一种面向云多层应用的费用高效的资源管理方法

杨 劲[1,2] 庞建民[1,2] 齐 宁[1,2] 刘 睿[3]
(解放军信息工程大学 郑州 450001)[1] (数学工程与先进计算国家重点实验室 郑州 450001)[2]
(解放军61345部队 西安 710100)[3]

摘 要 由于部署方便、维护简单并且不需要搭建自己的私有机房,云数据中心正成为大多数互联网公司尤其是初创公司和中小规模公司部署应用程序的首选。在以基础设施为服务的云环境里,互联网公司可以根据应用程序的需要动态租赁云基础设施,从而节省预算开支,并保证应用性能。然而,在现有的业界实践中,云服务提供商提供的负载均衡和资源伸缩服务只能监控虚拟机的使用状态,并不能监控应用程序的运行状态,因此无法准确根据应用程序的服务需求自适应变换资源规模。同时,现有的文献和实践中,也很少有研究从云基础设施使用者的角度出发,为使用者节省基础设施租赁费用或高效使用已租赁基础设施。据此提出了一种面向基础设施云环境下多层应用的费用高效的资源管理方法,其在降低用户费用的同时,还能充分利用所花费用提高应用程序性能。通过仿真对所提方法业界实际使用的方法进行比较,结果表明所提方法不仅能够提高应用程序的服务质量和服务性能,也能较大地降低公司在基础设施租赁方面的费用。

关键词 云计算,费用高效,多层架构,资源伸缩,负载均衡
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.018

Cost-efficient Resource Management for Cloud Multi-tier Applications

YANG Jin[1,2] PANG Jian-min[1,2] QI Ning[1,2] LIU Rui[3]
(PLA Information Engineering University, Zhengzhou 450001, China)[1]
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)[2]
(Unit 61345, Xi'an 710100, China)[3]

Abstract Cloud data centers are increasingly becoming the first choice for many Internet enterprises, especially the newly formed and small ones, because it is convenient to deploy the applications, easy to maintain them, and unnecessary to build a private in-house infrastructure. In the cloud environments based on infrastructure as a service, Internet companies can rent the cloud infrastructure dynamically in accordance with the service requirements. In this way, they have the chance to save the costs of renting infrastructure with the performance guarantees. However, the existing methods for balancing workloads and scaling resources in the industry practice are only able to monitor the states of virtual machines, not the custom performance metrics such as the number of live application sessions. So these scaling methods cannot decide the exact resource demands according to the service requirements. In addition, there are few academic studies on reducing the renting costs and employing the billing resources efficiently in the cloud environment. On the basis of these points, this paper proposed a cost-efficient resource management approach for the cloud multi-tier applications, aiming to save infrastructure costs and improve the application performance with the billing resources. Last, we compared the proposed method with the algorithms used in practice by simulating them with the ripe benchmarks. The results indicate that our method can not only improve the quality and performance of the application, but also reduce the cost of renting infrastructure largely.

Keywords Cloud computing, Cost-efficient, Multi-tier architecture, Resource scaling, Workload balance

1 引言

对于提供在线服务的互联网公司,尤其是资金紧张的初创公司或中小规模公司,减少硬件设施费用和提升服务质量(如响应时间)是提高企业竞争力的两个重要方面。然而,减少硬件费用和提升服务体验通常相互矛盾。良好的用户体验需要高性能的硬件基础设施,要求企业增加基础设施投资;而减少基础设施费用则意味着使用较少或者性能较差的硬件设备,这将降低应用程序的性能,进而使用户体验变差。因此,研究如何高效地利用有限的基础设施费用,提升应用程序的

到稿日期:2016-03-02 返修日期:2016-07-01 本文受上海市科委科技攻关项目(13DZ1108800),国家自然科学基金(61472447)资助。
杨 劲(1983—),男,博士生,主要研究方向为高性能计算、可重构计算,E-mail:ysire@163.com;庞建民(1964—),男,博士,教授,主要研究方向为高性能计算、信息安全;齐 宁(1978—),男,博士,讲师,主要研究方向为高性能计算、动态编译。

服务质量，具有重要的现实意义。

云数据中心的出现为提高基础设施的费用利用效率提供了可能性。将基础设施作为服务(Infrastructure as a Service, IaaS)[1]的云数据中心向用户提供虚拟机(Virtual Machine, VM)租赁服务，互联网公司不必再花费大量金钱购买物理基础设施来搭建规模伸缩性较差的私有机房；并且，大多数云数据中心提供具有弹性的按需租赁服务，允许用户根据自身需要随时更改租赁虚拟基础设施的数量，省去了基础设施的预先投资和私有机房的维护费用。按需租赁服务让互联网公司能够根据应用程序服务的需求动态改变租赁的基础设施的数量，使基础设施规模总是与动态变化的服务需求相匹配，达到基础设施费用的高效利用。

