而防止出现的一个应用实例状态改变而影响整个服务器集群上相应应用所有实例状态的现象,减少了触

发的弹性操作次数，降低了由此带来的资源开销.

图 4 是我们基于应用执行单元的请求分配方法，与图 3 的前置条件完全相同，应用 C 在 VM1、VM2 和 VM3 三台虚拟机上各部署了 1 个执行单元，它们承担的请求各有 2 单位的资源开销，且均处于满负荷状态. 当系统进行收缩操作关闭 VM3 时，需要将它上面的执行单元"整体"地迁移到其他虚拟机上. 在这个例子中执行单元被迁移到了资源占用还未超过阈值的 VM2 上. 上述过程只触发了一次弹性操作，相比基于轮询的负载平衡方法，其弹性操作次数较少，因此弹性操作带来的额外开销较低.

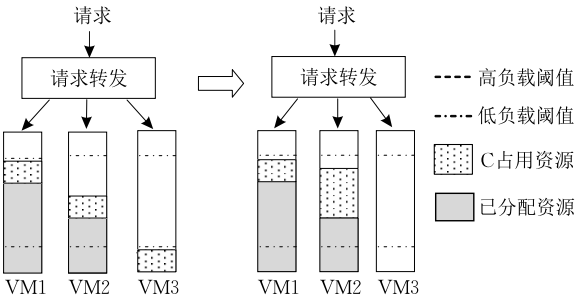


图 4　基于应用执行单元的请求分配方法

应用执行单元本质上仍为应用实例，因此实现形式也与后者相同，即"代码加中间件". 只是在对其转发请求时，会由转发服务器限定其接收到的最高请求率为 R_{max}. R_{max} 决定了单位时间内单个执行单元可以接收到的最大请求数，进而决定了单个执行单元的资源开销. 因此，为应用执行单元确定合适的 R_{max} 值至关重要. 我们基于 3.3 节方法获取到的应用特征以及应用的请求率上限 R_{ul} 来确定 R_{max}. 令 $R_{max}=R_{ul}/k$，其中 k 为正整数，将确定 R_{max} 的问题转化为确定 k 值.

如果 k 过小，则意味着满载的执行单元承载了较大的请求率，需要占用更多的资源量，会难以找到合适的其他应用的执行单元与其共同使用服务器的资源，从而丧失搭配部署的灵活性；如果 k 过大，则意味着满载的执行单元承载了较小的请求率，所需资源量较少，便能在同一个虚拟机上部署较多数目的应用执行单元，但由于处理器在接受各个执行单元的请求时要不断地进行进程的调度，执行单元数目过多会导致进程调度频繁，从而增大调度开销. 因此，k 值的确定需在保证搭配部署灵活性和系统调度开销之间做一个权衡. 另外，不同应用的 k 值也会不同：对于资源开销较大的应用，其 k 值应相对较小，否则需较多执行单元才能满足其资源需求，增大调度开销. 因此 k 值的大小需结合应用特征确定.

4.2　弹性操作时机

将完成单独部署阶段的应用搭配部署到平台上，会触发扩展操作，为该应用在服务器上增加一个或多个执行单元. 而在应用运行过程中，我们根据其历史请求率预测它在未来一段时间内的请求率，预先地为其确定所需的系统资源，并通过触发弹性操作对其进行资源调配，在保证服务质量的前提下提高资源利用率. 同时，考虑到基于预测的弹性操作有时不能很好地处理突发请求率变化(例如，"春运"期间铁路售票系统 12306 在开放售票的一瞬间可能会有数以百万计的并发请求[21])，我们会对平台整体的资源使用情况、应用响应时间等进行监测，并将它们作为弹性操作的触发条件，以实时应对突发请求率变化. 我们将上述所有可能触发弹性操作的情况——应用进入搭配部署阶段以及预测结果或监测数据满足特定条件定义为弹性操作时机. 具体地，我们按照弹性操作类型——扩展或收缩，设定弹性操作时机如下：

(1) 扩展时机

① 应用进入搭配部署阶段. 基于在单独部署阶段获得的应用特征，我们将应用搭配部署到平台服务器上. 此时，触发扩展操作，为应用创建一到多个执行单元，具体创建数量由该应用的请求率预测结果决定，我们将在后文详述.

② 应用平均响应时间过高. 当监测到应用平均响应时间超过阈值时，意味着该应用拥有的系统资源不足以处理目前的请求，部分请求被推迟处理，因此需为该应用增加执行单元.

③ 预测请求率过高. 当预测发现某应用下一时间窗的请求率大于当前时段其所有执行单元所能处理的最大请求率时，意味着需要增加执行单元以保证服务质量，即进行扩展操作.

