软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(2):358−377 [doi: 10.3724/SP.J.1001.2013.04216]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向应用服务级目标的虚拟化资源管理*

文 雨[1,2], 孟 丹[1], 詹剑锋[1]

[1](中国科学院 计算技术研究所 高性能计算机研究中心,北京 100190)

[2](中国科学院 研究生院,北京 100049)

通讯作者: 文雨, E-mail: rainix.oo@gmail.com

摘 要: 在虚拟环境中实现应用服务级目标,是当前数据中心系统管理的关键问题之一.解决该问题有两个方面的要求:一方面,在虚拟化层次和范围内,能够动态和分布式地按需调整虚拟机资源分配;另一方面,在虚拟化范围之外,能够控制由于虚拟机对非虚拟化资源的竞争所导致的性能干扰,实现虚拟机性能隔离.然而,已有工作不适用于虚拟化数据中心场景.提出一种面向应用服务级目标的虚拟化资源管理方法.首先,该方法基于反馈控制理论,通过动态调整虚拟机资源分配来实现每个应用的服务器目标;同时,还设计了一个两层结构的自适应机制,使得应用模型能够动态地捕捉虚拟机资源分配与应用性能的时变非线性关系;最后,该方法通过仲裁不同应用的资源分配请求来控制虚拟机在非虚拟化资源上的竞争干扰.实验在基于Xen的机群环境中检验了该方法在RUBiS系统和TPC-W基准上的效果.实验结果显示,该方法的应用服务级目标实现率比两种对比方法平均高 29.2%,而应用服务级目标平均偏离率比它们平均低 50.1%.另一方面,当 RUBiS 系统和 TPC-W 基准竞争非虚拟化的磁盘 I/O 资源时,该方法通过抑制 TPC-W 基准 28.7%的处理器资源需求来优先满足 RUBiS 系统的磁盘 I/O 需求.

关键词: 虚拟化;数据中心;多层应用;资源管理;服务级目标;控制论;线性二次方法;聚类算法

中图法分类号: TP316 **文献标识码**: A

中文引用格式: 文雨,孟丹,詹剑锋.面向应用服务级目标的虚拟化资源管理.软件学报,2013,24(2):358−377. http://www.jos.org.cn/1000-9825/4216.htm

英文引用格式: Wen Y, Meng D, Zhan JF. Adaptive virtualized resource management for application's SLO guarantees. Ruanjian Xuebao/Journal of Software, 2013,24(2):358−377 (in Chinese). http://www.jos.org.cn/1000-9825/4216.htm

Adaptive Virtualized Resource Management for Application's SLO Guarantees

WEN Yu[1,2], MENG Dan[1], ZHAN Jiang-Feng[1]

[1](High Performance Computer Research Center, Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

Corresponding author: WEN Yu, E-mail: rainix.oo@gmail.com

Abstract: Virtualized resources management for service level objectives (SLOs) of applications has been one of the key problems of system management in current data centers. To solve this problem one needs to: 1) dynamically and distributed allocating resources to virtual machines (VMs) of applications on demand; 2) efficiently control interference among VMs consolidated on a single physical server, due to their contention on non-virtualized resources. Many existing methods, however, are not suitable for this virtualized data center scenario. This paper presents a method for the virtualized resource management for SLOs of applications. First, based on the feedback control theory, this method can achieve SLOs of applications through dynamically resourced allocation. Second, a two-layer self-adaptive mechanism is devised and used to dynamically capture the non-linear relationship between the performance of applications and the resources allocation. Third, this method can control the performance interference among VMs on non-virtualized resources through

* 基金项目: 国家自然科学基金(6093303)

收稿时间: 2011-09-21; 修改时间: 2012-02-25; 定稿时间: 2012-03-15

virtualized resources allocation. The study has evaluated this method on the RUBiS system and TPC-W benchmark in a Xen-based virtualized cluster. The experimental results show that the average rate of SLOs achieved by this method is 29.2% higher than ones by two existing methods. Along with the average deviation from SLOs, this method is 50.1% lower than ones of the existing methods. Furthermore, when resource contention occurs on non-virtualized disk I/O between RUBiS and TPC-W, this method can almost entirely satisfy the disk I/O requirement of RUBiS of high priority through restraining TPC-W requests, e.g. 28.7%, on virtualized CPU.

Key words: virtualization; data center; multi-tier application; resource management; service level object; control theory; linear quadratic method; clustering algorithm

资源管理一直是数据中心系统管理的重要问题之一.数据中心是一个分布式和多层次的资源共享环境.应用被部署在多台服务器上,同时与其他应用不仅共享数据中心,甚至基于服务聚合方式分享服务器物理资源,如处理器、磁盘 I/O、网络带宽等.因此,数据中心系统管理需要高效的资源分配机制和策略.而近年来出现的虚拟化技术又促进了数据中心资源共享模式的发展.基于虚拟机,系统资源可被任意地划分和分配,这进一步提高了系统资源利用率,降低了系统投入成本.另一方面,随着云计算的出现,越来越多的应用和服务被开发和部署在数据中心里.由于这些应用不断增加的社会影响力和商业价值,它们在公众服务部门和企业中占据了极其重要的地位.因此,保障这些应用的服务级目标(service-level object)同样是系统管理的重要任务.应用的服务级目标通常使用应用端到端行为指标来描述,如请求平均响应时间不大于 1s,或吞吐率不小于 500 请求/s 等.这些目标在应用服务供应商与平台服务供应商的合同中被明确制定,如果违反,则意味着严重的经济后果.

本文的研究关注面向数据中心应用服务级目标的虚拟化资源管理问题.图 1 描述了一个部署了多层应用的虚拟化环境.其中,多层应用的各层模块通常分布在不同虚拟机中,如应用 A 的 Web 层、App 层和 DB 层分别在虚拟机 VM-1、虚拟机 VM-2 和虚拟机 VM-3.同时,多个虚拟机以服务聚合方式共享同一个物理服务器,如虚拟机 VM-2 和虚拟机 VM-5 同位于服务器 Node-2.在这种虚拟共享环境中确保应用服务级目标的实现,面临的挑战有:

1) 虚拟机行为具有时变性.应用负载通常表现为显著的波动性[1,2],这导致虚拟机资源需求也随之动态变化.资源管理系统需要为虚拟机动态地按需分配系统资源,在满足应用需求的同时保持较高的资源利用率.
2) 虚拟机行为具有相关性.多层应用的一次请求处理会涉及到多个模块,因此,应用负载变化会同时引起这些模块的资源需求的变化.资源管理系统需要同时调整多个虚拟机的资源分配.
3) 虚拟机之间仍然存在相互干扰.这些干扰由虚拟机的资源竞争行为导致,主要包括虚拟化范围之外的资源,即非虚拟化资源.虚拟技术完全实现对各类资源的任意分割,需要从硬件到软件的全面支持.而目前的许多硬件系统缺乏对虚拟化的支持,如磁盘控制器、网络接口、memory/cache 系统等.管理系统需要考虑虚拟机的这种相互影响行为.