2 相关工作

目前，许多云计算服务商已经提供了资源监控[2]和自动伸缩服务，根据用户租用基础设施(虚拟机实例)的规模和时间收费。例如，亚马逊弹性负载均衡器(AWS Elastic Load Balancer, ELB)[3]和自动伸缩(AWS Auto Scaling)[4]服务允许用户设定变化条件，系统按照用户设定的条件自动调整租用的虚拟机实例规模。如用户可以设定一个条件，当虚拟机实例的 CPU 利用率超过 70%时，以 3 个实例为增量，向自动伸缩资源池添加虚拟机实例；同样，也可以设置一个条件，在 CPU 利用率降低至 10%以下时，以同样的增量删除实例。这种动态资源管理服务主要存在以下不足：

1)监视的层次有限。只能监视虚拟机实例的运行状态，无法监视用户应用程序的内部运行数据，如任务队列、服务质量等；致使根据虚拟机实例的运行状态推测应用程序的服务质量，在某些情况下，虚拟机实例的运行状态与应用程序的运行状态并不一致。

2)性能提高有限。采用标准的负载均衡算法，在各个虚拟机实例间轮流分配负载，当虚拟机实例初始负载不均衡时，无法自动平衡各个实例的负载，从而导致某些虚拟机负载过多而另一些虚拟机实例负载较少，不利于应用程序整体性能的提高。

3)未考虑费用的高效利用。自动伸缩服务只能根据用户的“静态”设定测试虚拟机实例的运行状态，在满足条件时“自动”实时调整策略；并未从云消费者的角度考虑资源的充分利用和已支付费用的高效利用。

在当前的文献中，云环境下的资源管理和调度研究的主要目标是从云数据中心维护者的角度出发，通过负载均衡和资源调度来达到数据中心总体上的能量高效(Energy Efficiency)[5-6]或低能耗[7]。本文主要从云数据中心的消费者尤其是中小规模的云基础设施租赁用户(因为它们的预算开支和服务能力的矛盾最为突出)的角度出发，面向云数据中心环境下典型的商业应用程序架构——多层架构，在满足服务质量的前提下，提出一种降低基础设施租赁费用的方法。仿真实验结果表明，与当前商业实践中的资源管理方法相比，该方法不仅能够降低基础设施租赁费用，而且能够更加有效地使用租赁费用和已租赁资源来提高应用程序的性能，从而实现费用高效(Cost Efficiency)。值得注意的是，虽然能量高效和费用高效在某些情况下具有相关性，但由于算法的角度和目的都不同，它们实际上没有必然的联系。

3 总体架构

3.1 应用背景

在多层应用中，最典型和基础的是三层应用。根据文献[9-10]，三层应用程序的组成包括：

1)表示层(Presentation Layer)。指与用户交互的接口，负责服务提供和信息展示，如浏览器、移动端 App 或者命令行。

2)业务逻辑/域层(Business Logic/Domain Layer)。负责应用程序的逻辑处理，是整个应用的功能核心。

3)数据层(Data Layer)。负责和数据库、事务处理器或其他应用系统进行通信。

业务逻辑层通常包括一个或多个应用服务器，在云计算环境中，每个应用服务器对应着一个虚拟机。业务逻辑层可以通过增加或减少虚拟机数量调整应用程序的性能，同时节省费用。

按照业务逻辑层处理服务请求的方式，多层应用程序可分为两种：有状态应用程序和无状态应用程序。有状态应用程序会在服务器内存中保存会话(Session)数据(比如购物信息、用户元数据等)，以保证服务器能够识别并接着处理同一个会话发起的请求。在有多个应用服务器时，同一个会话发起的请求需要连接到同一个应用服务器，这也称为会话的粘性(Sticky Session)。相反，无状态应用程序只处理用户请求，不保存请求数据，因此也不需要粘性。

应用服务器通过为每个会话设定和刷新生命周期管理会话。当新访问请求到来时，应用服务器建立一个新的会话连接，并为其设定生命周期(如 Tomcat 默认为 20 分钟)；然后开始对会话进行生命计时。如果在计时过程中，会话发起了新的访问请求，则把计时清零，重新开始生命计时。如果会话长时间没有活动，应用程序则会把该会话从内存中清除，此时会话便会失效。在集群或多服务器系统中，会话可以在多个服务器之间通过拷贝保持同步[11-12]，以便会话请求能被系统中的任意服务器处理。

3.2 应用分层结构

在 IaaS 云环境中，为了方便管理会话和虚拟机，达到应用程序高性能和低费用的目的，在三层架构的基础上增加一个控制层(Control Layer)。系统架构如图 1 所示。

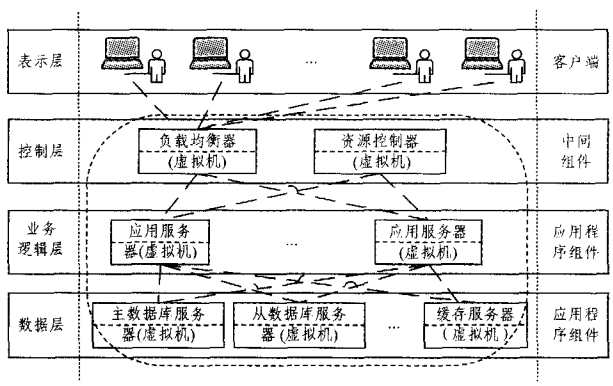

图 1 系统分层架构

从运行层次上看，表示层运行在客户端，其他三层运行在

后端的云数据中心(图中虚线框中的部分)。控制层主要包括两个中间组件:负载均衡器(Workload Balancer)和资源控制器(Resource Controller)。理论上,负载均衡器和资源控制器也应该是可以水平伸缩的,但由于运行它们的虚拟机数目远少于运行应用服务器的虚拟机数目,它们的水平伸缩对运行费用的影响可以忽略不计。