(2) 收缩时机

① 系统资源利用率过低. 应用请求降低会导致该应用出现多个不满执行单元(含空执行单元)，从而导致虚拟机资源利用率降低，因此有必要进行收缩操作，移除空执行单元并将多个资源利用率低的虚拟机上的执行单元合并到一台虚拟机上. 但为了避免突发式的应用请求率下降引发不必要的收缩操作，我们只在下一预测时间窗开始前分析各虚拟机的资源利用率，并结合对该虚拟机上所有应用请求率的预测结果，计算下一时间窗该虚拟机的资源利用率，如果发现系统资源利用率持续低于阈值，则进行收缩操作，将该虚拟机上的执行单元迁移合并到另一台虚拟机上，并在此过程中对空执行单元进行

移除,以提高系统资源利用率.

② 预测请求率过低.当预测发现某应用下一时间窗的请求率小于其 $N-1$(N 为应用当前时段的执行单元数)个执行单元所能处理的最大请求率时,则可以在开始下一时间窗时关闭一定数量的执行单元以达到节省资源的目的,进行弹性收缩.

由于我们使用了基于应用执行单元的请求分配机制,对应用实例能接收的最高请求率 $R_{\max}$ 进行了限定,并且在转发请求时,保证一个应用尽可能多的执行单元处于满负荷运行状态(期待通过收缩操作使得每个应用至多有一个不满状态的执行单元).因此,对于应用进入搭配部署阶段导致的扩展操作及预测请求率过高或过低导致的弹性操作,我们可通过以下方法确定需增加或减少的执行单元数量.

假设应用 A 当前在平台上运行了 N(对于应用进入搭配部署阶段的情况,$N=0$)个执行单元,则期待至少 $N-1$ 个执行单元处于满负荷状态,即请求率达到 $R_{\max}$,并且此时 A 能接受的总请求率至多为 $N\times R_{\max}$.设预测得到 A 下一时段请求率为 r,如果 $r>N\times R_{\max}$,此时需进行弹性扩展,且需增加 $(r/R_{\max}-N)$ 个执行单元;如果 $r\leq(N-1)\times R_{\max}$,此时需进行弹性收缩,且需减少 $(N-r/R_{\max})$ 个执行单元.

4.3 弹性操作方案

由 4.2 节我们确定了弹性操作类型——扩展或收缩,接下来我们对具体弹性操作方案进行实施.在这个过程中,无论是扩展操作还是收缩操作,我们都会对多种不同应用的执行单元进行搭配部署:对于扩展操作,新创建的执行单元与虚拟机上已有应用执行单元进行搭配部署;对于收缩操作,将较为空闲的虚拟机上的应用执行单元进行迁移合并,并对这些应用执行单元进行搭配部署.

通过搭配部署,可以均衡、充分地使用系统资源,有效提高资源利用率.应用执行单元的搭配部署是弹性操作实施的关键,因此我们将应用经过单独部署阶段获取到应用特征后,在平台上的正式部署阶段称为搭配部署阶段.

下面分别介绍扩展和收缩两种弹性操作的操作流程.

4.3.1 扩 展

在扩展过程中,需为应用执行单元找到一台合适的虚拟机进行部署,即需确定合适的部署位置.出于充分利用资源和保持部署结果稳定的考虑,我们采用如下的虚拟机查找算法为应用确定合适的部署位置:

(1)由该应用的资源消耗特征获取其满负荷执行单元的资源消耗 $RC_{R_{\max}}=\{C_1(R_{\max}),C_2(R_{\max})\}$(此处我们仅以应用 CPU 开销和磁盘开销为例,分别用 C_1、C_2 表示);

(2)列出当前所有活跃虚拟机的资源特征向量,其中第 i 个虚拟机的特征向量为 $\boldsymbol{SR}_i=\{R_1,R_2\}$($R_1$、$R_2$ 分别表示虚拟机当前的 CPU 开销和磁盘开销),并找出可以容纳该应用的所有虚拟机,即满足条件 $R_1+C_1(R_{\max})\leq 80\%$ 且 $R_2+C_2(R_{\max})\leq 80\%$(如 3.3.1 节所述,设定资源开销阈值为 80%);

(3)计算每一个虚拟机的 $\delta=|(R_1+C_1(R_{\max}))-(R_2+C_2(R_{\max}))|$,如果仅有一台虚拟机拥有最小 δ 值,则选择此虚拟机部署新应用;否则,如果有多台虚拟机 δ 值相等且均为最小值,则进行步骤(4);

(4)比较 $\alpha=|(R_1+C_1(R_{\max}))+(R_2+C_2(R_{\max}))|$,选出 α 最大的虚拟机,部署新应用.

如果找到符合上述条件的虚拟机,则在该虚拟机上创建执行单元;如果没有找到,则开启一个新的虚拟机,并部署执行单元.