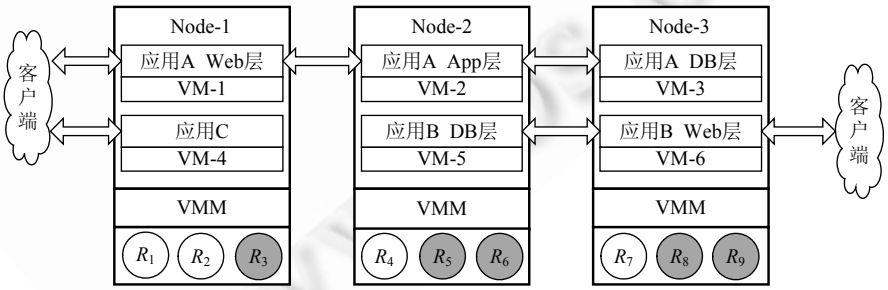


Fig.1 Data center application in virtualized environment

图 1 虚拟环境中的数据中心应用

共享环境中资源管理的代表工作有文献[1,3−5]等.以文献[1,3]为代表的工作主要为多实例的单层应用整

体的分配系统资源,没有考虑应用内部模块的资源需求变化和联系.此外,这类工作通常不是直接以实现应用服务级目标为目的.而在面向虚拟环境的资源管理工作中,文献[4]使用虚拟机迁移机制解决服务器过载问题,文献[5]通过动态的虚拟机部署来按需构建虚拟运行环境.这种仅依靠虚拟机部署调整的资源分配方法通常都会带来很高的系统开销,不适合单独作为虚拟化资源的管理手段.同时,这些方法都没有考虑虚拟机的性能隔离问题.

本文提出一种适用于虚拟环境的资源管理方法.我们为系统中每个多层应用配置一个应用资源控制器,并在每台服务器节点上部署一个服务器资源分配器.应用资源控制器周期性地计算应用虚拟机的资源需求,而服务器资源分配器决定实际的资源分配.本文的贡献主要有:

1) 通过动态的虚拟机资源分配实现所有应用的服务级目标.基于控制论原理,我们将应用资源控制器设计成一个直接面向应用服务级目标的资源反馈控制系统.将应用看作黑盒系统,我们使用一个时序模型将虚拟机资源分配与应用端到端行为直接关联起来.在这个模型的驱动下,应用资源控制器能够根据应用当前行为与应用服务级目标的差异,计算出应用虚拟机当前的资源需求.
2) 提出一个两层结构的管理系统自适应机制来帮助应用资源控制器动态地捕捉虚拟机资源分配与应用端到端行为的时变的非线性关系.其中,反应式的模型调节方式通过定期的在线系统辨识过程来调整应用资源控制器.而前摄式的模型调度方式识别和记录辨识过程的有效结果,然后利用这些结果直接配置驱动应用资源控制器的应用模型.
3) 通过虚拟化资源分配实现虚拟机在非虚拟化资源上的性能隔离.服务器资源分配器通过仲裁应用的虚拟化资源分配要求来控制虚拟机在非虚拟化资源上的相互干扰,实现虚拟机性能隔离.

实验中,我们在一个基于 Xen 的虚拟化机群系统中,使用 RUBiS 系统[6]和 TPC-W[7]基准作为测试应用,检验了本文方法的效果,并与两种方法做了对比.其中:一种方法同样基于反馈控制原理,但通过离线的系统辨识过程建立驱动反馈控制机制的应用模型.这种离线的系统辨识过程是目前基于控制论原理的系统管理方法中最常用的方法,因此,我们称其为 Baseline 方法;另一种方法是 Xen 的默认资源分配方法,它允许虚拟机任意使用系统资源.

实验结果显示:

1) 基于模型调节方式,本文方法的应用服务级目标实现率比 Baseline 方法平均高 21.6%,比 Xen 默认方式平均高 33.3%.这说明,本文方法能够通过动态的虚拟机资源分配来适应应用变化,从而更有效地实现应用服务级目标.
2) 模型调度结合模型调节的自适应方式,在应用服务级目标实现率指标上,比模型调节方式平均高 18.4%,比模型调度方式平均高 43.5%.这说明,混合方式综合了两个自适应方法的优点.
3) 当应用竞争非虚拟化的磁盘 I/O 资源时,优先级高的 RUBiS 系统的处理器需求被满足了 95.4%,而 TPC-W 基准的需求只被满足了 71.3%.同时,RUBiS 磁盘 I/O 使用率只比其磁盘 I/O 需求低了 4.7%,而 TPC-W 基准则低了 37.1%.这说明,我们的方法能够通过分配虚拟化的处理器资源,控制虚拟机对非虚拟化的磁盘 I/O 资源的竞争.

本文第 1 节描述本文工作面向的场景、问题以及目标.第 2 节介绍本文虚拟化资源管理系统中关键部分的设计思路,包括基本的反馈控制器、资源分配器、模型调节器和模型调度器.第 3 节是本文的实验评价部分.第 4 节介绍典型的相关研究工作,包括共享环境资源管理和控制论在计算机系统管理方面的应用.最后一节是本文的总结部分.

1 问题描述

引言中已经提到,在虚拟环境中,多层应用以分布式的聚合方式共享数据中心资源.此外,如图 1 所示,每台服务器包含有多项虚拟化资源(如 R_1,R_4 等)和多项非虚拟化资源(如 R_3,R_5 等).在本文中,我们考虑的场景主要包括虚拟化的处理器资源和非虚拟化的磁盘 I/O 资源.

本文的研究目的是,通过虚拟化资源分配实现数据中心应用服务级目标.我们使用应用级行为指标来描述应用的服务级目标,包括请求平均响应时间和应用吞吐率.应用服务级目标可表述为:平均响应时间不大于 1s,或吞吐率不低于 500 请求/s.