负载均衡器的主要功能是负责接收和转发服务请求,即按照一定的算法把服务请求重定向到一台合适的应用服务器,因此所有服务请求都要经过负载均衡分配应用服务器。而在处理完毕后,对服务请求的结果响应则直接返回给客户端的用户。对于有状态的会话,负载均衡实施粘性转发。对于无状态连接请求;负载均衡器按照设定的调度算法为它们分配应用服务器。

资源控制器监控虚拟机的 CPU 和 RAM 利用率、虚拟机的租赁时间和到期时间等,并根据监控数据判断虚拟机和应用程序的运行状态。对应用程序,资源控制器需要监控会话个数、生命周期、生命刷新时间等与会话有关的数据,推断会话状态和变化趋势。最终,资源控制器根据虚拟机的运行状态、付费状态以及会话的状态和变化趋势,决定在何时关闭哪个虚拟机或者购买新的虚拟机,同时满足用户性能要求和费用高效的目标。

3.3　层间交互

系统架构中,各层组件的交互主要分为用户请求交互和资源控制交互两类,分别如图 2 和图 3 所示。

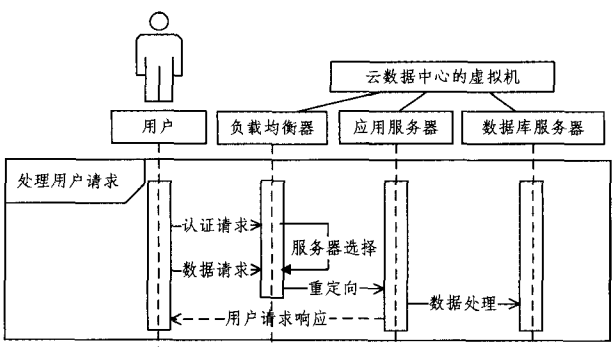

图 2　用户请求交互

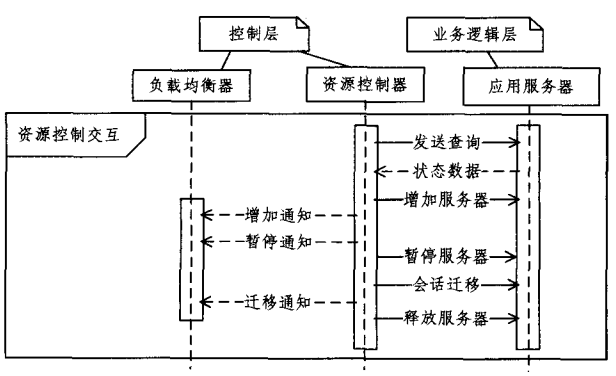

图 3　资源控制交互

在图 2 中,根据会话是否已经建立,把用户请求分为认证请求和数据请求两种。认证请求是指客户端申请建立会话的连接请求;数据请求是指会话建立后的数据查询申请。当认证请求到来时,负载均衡器无法找到该请求的连接分配信息,认为这是一个新的连接请求,因此按照均衡算法为其选择可用的应用服务器。当数据请求到来时,负载均衡器内已经存有该连接的分配信息,直接找到负责响应该连接的应用服务器,并把该数据请求重定向到这个应用服务器上。应用服务器在处理完数据请求后,直接把处理结果返回给请求客户端。

图 3 主要描述了控制层组件和业务逻辑层的应用服务器之间的信息交互。资源控制器按照一定的周期向各个应用服务器发送状态查询请求,应用服务器在收到请求后返回状态数据,如会话数、会话生命周期等。当应用程序服务能力不足而需要增加服务器时,资源控制器向服务器集群添加工作服务器,成功后通知负载均衡器,可以向新服务器分配负载。当应用程序服务能力过剩而需要减少工作服务器时,资源控制器首先通知负载均衡器某台服务器即将暂停,负载均衡器将不再向该服务器发送新的认证请求,然后资源控制器暂停该服务器。在释放应用服务器之前,资源控制器首先检查该服务器上是否还有正在被服务的会话,若有,则首先进行会话迁移,并通知负载均衡器迁移后的应用服务器,然后结束会话,释放应用服务器;否则直接释放。

3.4　资源状态转换

为了方便描述,把应用服务器的状态分为 3 类:活动(Alive)状态、暂停(Suspended)状态和已释放(Freed)状态。

当应用程序负载较低时,为了使应用服务器的数量和用户请求负载量相匹配,资源控制器会把一部分活动的应用服务器的状态置为暂停。负载均衡器将不再给处于暂停状态的服务器分配新的认证请求连接,但已有会话的数据请求将继续分配给该应用服务器,只是不再接受新会话创建请求。这样,随着用户离去,暂停状态服务器上的活动会话数将越来越少。当暂停服务器上没有活动会话或少量会话已经被迁移时,资源控制器将在合适的时机释放虚拟机。