选择 δ 最小值,是因为我们希望达到 CPU 消耗和磁盘消耗的均衡,这就意味着不存在一台虚拟机拥有很高的 CPU 开销和很低的磁盘开销,或者很低的 CPU 开销和很高的磁盘开销.这样有利于提高系统资源利用率,有利于下一次的部署调整.如图 5 所示,当前 VM1 的 CPU 时间开销为 20%,磁盘时间开销为 60%;VM2 的 CPU 时间开销为 40%,磁盘时间开销为 20%.现在需要给一个 CPU 时间开销为 40%、硬盘时间开销为 20%的新应用执行单

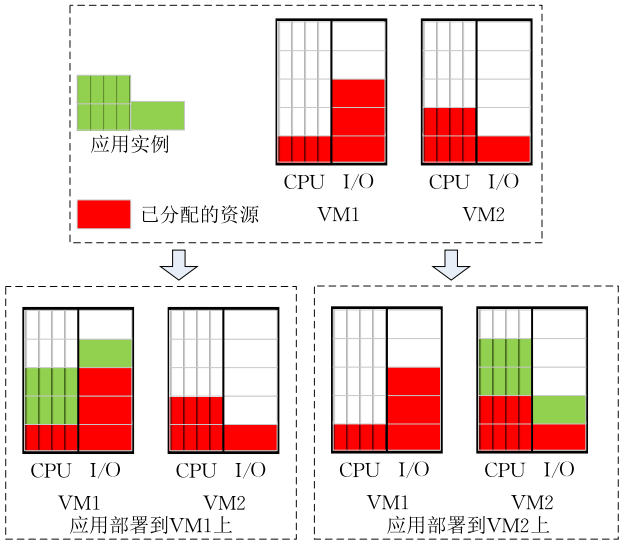


图 5 选取具有最小 δ 值的虚拟机进行部署

元寻找合适的部署位置. 若应用执行单元被部署在 VM1 上，则部署后该虚拟机 CPU 开销为 60%，硬盘开销为 80%，$\delta=|60\%-80\%|=20\%$；若应用执行单元被部署在 VM2 上，部署后该虚拟机 CPU 开销为80%，I/O开销为40%，$\delta=|80\%-40\%|=40\%$. 因此应该选择 δ 值较小的 VM1.

选择 α 最大值，是因为我们希望虚拟机的资源利用率尽可能地逼近阈值，这样就能尽可能地减少由于虚拟机资源利用率低而导致的收缩操作，降低由此带来的额外开销. 如图 6 所示，若应用执行单元被部署在 VM1 上，则部署后 VM1 的 CPU 开销为 80%，磁盘开销为 80%，$\delta=|80\%-80\%|=0$；若应用执行单元被部署在 VM2，则部署后 VM2 的 CPU 开销为 60%，磁盘开销将为 60%，$\delta=|60\%-60\%|=0$. 此时两者 δ 相等，因此进入步骤(4)的选择过程：若应用部署在 VM1 上，则 $\alpha=|80\%+80\%|=160\%$，若部署在 VM2 上，则 $\alpha=|60\%+60\%|=120\%$. 我们选择将应用执行单元部署在 α 值较大的 VM1 上.

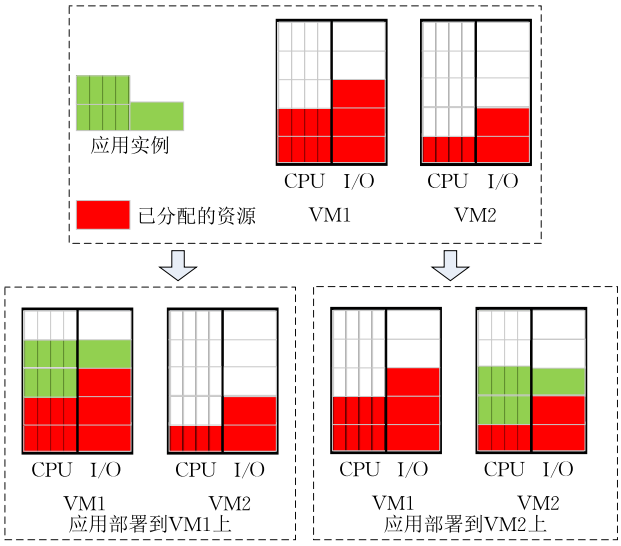


图 6　选取具有最大 α 值的虚拟机进行部署

4.3.2　收　缩

根据 4.2 节，收缩操作有两种触发时机——系统资源利用率过低或应用预测请求率过低，下面分别介绍它们各自的操作流程.

当监测到系统资源利用率过低时，需对多台虚拟机上的应用执行单元进行迁移合并，这实际上是对应用执行单元的重新部署，因此其过程与扩展操作过程类似. 例如：假设要将虚拟机 V 上的应用执行单元与其他虚拟机上的执行单元合并，则通过 4.3.1 节的虚拟机查找算法判断能否找到一台虚拟机 W 将 V 上的执行单元部署上去，如果可以，迁移 V 上的执行单元至 W.

而当预测应用请求率过低时，需在下一预测时间窗开始时移除该应用的部分执行单元：

(1) 列出所有拥有该应用执行单元的虚拟机；

(2) 找到一台其上执行单元总数(不仅是该应用的执行单元数)最少的的活跃虚拟机 V，将该虚拟机上该应用的执行单元移除；

(3) 如果虚拟机 V 上已经没有运行的应用，则将其关机；

(4) 如果 V 上该应用的执行单元数小于我们期待移除的数目，则重复进行上述操作.