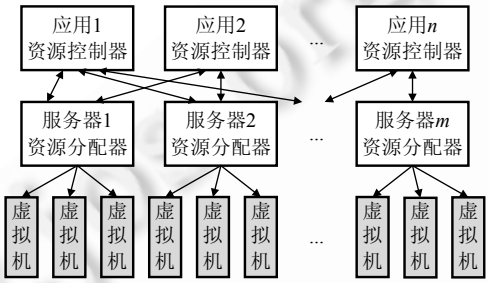
使用我们的虚拟化资源管理方法,能够帮助系统管理员完成以下几个方面的工作:

1) 动态的资源管理:应对负载波动,为虚拟机按需分配资源.
2) 分布式的资源分配:同时调整多个相关虚拟机的资源分配.
3) 虚拟机性能隔离:控制由虚拟机的非虚拟化资源竞争带来的性能干扰.
4) 基于情形 1)~情形 3),实现应用的服务级目标.

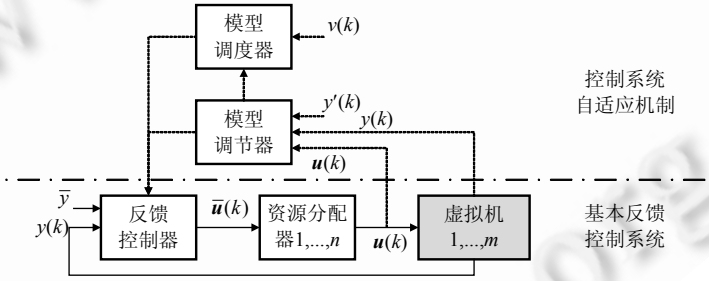
2 虚拟化资源管理方法

2.1 总体结构

图 2 是我们的虚拟化资源管理系统的逻辑结构.



(a) 虚拟资源管理系统总体结构



(b) 应用资源控制器结构

Fig.2 Logical structure of our method for virtualized resource management

图 2 本文虚拟化资源管理方法的逻辑结构

如图 2(a)所示,我们为每个应用配置一个资源控制器.应用资源控制器,周期性地根据应用当前实测性能与其目标的差异,动态地为应用计算出资源需求.实验中,以每 9s 为一个控制周期.同时,我们的方法在每个物理服务器上部署一个资源分配器.在每个控制周期,应用资源控制器将资源分配请求发送给它的虚拟机所在服务器上的资源分配器.然后,服务器资源分配器通过仲裁这些资源分配请求决定实际的资源分配来实现应用隔离.

图 2(b)是应用资源控制器的内部结构.它主要包括 3 个模块:反馈控制器、模型调节器和模型调度器.在每个控制周期,反馈控制器根据应用服务级目标 $\bar{y}$ 与应用当前实测性能 $y(k)$的差异,计算新的应用资源分配 $\bar{\boldsymbol{u}}(k)$,并发送给所在服务器的所有资源分配器.资源分配器执行实际的资源分配 $\boldsymbol{u}(k)$.在反馈控制器中,一个时序模型被用于确定性地描述虚拟机资源分配与应用性能的局部线性关系.为了实现反馈控制器在全局范围内的非线

性控制功能,我们使用模型调节器和模型调度器构成的自适应机制,动态地捕捉虚拟机资源分配与应用性能的时变非线性关系.在每个控制周期,模型调节器根据应用预计性能 $y'(k)$与实际性能 $y(k)$的差异以及资源实际分配 $\boldsymbol{u}(k)$,在线调整控制器模型.模型调度器基于模型调节器,构建和维护一个应用的有效模型集.然后,在每个控制周期,自适应机制首先基于对某个系统变量的观察 $v(k)$,通过直接配置应用模型来调整反馈控制器.如果没有合适的有效模型配置反馈控制器,则继续通过模型调节器调整反馈控制器.

下面,我们将分别介绍反馈控制器、资源分配器、模型调节器和模型调度器的设计思路.

2.2 反馈控制器

在本文中,我们采用被广泛应用于工程控制系统的自回归平均移动模型(简称 ARMA 模型)作为驱动反馈控制器的应用模型,它确定性地描述了应用性能与资源分配的局部线性关系.具体来说,我们使用ARMA模型将应用历史性能、资源历史分配情况和当前分配情况与应用未来性能联系起来.实验中我们发现,二阶的 ARMA 模型即可满足本文对控制器精度的要求.具体模型形式如下:

$$y_a(k)=a_1(k)y_a(k-1)+\boldsymbol{b}_0^{\tau}(k)\boldsymbol{u}_a(k)+\boldsymbol{b}_1^{\tau}(k)\boldsymbol{u}_a(k-1) \tag{1}$$

这里的 $y_a(k)$表示应用 A 在第 k 控制周期的预计性能,$y_a(k-1)$是前次控制周期里应用的实测性能,它们通过模型参数 a_1 被关联起来.$\boldsymbol{u}_a(k)$是应用 A 在控制周期 k 所在服务器的所有虚拟机资源分配,相应地,$\boldsymbol{u}_a(k-1)$是前次控制周期的资源分配,模型参数 $\boldsymbol{b}_0$ 和 $\boldsymbol{b}_1$ 分别将资源分配与应用预计性能关联起来.因为需要关联多项资源,$\boldsymbol{u}_a(k)$、$\boldsymbol{u}_a(k-1)$、参数 $\boldsymbol{b}_0$ 和参数 $\boldsymbol{b}_1$ 都是列向量.由于不同应用的虚拟机存在对非虚拟化资源的竞争带来的相互干扰,并且不同应用的虚拟化资源分配也决定了这种干扰程度,因此,为了在公式(1)中刻画这种干扰对应用 A 的影响,我们扩展了普通 ARMA 模型.扩展的 ARMA 模型中,$\boldsymbol{u}_a(k)$和 $\boldsymbol{u}_a(k-1)$不仅包括应用 A 的虚拟机资源分配,而且也包括应用 A 所在服务器的其他应用虚拟机的资源分配.这两种资源分配在模型中的意义不同,前者是应用 A 达到性能目标所依赖的资源分配,后者是从应用 A 受其他应用干扰的角度描述其他应用的资源分配.需要注意的是,模型参数 $\boldsymbol{b}_0$ 和 $\boldsymbol{b}_1$ 同时也是以控制周期 k 为自变量的函数.在下文中,我们将介绍如何通过动态调整模型参数实现反馈控制器的非线性控制功能.

下面,我们基于公式(1)的模型计算应用的资源需求.实际上,求解满足应用服务级目标的资源分配问题是一个优化控制问题.设应用 A 在控制周期 k 的资源需求表示为 $\bar{\boldsymbol{u}}_a(k)$,则优化目标可以如下描述:

(i) minimize $|y_a(k)-\bar{y}_a|$;

(ii) minimize $\|\bar{\boldsymbol{u}}_a(k)-\boldsymbol{u}_a(k-1)\|$.