反之,当负载较高,活动服务器性能不足时,资源控制器将优先把暂停服务器重新置为活动状态,不足时才会分配新的虚拟机。应用服务器的状态转换如图 4 所示。

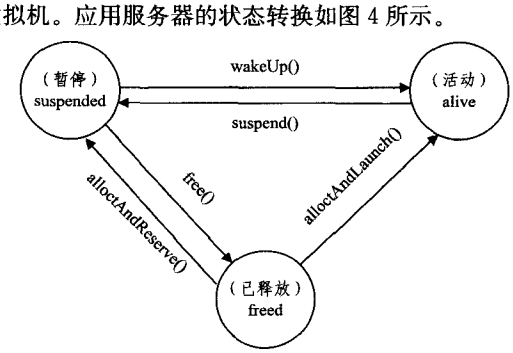

图 4　应用服务器的状态转换图

4　费用高效的实现

费用高效有两层含义:1)满足 SLA 的前提下,所花费用应该尽可能少;2)充分利用已经付费的资源向外界提供服务。

4.1　状态参数的定义与抽象

首先描述会话的活动度。假设会话的生命周期为 t。会话建立后,初始生命长度为 t,随着时间流逝,生命长度 l 逐渐缩短。当 $l=0$ 时,会话失效。在此过程中,如果该会话连接发起了数据请求,则重置 $l=t$。据此,定义会话活动度 $\alpha=f(l)$,则 $f(l)$ 应满足:

1)当 $l=t$ 时,$\alpha=1$;

2)当 $l=0$ 时,$\alpha=0$;

3)α 与 l 成反比,并且 $\alpha\in[0,1]$。

实际应用中,把会话的生命周期划分成 n 个等分区间,并用 n 个权值(分段函数)描述区间内会话的活动度。

在此基础上,定义应用服务器的活动度(activity)以及该服务器上所有会话的活动度之和。计算方法如式(1)所示:

$$activity=\sum_{i=1}^{n}\alpha_i\cdot c_i \tag{1}$$

其中,α_i 表示在第 i 个区间内会话的活动度,c_i 表示该区间内会话的个数。活动程度的值指示着应用服务器上会话的总体活动趋势,值越大,表示活动的用户或会话越多;值越小,表明用户越少或会话已经趋向于失效。

在实际中,应用服务器的最大处理能力是一个经验值,可通过反复多次测试求均值得到。处理能力与应用服务器的数目是可以相互转换的。一个运行中的应用服务器的剩余处理能力(surplus)指的是该应用服务器还能容纳新建会话的个数。在 CPU 利用率不超过阈值的情况下,剩余处理能力主要与 RAM 的剩余值有关,并且基本呈线性关系[10]。因此,可以根据 RAM 利用率估算应用服务器的剩余能力。在服务器中,会话数可以通过直接读取相关变量得到。应用程序的平均会话数表示应用程序的总体负载程度,可通过计算所有应用服务器的会话数之和与服务器个数的商得到。使用以下抽象函数表示上述处理过程。

1)*activity*( ):计算应用服务器的活动度;

2)*servToVMs*( ):把需要的服务能力转换为虚拟机的个数;

3)*surp*(*as*):计算应用服务器 *as* 的剩余处理能力;

4)*sessionNum*(*as*):计算 *as* 的会话数;

5)*sum*(*list*[1..*n*]):计算列表元素之和;

6)*average*(*sum*,*n*):计算平均数。

4.2 负载均衡

负载均衡算法采用粘性负载均衡策略,如算法 1 所示。算法首先按照 CPU 利用率的上升顺序对应用服务器列表进行排序(第 1 行);其次从头遍历列表,如果发现存在应用服务器 as_i 的 RAM 利用率不超过阈值,则停止遍历,并把新会话请求分配给该服务器;如果不存在,则把新会话请求分配给列表的第一个元素(2－10 行)。算法 1 运行在负载均衡阶段,把新建会话请求优先分给“最空闲”的应用服务器,以减少响应时间,提高用户体验。

算法 1 负载均衡算法

输入:新的会话认证请求 s_i,RAM 利用率的阈值 th_{ram},活动状态的应用服务器列表 AS_{alive}

功能:分配 s_i 到某一主机

```
1. Resort AS_alive by CPU utilization in ascending order;
2. host ← first element of AS_alive;
3. for as_i ∈ AS_alive do
4.   as_ram ← RAM utilization of as_i;
5.   if as_ram < th_ram then
6.     host ← as_i;
7.     break;
8.   end if
9. end for
10. Assign s_i to host;
```

4.3 资源按需伸缩

资源伸缩算法通过申请购买和释放虚拟机,在满足服务需求的条件下,达到费用高效。算法 2 给出了资源按需伸缩算法的基本实现。

算法 2 按需伸缩算法

输入:平均活动度的触发值 $tgr_{activity}$,应用程序平均会话数的触发值 $tgr_{avgsess}$,应用程序剩余处理能力的触发值 $tgr_{surplus}$;活动状态应用服务器列表 AS_{alive},暂停状态应用服务器列表 $AS_{suspend}$;算法两次调用的时间周期 Δ,会话生命周期分割区间的权值列表 Γ,会话生命周期分割区间的个数 n;本次算法调用的时刻 t_{cur}