5　方法评估

5.1　资源特征稳定性验证

如第 3 节所述，我们在应用的单独部署阶段获取其资源消耗特征，然后在搭配部署阶段基于应用特征进行弹性操作，以优化应用的资源分配. 为此，需通过实验验证应用在单独部署阶段的资源消耗情况是否与搭配部署阶段一致.

实验使用了 1 台 IBM 的 HS-21 Blade 服务器，并通过 VMware 硬件虚拟化系统将服务器划分为 2 台虚拟机V1 和 V2. 实验过程如下：

(1) 在虚拟机 V1 上部署待测应用 A，以每秒 5 次，10 次，15 次，…，45 次，50 次的速率向应用 A 发送请求，监测并记录 A 的 CPU 开销和磁盘开销；

(2) 在虚拟机 V2 上部署待测应用 B，以每秒 5 次，10 次，15 次，…，45 次，50 次的速率向应用 B 发送请求，监测并记录 B 的 CPU 开销和磁盘开销；

(3) 将应用 A 和应用 B 同时部署在虚拟机 V1 上，以每秒 25 次的速率向应用 A 发送请求，同时以每秒 5 次，10 次，15 次，…，45 次、50 次的速率向应用 B 发送请求，监测并记录应用 A 和应用 B 总的 CPU 开销和磁盘开销.

可测得单独部署在虚拟机 V1 上时，应用 A 在请求率为每秒 25 次时 CPU 开销为 20.35%，I/O 开销为 12.46%，记为 $CPU_A=20.35$，$IO_A=12.46$. 按照上述实验步骤，CPU 开销结果如表 1 所示，其中 rr_B 为应用 B 的请求率，单位为“次每秒”，CPU_B 为应用 B 单独部署在虚拟机上时相应请求率下的 CPU 开销，CPU_{A+B} 表示应用 A、应用 B 同时部署在虚拟机 V1 上时总的 CPU 开销，$\Delta_{CPU}=CPU_{A+B}-(CPU_A+CPU_B)$，表示应用搭配部署与单独部署时的 CPU 开

销差值.从表中可知,在负载不太高的情况下,Δ_{CPU}的值相对较低,这表明,将应用搭配部署在同一台虚拟机上时,CPU 开销与分别部署在两台虚拟机上时的 CPU 开销差别不大.表 2 为磁盘开销实验结果,可以得到与 CPU 开销类似的结论.因此在应用负载压力不大的情况下,其在单独部署阶段获取到的应用特征可应用于搭配部署阶段指导弹性操作.

表 1 CPU 资源特征稳定性验证结果

rr_B	CPU_B	CPU_A+CPU_B	CPU_{A+B}	Δ_{CPU}
5	2.98	23.33	24.39	1.06
10	6.01	26.36	27.92	1.56
15	9.32	29.67	30.88	1.21
20	12.10	32.45	34.45	2.00
25	14.89	35.24	39.06	3.82
30	18.28	38.63	43.30	4.66
35	21.48	41.83	46.87	5.04
40	24.36	44.71	49.99	5.29
45	27.54	47.89	56.28	8.38
50	30.60	50.95	60.49	9.54

表 2 磁盘资源特征稳定性验证结果

rr_B	IO_B	IO_A+IO_B	IO_{A+B}	Δ_{IO}
5	4.02	16.48	17.78	1.30
10	8.06	20.52	22.31	1.79
15	12.08	24.54	26.75	2.20
20	16.08	28.54	31.11	2.58
25	20.45	32.91	35.75	2.84
30	24.36	36.82	40.87	4.05
35	28.44	40.90	45.05	4.15
40	32.48	44.94	51.64	6.70
45	37.36	49.82	58.12	8.30
50	40.81	53.27	63.88	10.60

5.2 实验环境

下面我们在自主开发的 PaaS 平台 PAE(PKU App Engine)上对本文提出的基于应用特征的 PaaS 弹性资源管理机制 AFERM 进行实验验证.实验使用了 5 台同构的物理服务器,它们每个都通过 VMware 硬件虚拟化系统,被划分为 4 台虚拟机,因此共有 20 台同构的虚拟机.在各虚拟机上安装了实验所必须的系统软件和相关中间件,其中数据库软件为 MySQL Server 5.5.36,JVM 为 Oracle JDK 7.0.15&Oracle JRE 7.0.400.43,应用服务器为 Tomcat7.0.6,代理服务器为 Nginx1.4.1.

5.3 实验数据集

实验所使用的数据来自 NLANR① 实验数据集和百度公司提供的 BAE 3.0 真实应用监测数据集,它们实际上都是 Web 应用的访问日志.为了便于进行实验,对其进行如下处理:

(1)将每一条日志按照其行为进行区分:对于访问一个 servelet 或者一个 action 的行为,认为该应用执行了一次 CPU 密集型的操作,该条日志中的请求响应时间长度代表这次 CPU 密集型操作的任务量;对于访问一个文件的行为,认为该应用执行了一次 I/O 密集型的操作,该条日志中的请求文件读写量的值代表这次 I/O 密集型操作的任务量.