这里,$\bar{y}_a$ 是应用 A 的服务级目标,如请求平均响应时间.优化目标的含义是,控制系统通过最小程度地调整资源分配来实现应用服务级目标.尽可能小地调整资源分配能够避免系统震荡.

大多数优化控制问题可以通过线性二次(linear quadratic,简称LQ)方法求解[8].接下来,优化目标可写成如下的代价函数:

$$J_a=W_y|y_a(k)-\bar{y}_a|^2+W_{\boldsymbol{u}}\|\bar{\boldsymbol{u}}_a(k)-\boldsymbol{u}_a(k-1)\|^2 \tag{2}$$

其中,J_a 表示应用 A 的代价函数;W_y 和 $W_{\boldsymbol{u}}$ 分别是优化目标(i)和目标(ii)的控制权重,二者是控制系统实现应用目标与调整资源分配之间的权衡.实验中,我们设置 W_y 为 $\frac{1}{\bar{y}_a}$,设置 $W_{\boldsymbol{u}}$ 为 $\frac{1}{\|\boldsymbol{u}_{\max}\|}$,其中,$\boldsymbol{u}_{\max}$ 代表最大资源分配值.

当 $\bar{\boldsymbol{u}}_a(k)$ 的取值使得 J_a 函数在 $\bar{\boldsymbol{u}}_a(k)$ 上的偏导数为 0 时,函数 J_a 有最小值.

为了便于描述,设 $\boldsymbol{x}(k)=[\boldsymbol{b}_1(k)a_1(k)]$,$\boldsymbol{\varphi}(k-1)=[\boldsymbol{u}_a^{\tau}(k-1)y_a(k-1)]^{\tau}$. 接着,将公式(1)代入公式(2)展开,有

$$\begin{aligned}J_a=&W_y((\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1)-\bar{y}_a(k))^2+(\boldsymbol{b}_0(k)\bar{\boldsymbol{u}}_a(k))^2+2\bar{\boldsymbol{u}}_a^{\tau}(k)\boldsymbol{b}_0^{\tau}(k)(\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1)-\bar{y}_a(k)))+\\&W_{\boldsymbol{u}}(\|\bar{\boldsymbol{u}}_a(k)\|^2+\|\boldsymbol{u}_a(k-1)\|^2-2\boldsymbol{u}_a^{\tau}(k-1)\bar{\boldsymbol{u}}_a(k)).\end{aligned}$$

然后,求函数 J_a 在 $\bar{\boldsymbol{u}}_a(k)$ 上的偏导函数,有

$$\frac{\partial J_a}{\partial\bar{\boldsymbol{u}}_a(k)}=2W_y\boldsymbol{b}_0^{\tau}(k)\boldsymbol{b}_0(k)\bar{\boldsymbol{u}}_a(k)+2W_y\boldsymbol{b}_0^{\tau}(k)(\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1)-\bar{y}_a(k))+2W_{\boldsymbol{u}}\bar{\boldsymbol{u}}_a(k)-2W_{\boldsymbol{u}}\boldsymbol{u}_a^{\tau}(k-1).$$

最后,当 $\frac{\partial J_a}{\partial \bar{\boldsymbol{u}}_a(k)}=0$ 时,可推导出

$$\bar{\boldsymbol{u}}_a^*(k)=(W_y\boldsymbol{b}_0^\tau(k)\boldsymbol{b}_0(k)+2W_{\boldsymbol{u}}I)^{-1}(W_y\boldsymbol{b}_0^\tau(k)(\bar{y}_a(k)-\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1))+W_{\boldsymbol{u}}\boldsymbol{u}_a^\tau(k-1)).$$

这里,I 是单位矩阵.

2.3 资源分配器

当反馈控制器计算出应用当前的资源分配需求后,向应用所在的服务器上的所有资源分配器发送资源分配请求.每个服务器资源分配器仲裁接收到的所有资源分配请求,决定实际的资源分配.

这里以图 1 中应用 A 和应用 B 请求服务器 Node-3 分配资源 R_7 为例来说明如何分配资源.设 $\bar{u}_{a,3,7}$ 和 $\bar{u}_{a,6,7}$ 分别表示应用 A 在某个控制周期里请求 Node-3 分配给虚拟机 VM-3 和虚拟机 VM-6 的资源 R_7 份额.同时,$\bar{u}_{a,3,7}$ 和 $\bar{u}_{a,6,7}$ 也是应用 A 的资源需求向量 $\bar{\boldsymbol{u}}_a$ 里的元素.相应地,应用 B 的资源分配请求是 $\bar{u}_{b,3,7}$ 和 $\bar{u}_{b,6,7}$.

当 Node-3 的资源满足所有应用需求时,服务器资源分配器按照每个应用的资源请求来分配资源.具体来说,此时应用的资源请求应该符合以下 3 个约束:

(i) $\bar{u}_{a,3,7}+\bar{u}_{b,6,7}\leqslant 1$;

(ii) $\bar{u}_{a,3,7}\leqslant \bar{u}_{b,3,7}$;

(iii) $\bar{u}_{b,6,7}\leqslant \bar{u}_{a,6,7}$.

约束(i)表示资源 R_7 能够同时满足应用 A 和应用 B 的当前需求,而约束(ii)和约束(iii)表示应用 A 和应用 B 对资源 R_7 的分配要求没有冲突.

当服务器资源缺乏时,可能出现两种资源竞争情况:一种发生在虚拟化资源,如 R_7;另一种发生在非虚拟化资源,如 R_8 和 R_9.首先,如果约束(i)没有被满足,则说明资源 R_7 过载,此时需要通过过载控制或者虚拟机迁移等方法解决.已有一些针对这个问题的方法被提出来,它们可与本文方法形成互补,因此这里不作考虑.另外,如果应用的资源分配请求满足约束(i)但是不满足约束(ii)或者约束(iii),则说明应用在非虚拟化资源 R_8 或 R_9 上存在竞争带来的相互干扰.