功能:调整应用服务器的规模,达到费用高效

```
//初始化会话数总和、活动度总和为 0
1. sessSum ← 0;activitySum ← 0;
//计算剩余能力、平均会话数、平均活动度
2. for as_i ∈ AS_alive do
3. sessSum←sessSum+sessionNum(as_i);
4. surplus[i]←surp(as_i);
5. activitySum←activitySum+activity(as_i,n,Γ[n]);
6. end for
7. as_surplus←sum(surplus[1..i]);
8. as_avgsess←average(sessSum,i);
9. as_avgact←average(activitySum,i);
//增加活动状态服务器
10. if as_avgsess > tgr_avgsess and as_surplus < tgr_surplus and as_avgact > tgr_activity
      then
11.   ass_surplus←computeSurplus(AS_suspend);
12.   if as_surplus+ass_surplus>= tgr_surplus then
13.     Resort AS_suspend decendingly by surplus;
14.     for ass_i ∈ AS_suspend do
15.       AS_alive←AS_alive+ass_i;//改变 ass_i 状态
16.       AS_suspend←AS_suspend−ass_i;
17.       as_surplus=as_surplus+surp(ass_i);
18.       if as_surplus>=tgr_surplus then
19.         break;
20.       end if
21.     end for
22.   else //需要购买新的服务器
23.     diff=tgr_surplus−(as_surplus+ass_surplus);
24.     nVMToStart=servToVMs(diff);
25.     AS_alive←AS_alive+AS_suspend;
26.     Lanuch nVMToStart VMs and add them to AS_alive;
27.   end if
      //减少活动状态服务器
28. else
29.   if (as_avgsess< tgr_avgsess and_a s_avgact<tgr_activity) or as_surplus>tgr_surplus
      then
30.     diff=as_surplus−tgr_surplus;
31.     nVMToStop=servToVMs(diff);
32.     Resort AS_alive ascendingly by activity;
33.     AS_suspend←AS_alive[1..nVMToStop];
34.     AS_alive←AS_alive−AS_alive[1..nVMToStop];
35.   end if
      //释放虚拟机
36.   for vm_i ∈ AS_suspend do
```

```
37.    billTime←Billing time of vm_i;
38.    if billTime−t_cur<Δ then
39.      if vm_i already has sessions then
40.        Migrate sessions from vm_i and notify workload balancer;
41.      end if
42.    Terminate vm_i;
43.    end if
44.  end for
45. end if
```

算法 2 首先计算应用程序的剩余处理能力、平均会话数和平均活动度(2－9 行)，然后判断应用程序的服务能力与用户需求相比是不足(10 行)还是盈余(29 行)。当需要增加服务器时，算法总是优先使用暂停列表中的服务器(11－21 行)，不足时则申请购买新的虚拟机(22－27 行)。当需要减少服务器时，算法则会把活动程度较小的服务器移到暂停服务器列表(29－35 行)。最后计算暂停列表中每台服务器的已支付时间能否持续到算法的下次调用时间，如果不能(37－38 行)，则把该服务器上的活动会话迁移到活动服务器上，并告知负载均衡器，然后释放该虚拟机(39－43 行)；如果能够持续到下次调用时间，则让服务器继续运行。

5　实验与评估

5.1　实验环境与参数设置

本文使用 CloudSim 离散事件模拟器[13]模拟云数据中心环境，版本为 3.0.3。实验中所用虚拟机的性能特征和价格根据亚马逊 AWS EC2 日本地区 m3.medium 的 Linux 虚拟机实例进行设置。虚拟机实例按需收费的周期以小时为单位，启动和配置时间基于文献[15]中的经验数据。上述性能参数可以通过修改或继承 cloudsim 包中的 Host，Vm 和其他相关类进行设置。为了弱化数据层的资源竞争对调度算法产生的影响，使用 3 个虚拟机部署主从分布式数据库服务器，其中 1 个作为写服务器(主)，另外 2 个作为读服务器(从)，并在整个过程中不对数据层进行资源伸缩调度。表 1 列出了算法中各个参数的设置情况。

表 1　仿真实验参数

参数	值	组件	算法
th_{ram}	0.75	Workload Balancer	ALL
th_{cpu}	0.75	Workload Balancer	WCRP, Baseline
$tgr_{surplus}$	6100	Resource Controller	OURS, WCRP
$tgr_{avgsess}$	5000	Resource Controller	OURS
$tgr_{activity}$	4100	Resource Controller	OURS
Δ	10sec	Resource Controller	ALL
n	20	Resource Controller	OURS

云多层应用的仿真基于 RUBiS(Rice University Bidding System)基准测试环境[14]。RUBiS 遵从了 TPC-W[17]标准，它甚至根据统计规律定义了用户思考(发呆)时间和页面转换时间。

5.2　应用负载

根据文献[16]，在较短的一段时间内，到达的会话数的模型近似于均值为常量的泊松分布；长时间会话数的模型可以用时间频率函数的泊松分布 $Po(\lambda(t))$ 近似表示，其中 $\lambda(t)$ 表示时间的阶梯函数[10]。