(2)统计应用访问情况,以其中连续 29 天的统计结果作为数据集.其中,前 28 天的数据作为训练集,用于获取应用特征,最后 1 天的数据作为实验集,用于验证本文提出的弹性策略.

5.4 实验过程

我们在平台上自行设计了 4 种类型的应用进行实验,它们在 CPU、I/O 等资源开销方面各不相同.所有应用的请求率变化数据均来自 5.3 节所述的 NLANR 数据集或 BAE3.0 数据集,每个应用在 24 小时内的请求次数在 30 万次到 40 万次之间,与云平台上比较典型的应用的请求情况相符.

4 个应用作为实验集的最后一天 24 小时的请求率变化趋势分别如图 7(a)~(d)所示,图中横轴表示时间,每 0.5 小时统计一次请求数,共计 48 个统计时间点;纵轴表示应用在相应的 0.5 小时内的请求数.

图 7(a)展示了以 CPU 占用为主的应用 A 的请求率变化情况,该应用请求数据来自 NLANR 数据集.其执行单元 R_{max} 为 60 req/min(次/分钟,下同),满负荷执行单元的 CPU 时间开销为 23%,磁盘时间开销为 3%.

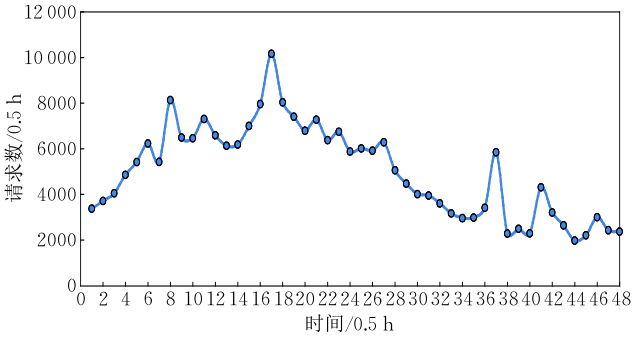
图 7(b)展示了以 I/O 占用为主的应用 B 的请求率变化情况,该应用请求数据来自 BAE3.0 数据集.其执行单元 R_{max} 为 80 req/min,满负荷执行单元的 CPU 时间开销为 4%,磁盘时间开销为 20%.

图 7(c)展示了 CPU 占用相对较多而 I/O 占用相对较少的应用 C 的请求率变化情况,该应用请求数据来自 BAE3.0 数据集.其执行单元 R_{max} 为 100 req/min,满负荷执行单元的 CPU 时间开销为 17%,磁盘时间开销为 6%.

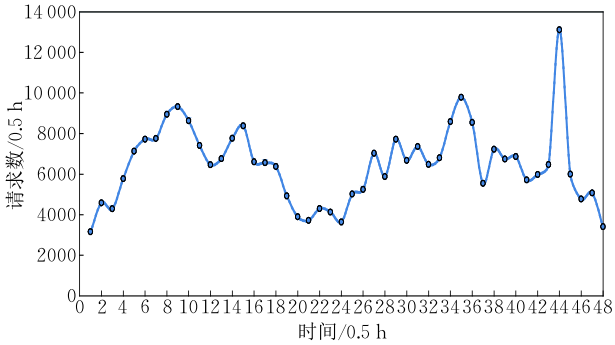
图 7(d)展示了 I/O 占用相对较多而 CPU 占用相对较少的应用 D 的请求率变化情况,该应用请求数据来自 BAE3.0 数据集.其执行单元 R_{max} 为 90 req/min,满负荷执行单元的 CPU 时间开销为 7%,磁盘时间开销为 17%.

平台上应用的总请求率由上述 4 个应用的请求

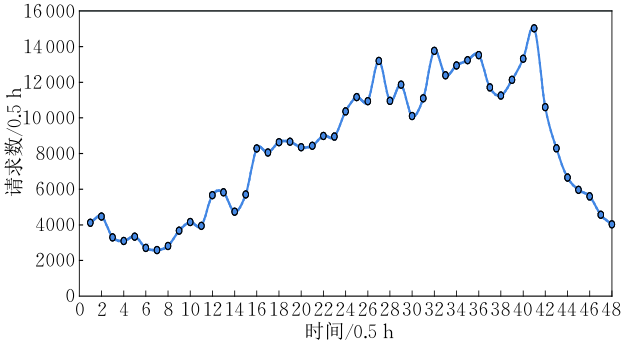
① NLANR(National Laboratory for Applied Network Research) Anonymized access logs. ftp://ftp.ircache.net/Traces/, 1995