现在以应用资源请求违反约束(ii)为例,说明如何通过虚拟化资源 R_7 的分配来控制应用在非虚拟化资源上的相互影响.设 $u_{3,7}$ 表示实际分配给虚拟机 VM-3 的资源 R_7,并设 $\Delta u_{a,3,7}=|u_{3,7}-\bar{u}_{a,3,7}|$,$\Delta u_{b,3,7}=|u_{3,7}-\bar{u}_{b,3,7}|$,它们表示虚拟机 VM-3 资源 R_7 实际分配分别与应用 A 和应用 B 期望的该虚拟机资源分配的差值.根据公式(1),可以得出,当资源分配请求没有被完全满足时,应用 A 和应用 B 的性能损失分别是 $\frac{\partial y_a}{\partial u_{a,3,7}}\Delta u_{a,3,7}$ 和 $\frac{\partial y_b}{\partial u_{b,3,7}}\Delta u_{b,3,7}$.这里,应用 A 的性能损失是由于虚拟机 VM-3 没有得到足够的资源 R_7,而应用 B 的性能损失则是由于虚拟机 VM-3 继续获得资源 R_7 所带来的性能影响.基于这两个性能损失可以得到一个总体的代价函数:

$$J_{3,7}=W_a\left(\frac{\partial y_a}{\partial u_{a,3,7}}\Delta u_{a,3,7}\right)^2+W_b\left(\frac{\partial y_b}{\partial u_{b,3,7}}\Delta u_{b,3,7}\right)^2.$$

这里,W_a 和 W_b 分别表示应用资源分配的优先权,$W_a>W_b$ 表示优先满足应用 A 的资源分配要求.代价函数的意义在于,找到使得总代价最小的资源 R_7 分配方案.

与上一节类似,借助线性二次求解工具可以计算出具体的资源分配.

2.4 模型调节器

第 2.2 节中提到,在反馈控制器中,我们使用公式(1)时序模型刻画应用性能与资源分配的局部线性关系.为了适应资源分配与应用性能的非线性关系,需要动态的系统辨识过程来调整应用模型.

在经典的控制系统设计中,系统辨识通常采用最小二乘法来估计模型参数.这里,我们同样采用最小二乘估计,但不同的是,为了满足动态的估计模型的要求,我们使用了递归最小二乘算法[9].递归最小二乘法是普通最小二乘回归过程的在线方法.该方法不仅能够产生与普通最小二乘法同样的估计效果,而且由于递归的特点,它与

普通最小二乘法相比还具有低计算复杂度的优点.

为了便于描述递归最小二乘过程,我们将公式(1)改写成 $y(k)=x(k)\varphi(k)$,其中,

$$\boldsymbol{x}(k)=[\boldsymbol{b}_0(k)\boldsymbol{b}_1(k)a_1(k)],\varphi(k)=[\boldsymbol{u}_a^\tau(k)\boldsymbol{u}_a^\tau(k-1)y_a(k-1)]^\tau.$$

递归最小二乘参数估计描述如下:

$$\varepsilon(k)=y_a(k)-\hat{\boldsymbol{x}}(k-1)\varphi(k) \tag{3}$$

$$\hat{\boldsymbol{x}}(k)=\hat{\boldsymbol{x}}(k-1)+\frac{\varepsilon(k)\varphi^\tau(k)A(k-2)}{1+\varphi^\tau(k)A(k-2)\varphi^\tau(k)} \tag{4}$$

$$A(k-1)=A(k-2)-\frac{A(k-2)\varphi(k)\varphi^\tau(k)A(k-2)}{1+\varphi^\tau(k)A(k-2)\varphi(k)} \tag{5}$$

这里,$y_a(k)$是应用的实际测量性能, $\hat{\boldsymbol{x}}(k)$ 是 $\boldsymbol{x}(k)$的估计值,则 $\hat{\boldsymbol{x}}(k-1)\varphi(k)$ 是应用的预计性能,而$\varepsilon(k)$就是模型精度误差,$A(k)$是协方差矩阵.

在每个控制周期,如果公式(3)的误差为 0 或者可以忽略不计,则说明当前模型能够准确地描述应用性能与资源分配的关系,因此无须调整;否则,按照公式(4)和公式(5)重新计算模型参数的估计值 $\hat{\boldsymbol{x}}(k)$.

2.5 模型调度器

上一节介绍了模型调节器通过动态估计模型参数实现控制系统的非线性控制机制.但是它有两个缺点:一是在模型调整过程中通常需要经过模型收敛的过程,尤其是应用负载波动较显著的情况;另一个是在应用重复变化过程中不能利用已得到的有效调整结果.本节我们将介绍另一种控制系统自适应机制——模型调度器.与模型调节器不同的是,模型调度器记录已知的有效模型并建立一个模型集,然后基于模型集直接配置应用模型来调整控制系统.这样,可以避免模型收敛的开销和避免模型重复调整的开销.

2.5.1 模型集管理

模型调度器基于模型调节器产生的模型调整结果,通过模型集管理过程,识别和存储有效的应用模型并建立一个精简有效的模型集.

图 3 描述了模型集管理过程的逻辑结构(我们首先在识别窗口中对当前一系列模型执行在线聚类分析,然后选择最大的模型聚类作为当前控制系统的有效模型,再通过归类操作将这个有效模型记录在模型集中).过程涉及两个存储结构:识别窗口和模型集.我们使用识别窗口跟踪和记录应用模型的调整结果,并通过聚类操作发现当前的主要有效的应用模型.在识别窗口中,我们为每个模型聚类维护一组基本信息,包括代表聚类的典型模型、聚类的尺寸和组成成员等.当新成员加入到某个聚类时,这些基本信息被相应地更新.模型集是一个存储有效模型的专用表,其中,每个表项主要包括模型 ID、模型和模型签名.模型签名是某个能够描述应用模型有效范围的系统变量或系统指标,同时也具有区分和指示不同模型的功能.我们使用描述模型范围的二元形式〈中心,半径〉来描述模型签名.例如,假设使用应用请求到达率作为模型签名,则模型签名〈500,50〉表示模型的有效范围在 450 请求/s~550 请求/s.而且,当应用负载在此范围变化时,系统可以使用此模型签名对应的应用模型来驱动反馈控制器.