本文据此实现了负载产生模块 genworkload，它负责按照泊松分布向多层应用发起会话请求，会话生命周期为 20 分钟，会话持续时间和会话数据访问请求频率基于 RUBiS 中的实现模型。实验模拟了 24 小时的负载产生，为了更加真实和细致地模拟负载变化，在总体遵循泊松分布的基础上，在一些时刻加入了突发会话。负载分布和变化如图 5 所示。

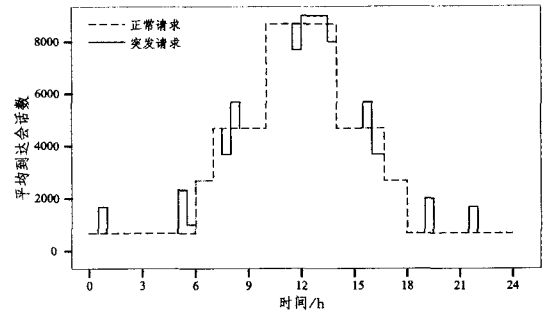

图 5　每分钟到达会话数的变化图

5.3　对比实验

为了使对比结果更加全面，分别从文献和工业实践中选取了两种方法与本文所提方法(OURS)进行对比实验：1)基于负载合并和虚拟机预取的资源管理方法(WCRP)[8]；2)基于商用云数据中心的资源按需伸缩方法(Baseline)。WCRP 中的虚拟机预取的个数为 1，即要保证至少有 1 个虚拟机是处于空闲备用状态，当少于 1 个时，启动增加操作；多于 1 个时，启动减少操作。在 Baseline 中，根据 AWS ELB[3]的设计方法，实现了简单 RR 算法，并且根据 AWS AutoScaling 方法实现了资源伸缩算法，即根据虚拟机实例组合的平均使用率判断增加或减少，增加和减少的阈值分别为 75%和 25%。冷却时间设置为 3min，即在 3min 内不允许出现连续两次资源伸缩，以稳定应用程序状态。虚拟机增加和缩小增量设置为 1 个。

5.4　结果对比

图 6 给出了应用程序正在服务的会话数。其中，如果发起认证请求，但建立连接失败，则标记该会话为拒绝；如果建立成功，并且在服务期间会话一直都能正常使用，则标记该会话为成功；如果建立成功，但在服务期间由于各种原因，会话无法进行数据请求，则标记为失败。从图 6 中可以看到，在会话认证请求数量突然上升的过程中，3 种方法都有不同程度的拒绝率，其中 Baseline 的拒绝率最大，WCRP 次之，OURS 最小。这主要是因为 Baseline 是在冷却时间的限制下按增量进行增加的，致使在新建会话请求突然增加时，VM 增加的速度无法满足服务需求，以至于出现大量的拒绝服务；但如果把增量调大、冷却时间调低，则会导致应用程序频繁开启/关闭虚拟机，将会大大增加租赁费用，同时造成系统服务性能不稳定。WCRP 虽然使用了资源预取机制，但是在分配负载时，负载均衡器优先选择高负载的服务器，当负载流量较大时，应用服务器将出现响应超时等错误。在到达会话请求频率下降的过程中，Baseline 方法出现了较高的访问失败率。一个最主要的原因是，云服务商提供的资源管理方法无法监控应用程序的内部状态，致使在释放虚拟机的时候，被关闭的应用服务器上仍有大量的活动会话，造成了较高的访问失败率。本文方法在虚拟机被释放之前，应用服务器都有一段时间的暂停状态用于处理剩余的会话，在释放时服务器上的活动会话

数已经很少，并且还会将它们迁移到其他活动服务器上，极少的服务失败主要是由于网络丢失等其他原因造成的。WCRP方法的失败率也很低，但访问拒绝率仍然相对较高。

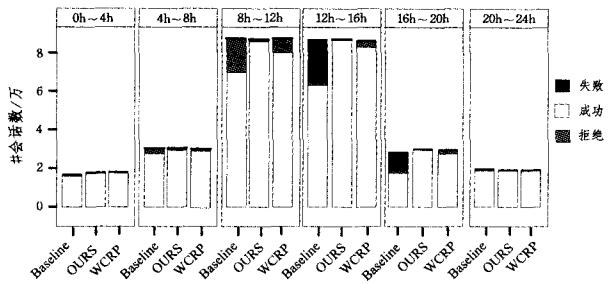

图6 会话数随时间变化图

图7给出了3种方法的成功会话的延迟分布图，其中菱形、箱型、竖线分别代表延迟数据在统计学上的均值、四分位差和中值。可以看出，WCRP方法的时延均值和中值明显比其他两种方法的高，并且分布也比较分散。OURS方法在均值和中值上略优于Baseline方法(分别小7%和5%)，虽然差别不大，但Baseline方法值的分布更加分散，而OURS值的范围更加集中。这说明，在Baseline方法下，一部分用户延迟较小，而另一部分用户延迟较大；而OURS方法的用户延迟则相对比较平均，绝大多数用户都无明显感受。同时也表明，当新增加服务器时，简单RR算法对各个服务器上负载的主动平衡能力比本文负载均衡算法要差。从总体上看，OURS方法的平均性能虽然只是略优于Baseline方法，但有较多会话的延迟要低于Baseline方法，总体服务质量和用户体验高于Baseline方法。OURS和Baseline方法的延迟都要远好于WCRP方法，平均延迟分别减少了22%和20%。这主要基于以下几个原因：1)WCRP方法由于负载均衡分配算法的原因，应用服务器始终维持较高负载，致使会话平均响应时间较长；2)Baseline方法成功会话的数目相对其他两种方法比较少，因此参与统计的元素较少。