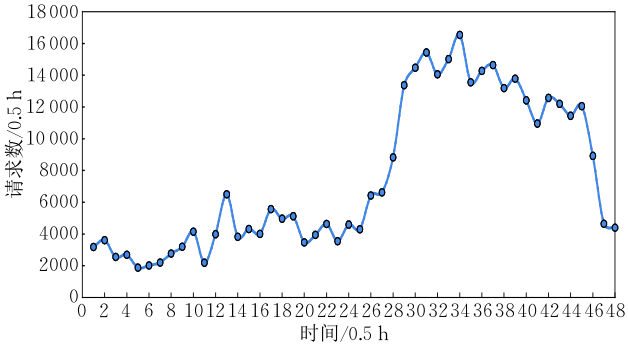
(a) 应用A请求率变化



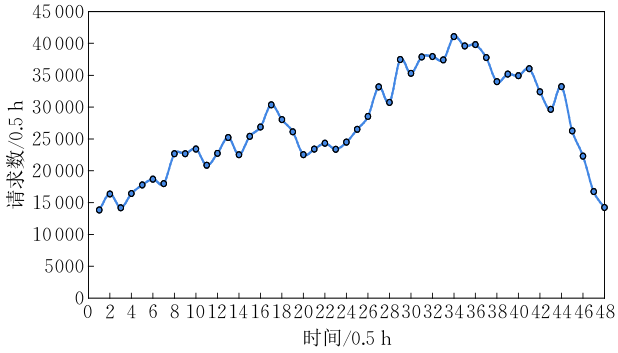
(b) 应用B请求率变化



(c) 应用C请求率变化



(d) 应用D请求率变化



(e) 实验应用总请求率变化

图 7　实验应用的请求率变化

率叠加而成，其变化情况如图 7(e)所示. 系统设定单台虚拟机的 CPU 时间开销上限为 80%，磁盘时间开销上限为 75%；每台虚拟机应用服务器数上限为 6，即最多可将 6 个应用部署到一台虚拟机上；虚拟机启动时间为 120 s，应用服务器启动时间为 30 s.

我们按照图 7 所示请求率变化情况向部署在系统上的实验应用发出访问请求，初始情况下在 5 台物理服务器上各启动 1 台虚拟机，即共启动 5 台虚拟机，在其中一台上部署应用 A、应用 B、应用 C、应用 D 的执行单元各 1 个，其余虚拟机在初始情况下空闲.

5.5　实验结果分析

本文对比了以下 4 种弹性策略，分别是：

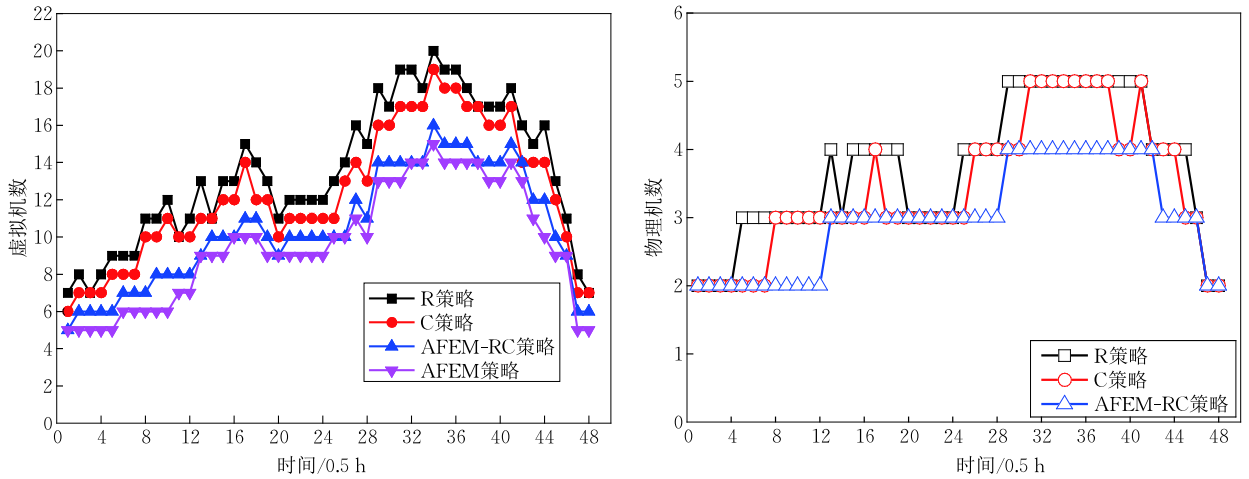
(1) 基于请求的弹性策略 R(Request Based Elastic Approach). 这是我们参照 Gandhi 等人[6]的工作实现的对比策略，以应用实时的请求率作为系统执行弹性操作的依据.

(2) 基于 CPU 占用的弹性策略 C(CPU Based Elastic Approach). 这是我们参照 Xiao 等人[15]的工作实现的对比策略，以单一的 CPU 开销情况作为系统执行弹性操作的依据.

(3) 基于应用资源消耗特征的弹性策略 AFERM-RC(Application Feature Based Elastic Management-Resource Consumption). 这是本文提出的策略，但仅考虑应用的资源消耗特征，未考虑请求率变化特征. 这个策略用来评估 AFERM 增加请求预测后的效果.

(4) 请求率预测与资源开销实时监测相结合的基于应用特征的弹性策略 AFERM. 这是本文提出的完整的策略，包含了预测机制.