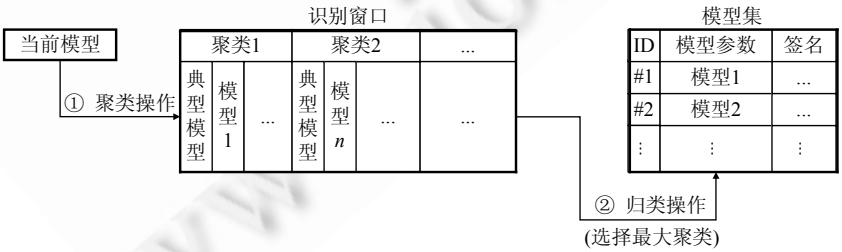


Fig.3 Procedure of model set management

图 3 模型集管理过程

模型集管理过程还涉及两个操作:聚类操作和归类操作.模型聚类操作用于识别当前控制系统的主要有效模型,它是基于一个增量式聚类算法实现.每当一个新模型产生时,聚类算法通过匹配运算,在识别窗口中寻找与该模型相近的模型聚类:如果找到,则将此模型归入与其最匹配的聚类中,并更新该聚类的基本信息;如果没有找到,则建立一个新的聚类.匹配运算通过两个模型的差异来判断它们是否匹配.这里,我们使用曼哈顿距离表示模型差异,它是两个向量的对应各数据项绝对差值之和.计算两个向量的曼哈顿距离是将向量看作多维空间中的两个点,而两点之间平行于坐标轴的最短距离则是曼哈顿距离,类似于城市中相隔若干街区两点的行车距离.例如,N-维空间中的向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的曼哈顿距离为

$$Dist(\boldsymbol{a},\boldsymbol{b})=\sum_{i=1}^{N}|\boldsymbol{a}_i-\boldsymbol{b}_i|.$$

在得到模型距离后,基于一个模型相似度阈值即可判断两个模型是否类似:如果两者的曼哈顿距离在该阈值范围内,则认为两个模型匹配;否则,判定二者相异.当距离阈值设置过小时,将形成许多小的模型聚类,难以识别出当前主要有效模型,偏离了使用模型调度器的初衷;而当距离阈值设置过大时,则模型识别精度降低,造成对重要模型的遗漏.

为了考察距离阈值对有效模型识别的影响,在实验中,我们使用离线聚类算法 K-means[10]的分析结果作为参考,检验不同距离阈值时本文在线聚类方法的精度.实验结果见第 3.5.1 节.K-means 算法通过在全局范围内发现模型变化结构来识别控制器主要状态,无须使用距离阈值,所以能够优化聚类过程.但是,本文提出的在线聚类方法能够满足在线分析要求并具有低开销的特点,虽然会导致一定的误差,但实验显示可控制在能够被接受的范围内.

当识别窗口填满后,其中最大的聚类就是当前控制系统的主要有效模型.然后,通过归类操作记录该有效模型.与聚类操作类似,归类操作使用匹配运算来判断该有效模型是否已经存在于模型集中:如果存在,则归入该有效模型并更新模型签名;如果不存在,则作为新的有效模型记入模型集,并分配一个唯一的模型 ID.

2.5.2 模型签名

前面提到,模型签名描述了应用模型的有效范围,并且能够区分和指示不同模型.为了选择合适的系统变量或系统指标作为模型签名,我们定义了“变量-模型-关联”指标来评价其作为模型签名的能力.该评价指标的含义是:某系统变量的值,如果在不同时刻相同时,对应的应用模型也相同,则认为该变量与模型变化具有关联性.这种关联性越强,该系统变量指示模型的能力越强.

“变量-模型-关联”指标的计算方法如下:假设模型调节器产生的所有模型,两两组对可构成 n 对;并且设 m_i 表示第 i 对模型的差异,而 v_i 为对应的待评价的系统变量的差值.这里,我们还是使用曼哈顿距离描述模型差异.“变量-模型-关联”指标通过如下公式计算:

$$p_{v,m}=\frac{Covariance(v,m)}{\sigma_v\cdot\sigma_m}=\frac{\sum_{i=1}^{n}(v_i-\bar{v})(m_i-\bar{m})}{n\cdot\sigma_v\cdot\sigma_m} \tag{6}$$

这里,$\bar{v}$ 和 σ_v 分别是 v_i 的平均期望和非零标准差,$\bar{m}$ 和 σ_v 分别是 m_i 的平均期望和非零标准差.$P_{v,m}$ 值越大,表示系统变量与模型变化的正相关性越强,即变量指示模型的能力越强.注意,该系数的值最大不超过 1.

在实验中,我们评价了应用负载作为模型签名的能力,结果见第 3.5.2 节的实验.

2.5.3 模型调节器结合模型调度器的资源管理

在建立起模型集后,我们同时通过模型调节和模型调度的方式来调整反馈控制器.在每个控制周期里,我们的资源管理系统首先使用模型调度器调整应用模型.模型调度器基于作为模型签名的系统变量的当前观察值,在模型集中寻找覆盖这个观察值的模型签名,而它所对应的模型就是适合当前情况的应用模型.例如,假设应用负载作为模型签名,则请求到达率为 300 请求/s 时,模型集中模型签名〈280,40〉对应的模型被用于配置反馈控制器.如果在模型集中没有找到合适的模型,则管理系统依然使用模型调节器在线估计新的模型参数.同时,无论通过模型调节器还是模型调度器调整应用模型,模型集管理过程都持续维护和更新模型集.

3 实验评价

3.1 实验环境

我们在一个基于Xen的机群系统上实现和评价了本文的虚拟化资源管理方法.我们首先在6台服务器上评价了本文方法实现应用服务级目标的效果.然后在不同系统规模下评价了本文方法的可扩展性.实验环境如图4 所示,其中,3 台应用服务器配置有双路 1.6GHz Opteron 处理器,2G 内存,146G 磁盘和 1000M 网卡,运行 OpenSuSE 10.3 和 Xen 3.1(适合 Linux2.6.22.5-31 SMP 内核);同样的,另外一台服务器运行所有测试应用的资源控制器;还有另两台用于模拟测试应用客户端的服务器配置有双路1GHz PIII处理器,1G内存,56G磁盘和100M网卡,同样运行 OpenSuSE 10.3 和 Xen 3.1.服务器通过自适应千兆交换机连接.

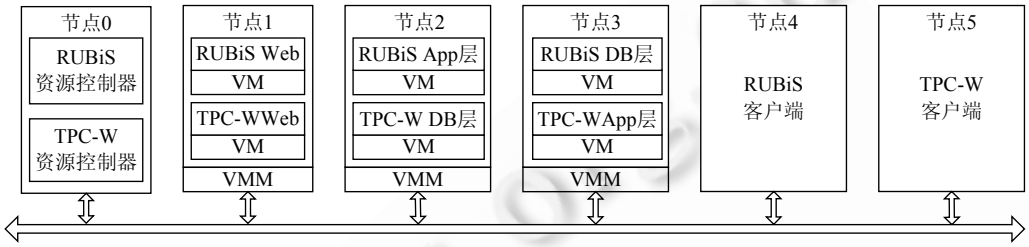


Fig.4 Setup of experimental environment

图 4 实验环境部署

实验中,我们使用了两个标准测试应用:RUBiS 系统[6]和 TPC-W 基准[7].