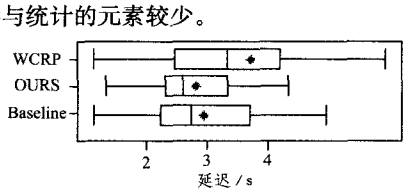

图7 延迟分布图

最后，对3种算法的租赁费用进行比较。每种算法可能使用多个虚拟机，一个虚拟机也可能在运行过程中甚至一个购买时段内多次购买和释放，虚拟机按小时计费(不足1小时按1小时计算)，且租赁单价相同，因此可以用虚拟机使用总时数与单价的积定义租赁费用。虚拟机总时数和租赁费用的公式化描述分别如式(2)和式(3)所示。在式(2)中，如果对一个虚拟机一次性连续购买多个时段，则购买次数按照多次计算。

$$\text{虚拟机总时数}=\sum_{0}^{23}\text{小时内购买次数} \tag{2}$$

$$\text{租赁费用}=\text{虚拟机总时数}\times\text{单价} \tag{3}$$

因此只需要比较3种方法在一天时间内所使用的虚拟机总时数即可，如图8所示。可以看出，所提方法的租赁费用要低于基准方法和WCRP方法，在本文的负载模型下，其花费分别是Baseline方法所用费用的78%和WCRP方法所用费用的60%。原因可能包括：1)在遇到突发流量请求时，Baseline方法出现了多次重复申请虚拟机的情况，而WCRP方法总是需要额外保持一定数量的预取虚拟机应对突发流量，它们都额外增加了租赁费用；2)在决定释放虚拟机时，Baseline方法没有考虑已购买时间，致使已付费虚拟机未得到充分利用。另外，Baseline方法的费用低于WCRP的主要原因是前者的服务质量较差，服务成功的会话数比后者少，需要的应用服务器也少于后者。

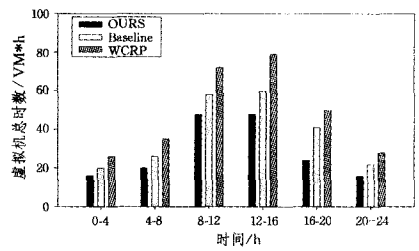

图8 费用对比图

结束语 本文从云计算中心使用者的角度出发，提出了一种面向云多层应用的费用高效的资源管理方法，并根据真实的云数据中心的设置、虚拟机类型和价格，以及接近真实的负载模型，使用多个工业级的仿真软件对该方法进行了仿真验证。对比结果表明，本文方法的平均性能略优于基准方法，远好于WCRP方法；服务质量远优于基准方法，与WCRP相当；租赁费用分别降低了22%和40%。

参 考 文 献

[1] AMAZON. Amazon elastic compute cloud (Amazon EC2) [EB/OL]. [2015-3-1]. http://aws.amazon.com/cn/ec2/purchasing-options/.

[2] AMAZON. Amazon CloudWatch [EB/OL]. [2015-3-1]. http://aws.amazon.com/cn/cloudwatch/.

[3] AMAZON. Elastic Load Balancing[EB/OL]. [2015-3-1]. http://aws.amazon.com/cn/elasticloadbalancing/.

[4] AMAZON. Amazon Auto Scaling [EB/OL]. [2015-3-1]. http://aws.amazon.com/cn/autoscaling/.

[5] MASTELIC T, OLEKSIAK A, CLAUSSEN H, et al. Cloud Computing: Survey on Energy Efficiency [J]. ACM Comput. Surv., 2014, 47(2): 1-36.

[6] DING Y W, QIN X L, LIU L, 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.

[7] KONG F, LIU X. A Survey on Green-Energy-Aware Power Management for Datacenters [J]. ACM Comput. Surv., 2014, 47(2): 1-38.

[8] GROZEV N, BUYYA R. Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications [J]. ACM Trans. Auton. Adapt. Syst., 2014, 9(3): 1-21.

[9] FOWLER M. Patterns of enterprise application architecture [M]. Addison-Wesley Longman Publishing Co., Inc., 2002.

(下转第117页)

本相同，而自相关法的正确识别率则有所下降。从图 5 中可以看出，循环前缀长度比例为 1/16，1/32 时，本文算法的正确率识别率受循环前缀长度比例影响不大，符号数大于 15 时的正确识别率均能达到 90%以上。相比之下，自相关法的正确识别率则受循环前缀长度比例影响较大，循环前缀长度比例为 1/16 时，自相关法的正确识别率最高还不到 50%，无法有效识别 OFDM 信号；而当循环前缀长度比例为 1/32 时，自相关法的正确识别率已经很低，不再适用于此条件下的 OFDM 信号识别。由此可见，相比自相关法识别 OFDM 信号，本文算法的识别性能对符号数变化不敏感，受循环前缀长度比例影响不大，估计性能更优，适用范围更广。

结束语　针对多径信道下传统的 OFDM 信号识别方法存在循环前缀较短时估计性能不高、所需 OFDM 符号数过多等问题，提出一种基于 MUSIC 算法的 OFDM 信号识别方法。该算法利用噪声子空间和信号子空间之间的正交性实现 OFDM 信号的识别，并从信噪比和 OFDM 符号数两个角度考察对实验结果的影响。仿真实验表明，该方法仅需要较少的 OFDM 符号就可以实现短循环前缀 OFDM 信号的识别，且识别性能优于传统方法。

参 考 文 献

[1] HAN N, ZHENG G, SOHN S H, et al. Cyclic autocorrelation based blind OFDM detection and identification for cognitive radio[C]// IEEE 4th International Conference on Wireless Communications Networking and Mobile Computing. 2008: 1-5.