按照 5.3 节所述实验过程，我们对 AFERM 与其他方法进行了对比实验，实验结果如图 8 及表 3 所示.



(a) 虚拟机占用数目 (b) 物理机占用数目

图 8 不同策略的虚拟机、物理机占用数

表 3 资源占用量和应用服务质量对比

策略名称	资源占用量 U	最大虚拟机数	请求违反数	请求违反率/%
AFERM策略	230.5	15	437	0.0334
AFERM-RC策略	251.0	16	615	0.0470
C策略	295.0	19	431	0.0329
R策略	321.5	20	343	0.0262

在图 8 中,横轴表示时间,每 30 分钟对应用所占用的虚拟机或物理机数量进行一次统计,因此 24 小时的实验周期内共有 48 个统计点;纵轴表示统计所得的虚拟机或物理机数.

由图 8 可见,基于应用特征的两种弹性策略,相比其他两种对比策略,占用更少数量的虚拟机和物理机,其中 AFERM 策略相比 R 策略最多能减少约 50%的虚拟机占用,因此基于应用特征的弹性策略更加节省服务器资源.进一步看,结合了应用请求率预测的 AFERM 策略相比于 AFERM-RC 策略,占用更少数量的虚拟机,这表明通过对应用请求率的预测,弹性机制性能得到了进一步的提高.需要注意的是,我们的实验应用都属于请求率规律相对较好的应用,对于请求规律性较差的应用,请求率预测的方法就失去了其优越性.

除了虚拟机、物理机占用数,我们的实验还对比了各种策略的资源占用量和应用服务质量.为了获得量化的资源占用情况,我们定义了一个资源占用量:

$$U=\Sigma VM\times Hour,$$

其中,VM 表示统计点对应的虚拟机数,$Hour$ 表示统计时间长度(每半小时统计一次).表 3 是对实验结果的量化分析.

从表中可知,与 R 策略相比,AFERM 平均可以减少 28.3%的虚拟机占用,最大虚拟机占用数少了 5 台,由于每台物理机都有 4 个虚拟机,因此可以直接减少 1 台物理机的使用(这在图 8(b)中也有反映).尽管由于更加"吝啬"的资源分配策略导致请求违反数增加了 94 次,但在总请求数为 1 307 888 次的前提下,这个增加是可以接受的.因此 AFERM 在保证服务质量的表现上,与 R 策略相比并无明显劣势.

与 C 策略相比,AFERM 平均可以减少 21.9%的虚拟机占用,最大虚拟机占用数少了 4 台,即可以直接减少 1 台物理机的使用.请求违反数仅仅增加了 6 次,在保证服务质量的表现上,两者表现相当.

与 AFERM-RC 相比,AFERM 在虚拟机占用上减少了 8.2%,最大虚拟机占用数也减少了 1 台,同时请求违反数减少了 178 次.可见由于引入了预测,我们的 AFERM 无论是在节省服务器资源方面,还是在保证服务质量方面,都使其得到了有效的提高.

综上所述,AFERM 能够在保证服务质量的情况下,有效减少服务器的使用.与对比策略相比,平均可以减少 28.3%的虚拟机占用,最多时可以减少 50%.

6 总 结

本文提出的基于应用特征的 PaaS 弹性资源管理机制 AFERM 通过分析应用的日志信息,对应用请求率进行预测;并结合对应用所在系统环境的监

测记录分析，建立起涵盖应用请求率变化信息和资源占用信息的应用特征模型；将不同类型的应用进行搭配部署，使不同特征的应用竞争不同的系统资源，提高各种系统资源的利用率；提出了一种更为合理的请求分配机制，将一般的应用实例转化为请求率和资源开销相对固定的应用执行单元，从而保持部署方案的相对稳定，有效地减少了弹性操作的次数，降低了弹性操作带来的资源开销. 实验证明，AFERM 在保证应用服务质量的基础上，可以有效地降低资源的使用.

现阶段 AFERM 只考虑了应用的不同特征，还没有考虑 PaaS 平台的差异，因此对于由不同性能的物理机和虚拟机构成的 PaaS 系统，目前的方法不能提供有效的支持. 今后我们将把系统的资源特征作为参数加入到特征模型中，以便更广泛地支持各种不同的 PaaS 系统. 另外，在获取请求率变化特征时可以尝试采用其他工具或方法，如小波变换等，从而提高请求预测精度，更好的指导弹性资源分配.