- RUBiS 系统是一个由 Rice 大学基于 EJB 开发的 Internet 应用程序,模拟类似于 ebay 商业模式的在线拍卖网站,具备在线拍卖网站的核心功能.该系统采用 3 层结构:前端是 Web 服务器,后端是 MySql 数据库系统,中间层是 Java 语言实现的应用逻辑.RUBiS 共支持 22 类请求,主要包括浏览、竞标和查看用户竞标历史等操作.RUBiS 官方网站提供了 Browsing 和 Bidding 两种标准负载,分别模拟用户浏览和竞价行为.
- TPC-W 基准测试是另一个由 Java 语言实现并用于模拟事务型电子书店的测试基准系统.与 RUBiS 类似,该基准测试也是 3 层结构,应用层包括 Image 应用服务器和 Web Cache 服务器.TPC-W 共支持 14 类请求,主要包括书籍搜索、用户注册和价格更新管理等操作,并提供了 3 种标准负载:Browsing, Shopping 和 Ordering,分别模拟用户浏览、采购和订购行为.

3.2 实验设计

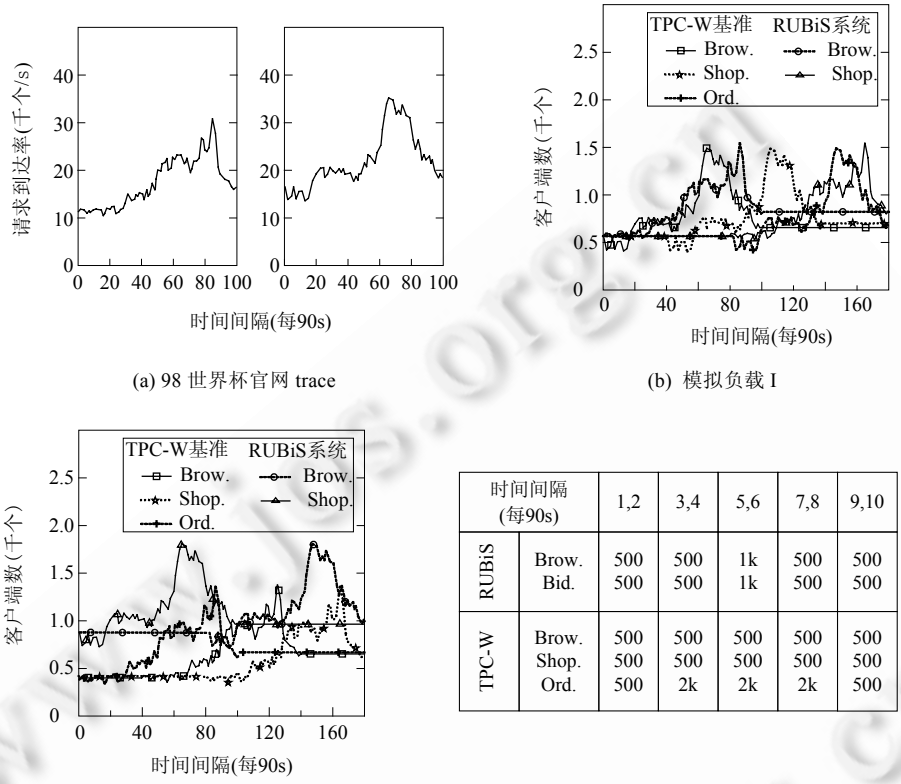
我们的实验目的是,检验本文资源管理系统实现应用服务级目标的效果.为了评价实验结果,我们采用应用目标实现率和应用目标平均偏离率作为实验评价指标.应用目标实现率用于描述在实验过程中,所有应用实测性能符合应用目标的比例;应用目标平均偏离率则表示违背目标的应用行为的平均严重程度.我们使用请求平均响应时间和吞吐率描述应用服务级目标.

在应用负载方面,我们模拟了 3 种负载情形,如图 5 所示.图 5(a)是 1998 年世界杯官方网站的请求到达率在两个时间段的变化记录[11].我们使用这些 trace 合成了表现应用负载波动的模拟负载 I 和模拟负载 II,分别如图 5(b)和图 5(c)所示,用于表现应用负载波动.在模拟负载中,我们通过调整应用客户端模拟了请求到达率变化,并且分别模拟了不同负载的变化,如 RUBiS 系统的 browsing 负载和 bidding 负载.图 5(d)的模拟负载 III 模拟了导致应用竞争非虚拟化资源磁盘 I/O 的情形.我们通过增加 TPC-W 基准的磁盘密集型负载 ordering 和 RUBiS 系统的负载来对节点 2 的磁盘 I/O 施加压力.

为了评价本文方法的有效性,我们将本文方法的实验结果与其他两个方法做了对比.Baseline 方法同样基于反馈控制原理,但是通过离线的系统辨识过程建立驱动反馈控制机制的应用模型.这种离线的系统辨识过程

是目前基于控制论原理的系统管理方法中最常用的方法.而 Xen 的默认资源分配方式,允许虚拟机任意地使用系统资源.

另外,第 2.3 节中提到,当应用资源需求冲突时,需要根据优先权执行实际的资源分配.实验中,我们设置 RUBiS 系统的资源分配权重 W_{rubis}=3,和 TPC-W 基准的资源分配权重 W_{tpcw}=1.这表示 RUBiS 系统的资源需求优先被满足.



(a) 98 世界杯官网 trace

(b) 模拟负载 I

(c) 模拟负载 II

时间间隔(每90s)		1,2	3,4	5,6	7,8	9,10
RUBiS	Brow.	500	500	1k	500	500
	Bid.	500	500	1k	500	500
TPC-W	Brow.	500	500	500	500	500
	Shop.	500	500	500	500	500
	Ord.	500	2k	2k	2k	500

(d) 模拟负载 III

Fig.5 Workload simulation

图 5 实验负载

本文的实验包括 6 个方面:第 3.3 节检验本文的应用模型的精度;第 3.4 节评价基于模型调节的资源管理方法实现应用服务级目标的效果;第 3.5 节是对模型集管理过程的评价;第 3.6 节评价基于模型调节结合模型调度的资源管理方法实现应用服务级目标的效果;第 3.7 节是对本文方法实现虚拟机隔离的评价;第 3.8 节评价本文方法的可扩展性.