[2] 张晶晶. 基于小波变换的 OFDM 信号识别技术研究[D]. 西安：西安电子科技大学，2009.

[3] QI T, YANG H, ZHANG H M. Application Research of Military Wireless Multimedia Communication Baesd on OFDM Technology[J]. Computer Knowledge and Thchnology, 2010 (4): 951-952. (in Chinese)
齐涛，杨浩，张宏珉. 基于 OFDM 的军事无线多媒体通信应用研究[J]. 电脑知识与技术，2010(4)：951-952.

[4] REDDY S B, YÜCEK T, ARSLAN H. An efficient blind modulation detection for adaptive OFDM systems[C]// Vehicular Technology Conference, IEEE 58th. 2003: 1895-1899.

[5] CHATTERJEE S, FERNADNO W A C. Blind estimation of channel and modulation scheme in adaptive modulation schemes for OFDM-CDMA based 4G systems[J]. IEEE Transactions on Consumer Electronics, 2004, 50(4): 1065-1075.

[6] HIJAZI H, ROS L. Polynomial estimation of time-varying multipath gains with intercarrier interference mitigation in OFDM systems[J]. IEEE Transaction on Vehicular Technology, 2009, 58(1): 140-151.

[7] PANAYIRCI E, ŞENOL H, POOR H V. Joint channel estimation, equalization, and data detection for OFDM systems in the presence of very high mobility[J]. IEEE Transactions on Signal Processing, 2010, 58(8): 4225-4238.

[8] ABOUTORAB N, HARDJAWANA W, VUCETIC B. A new iterative Doppler-assisted channel estimation joint with parallel ICI cancellation for high-mobility MIMO-OFDM systems[J]. IEEE Transactions on Vehicular Technology, 2012, 61(4): 1577-1589.

[9] LI G H, WANG K R, JIN H. Robust method of detecting OFDM signals based on cyclic autocorrelation[J]. Application Research of Computers, 2012, 29(2): 714-716. (in Chinese)
李国汉，王可人，金虎. 一种基于循环相关的 OFDM 信号稳健检测法[J]. 计算机应用研究，2012，29(2)：714-716.

[10] HAN G, LI J D, LI C L. Study of blind detection techniques in adaptive OFDM[J]. Journal of Xidian University, 2006, 33(4): 602-606. (in Chinese)
韩钢，李建东，李长乐. 自适应 OFDM 中信号盲检测技术[J]. 西安电子科技大学学报(自然科学版)，2006，33(4)：602-606.

[11] EDFORS O, SANDELL M, JAN-JAAP V D B, et al. OFDM channel estimation by singular value decomposition[J]. IEEE Transactions on Communications, 1998, 46(7): 931-939.

[12] 刘汝哲. 基于循环平稳的非合作 OFDM 信号检测与识别[D]. 北京：北京邮电大学，2013.

[13] YANG W, HUANG D S, MA Z H. Root-MUSIC with real-valued eigen-decomposition for coherent signals based on cross-correlation vector reconstruction[J]. Application Research of Computers, 2010(11): 4254-4256. (in Chinese)
杨武，黄登山，马振华. 基于互相关矢量重构的解相干实值 Root-MUSIC 算法[J]. 计算机应用研究，2010(11)：4254-4256.

(上接第 78 页)

[10] GROZEV N, BUYYA R. Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments [J]. The Computer Journal, 2015, 58(1): 1-22.

[11] PULLARA S, HALPERN E, PEDDADA P, et al. Method and apparatus for session replication and failover: CN1549978 A [P]. 2003.

[12] REVANURU N, TORSTENSSON P, AGARWAL P, et al. System and method for supporting one-way remote method invocation for session replication in a server cluster: US8856352[P]. 2014.

[13] 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.

[14] RUBiS. RUBiS: Rice University Bidding System[EB/OL]. [2015-3-1]. http://rubis.ow2.org.

[15] MAO M, HUMPHREY M. A performance study on the vm startup time in the cloud [C]// 2012 IEEE 5th International Conference on Cloud Computing (CLOUD). 2012: 423-430.

[16] CAO J, Andersson M, Nyberg C, et al. Web server performance modeling using an M/G/1/K * PS queue[C]// 10th International Conference on Telecommunications, 2003 (ICT 2003). 2003: 1501-1506.

[17] MENASCE, DANIEL. TPC-W: A benchmark for e-commerce [J]. Internet Computing, IEEE, 2002, 6(3): 83-87.