参　考　文　献

[1] Mell P, Grance T. The NIST definition of cloud computing. Recommendations of the National Institute of Standards and Technology, 2011, 53: 50-50

[2] Gong Z, Gu X, Wilkes J. PRESS: Predictive elastic resource scaling for cloud systems//Proceedings of the 6th International Conference on Network and Service Management (CNSM). Niagara Falls, Canada, 2010: 9-16

[3] Shen Z, Subbiah S, Gu X, Wilkes J. CloudScale: Elastic resource scaling for multi-tenant cloud systems//Proceedings of the ACM Symposium on Cloud Computing (SOCC) in Conjunction with SOSP 2011. Cascais, Portugal, 2011: 5

[4] Emeakaroha V, Brandic I, Maurer M, Breskovic I. SLA aware application deployment and resource allocation in clouds//Proceedings of the IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW). Munich, Germany, 2011: 298-303

[5] Nguyen H, Shen Z, Gu X. AGILE: Elastic distributed resource scaling for Infrastructure-as-a-Service//Proceedings of the USENIX International Conference on Automated Computing (ICAC'13). San Jose, USA, 2013: 69-82

[6] Gandhi A, Chen Y, Gmach D. Minimizing data center SLA violations and power consumption via hybrid resource provisioning//Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC). Orlando, USA, 2011: 1-8

[7] Menarini M, Seracini F, Zhang X. Green Web services: Improving energy efficiency in data centers via workload predictions//Proceedings of the 2nd International Workshop on Green and Sustainable Software (GREENS). San Francisco, USA, 2013: 8-15

[8] Krioukov A, Mohan P, Alspaugh S. NapSAC: Design and implementation of a power-proportional Web cluster. ACM SIGCOMM Computer Communication Review, 2011, 41(1): 102-108

[9] Han R, Guo L, Ghanem M, Guo Y. Lightweight resource scaling for cloud applications//Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Ottawa, Canada, 2012: 644-651

[10] Goudarzi H, Ghasemazar M, Pedram M. SLA-based optimization of power and migration cost in cloud computing//Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Ottawa, Canada, 2012: 172-179

[11] Islam S, Keung J, Lee K, Liu A. Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems, 2012, 28(1): 155-162

[12] Huang J, Li C, Yu J. Resource prediction based on double exponential smoothing in cloud computing//Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet). Yichang, China, 2012: 2056-2060

[13] Gandhi A, Chen Y, Gmach D. Optimal power allocation in server farms. ACM SIGMETRICS Performance Evaluation Review, 2009, 37(1): 157-168

[14] Iqbal W, Dailey M, Carrera D, Janecek P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computer Systems, 2011, 27(6): 871

[15] Xiao Z, Chen Q, Luo H. Automatic scaling of internet applications for cloud computing services. IEEE Transactions on Computers, 2014, 63(5): 1111-1123

[16] Gong Z, Ramaswamy P, Gu X. SigLM: Signature-driven load management for cloud computing infrastructures//Proceedings of the 17th International Workshop on Quality of Service (IWQoS). Charleston, USA, 2009: 1-9

[17] Qin X, Jiang H, Zhu Y. Improving the performance of I/O-intensive applications on clusters of workstations. Cluster Computing, 2006, 9(3): 297-311

[18] Shao J, Wang Q, Mei H. Model based monitoring and controlling for Platform-as-a-Service (PaaS). International Journal of Cloud Applications and Computing, 2012, 2(1): 1-15

[19] Xu W, Huang L, Fox A. Detecting large-scale system problems by mining console logs//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. Big Sky, USA, 2009: 117-132

[20] Xu W, Huang L, Fox A. Online system problem detection by mining patterns of console logs//Proceedings of the 9th IEEE International Conference on Data Mining. Miami, USA, 2009: 588-597

[21] Song C, Wu W. Petri net modeling of information flow in the online train ticket booking system//Proceedings of the 2012 IEEE International Conference on Automation and Logistics (ICAL). Zhengzhou, China, 2012: 517-522

WEI Hao, born in 1986, Ph. D. candidate. His current research interest is system software for cloud computing.

ZHOU Shu-Rui, born in 1989, M. S. candidate. Her current research interest is system software for cloud computing.

ZHANG Rui, born in 1989, M. S. candidate. His current research interest is system software for cloud computing.

YANG Ting, born in 1989, M. S. candidate. His current research interest is system software for cloud computing.

WANG Qian-Xiang, born in 1970, Ph. D., professor. His current research interests include software engineering, system software.

Background

In cloud computing, how to save various resources and keep service quality is one of the main challenges faced by the resource management of cloud platforms. The elastic resource management mechanism can contribute to address this challenge. However, existing elastic approaches focus mainly on IaaS and consider few of the features of different applications. For example, some researchers only consider the CPU consumption feature of applications, overlooking applications' features of other resource consumptions and request rates, which does not quite accord to the facts. Applications deployed on a PaaS are usually heterogeneous. While sharing the same resource, these applications usually have different features in resource consuming and requests' varying, all of which should be taken into consideration.

In this paper, we presented an application feature based elastic resource management mechanism (AFERM) on PaaS. When deploying and managing applications on PaaS, we considered both application requests' varying pattern and resource consumptions. Also, a novel executing unit based requests distribution approach was designed to help further reduce resource consumption. The evaluation showed that AFERM can effectively reduce the number of servers, while keeping the service quality. Compared with the exiting approaches, AFERM can reduce 28.3% of virtual machines on average, with the highest of 50%.

This work is supported by the National High Technology Research and Development Program (863 Program) of China (No. 2013AA01A213), the National Basic Research Program (973 Program) of China (No. 2011CB302604), and the National Natural Science Foundation of China (No. 61421091).