3.3 应用模型精度评价

在本节的实验中,我们使用模拟负载 I,利用应用性能预测误差评价本文应用模型的精度.设平均预测误差

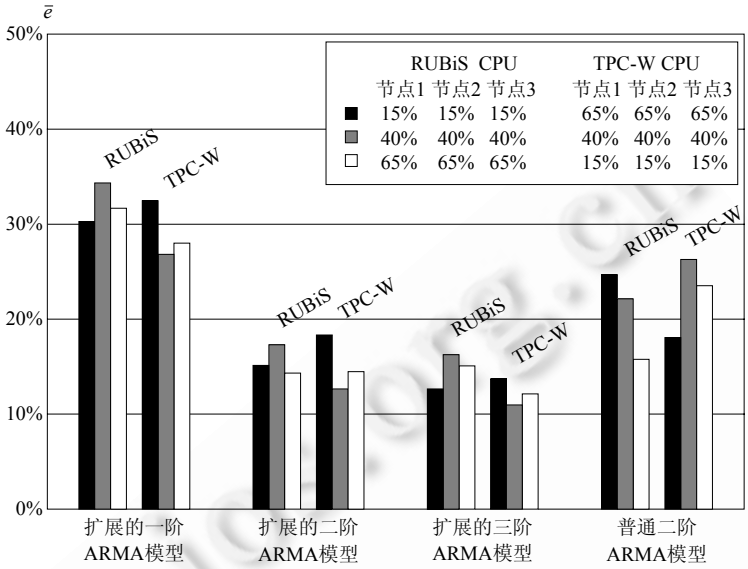
$$\overline{e}=\frac{\sum_{k=1}^{n}|y'(k)-y(k)|}{n},$$

其中,$y'(k)$是应用预测性能,$y(k)$是应用实测性能,n 是总周期数.

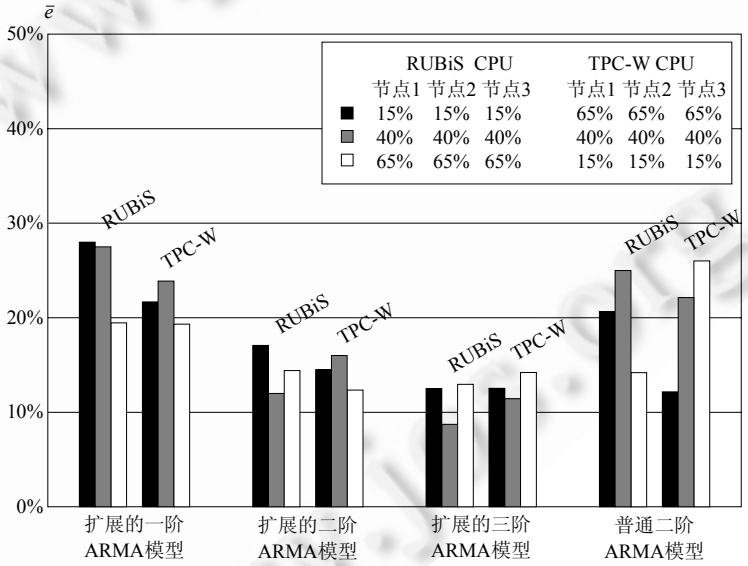
在第 2.2 节中已经提到,我们使用扩展的 ARMA 模型作为应用模型.实验中,我们比较了扩展的 ARMA 模型在 3 种形式的精度,同时还比较了普通二阶 ARMA 模型的精度.

同样地,在第 2.2 节中提到,扩展的 ARMA 模型与普通 ARMA 模型的不同是,扩展的 ARMA 模型的资源分配向量包括了对不同应用资源分配的描述,能够量化应用在非虚拟化资源上的竞争干扰.

图 6 比较了以上 4 种 ARMA 模型用于预测 RUBiS 系统和 TPC-W 基准,在 3 种处理器资源分配情形的平均响应时间和吞吐率的实验结果.



(a) 应用平均响应时间预测误差



(b) 应用吞吐率预测误差

Fig.6 Accuracy comparison of ARMA models

图 6 ARMA 模型精度对比

实验结果显示:

- 扩展的二阶 ARMA 模型总体的平均预测误差(包括平均响应时间和吞吐率)是 14.4%,与一阶 ARMA 模

型的总体平均预测误差 26.8%相比,模型的精度有较大的改善.这主要是由于一阶 ARMA 模型的随机冲量未考虑资源历史分配情况,使得时序模拟不够全面.

- 三阶 ARMA 模型的平均预测误差是 12.1%,虽然与二阶 ARMA 模型相比又有所提高,但是从可扩展性和计算开销的角度考虑,优势并不明显.同时,二阶的普通 ARMA 模型的平均预测误差是 22.7%,模型精度明显低于扩展的二阶 ARMA 模型的精度.这主要是由于普通 ARMA 模型将应用对非虚拟化的磁盘 I/O 资源的竞争引起的性能影响,不恰当地转化为应用的处理器资源需求.这样会在两方面恶化应用性能预测误差:
  1) 处理器资源预计需求进一步偏离其实际消耗量,导致应用性能与资源分配关系出现较大偏差;
  2) 仅通过增加对处理器资源的需求,并不能有效地避免其他应用在磁盘 I/O 资源的干扰.
- 扩展的 ARMA 模型能够尽量避免以上两个恶化模型精度的因素:
  1) 首先,扩展的 ARMA 模型使用其他应用的资源分配来分担影响应用性能的来源,避免应用处理器资源预计需求与实际消耗出现较大偏差;
  2) 其次,能够通过干涉其他应用的处理器资源分配来控制其他应用对磁盘 I/O 资源竞争导致的性能上的影响.

综合上面的比较结果,本文选择扩展的二阶 ARMA 模型作为应用的目标控制模型.

3.4 基于模型调节的资源管理

在本节的实验中,我们使用模拟负载 I 对比了本文方法与 Baseline 方法和 Xen 默认方式,在应用目标实现率指标方面和应用目标平均偏离率指标方面的实验结果.

图 7 是 3 种资源管理方法实现 RUBiS 系统和 TPC-W 基准,在平均响应时间目标和吞吐率目标上的效果对比(其中,应用目标实现率描述了所有应用实测性能符合应用目标的比例,应用目标平均偏离率描述了违背应用目标的应用实测性能的平均严重程度).

实验中,我们设置 RUBiS 系统的平均响应时间目标为不大于 1s,TPC-W 基准的平均响应时间目标为不大于 1.5s.实验结果显示:本文方法的应用平均响应时间目标实现率平均比 Baseline 方法高 21.6%,比 Xen 默认方式高 33.3%;同时,应用平均响应时间目标平均偏离率比 Baseline 方法低 65.3%,比 Xen 默认方式低 62%.

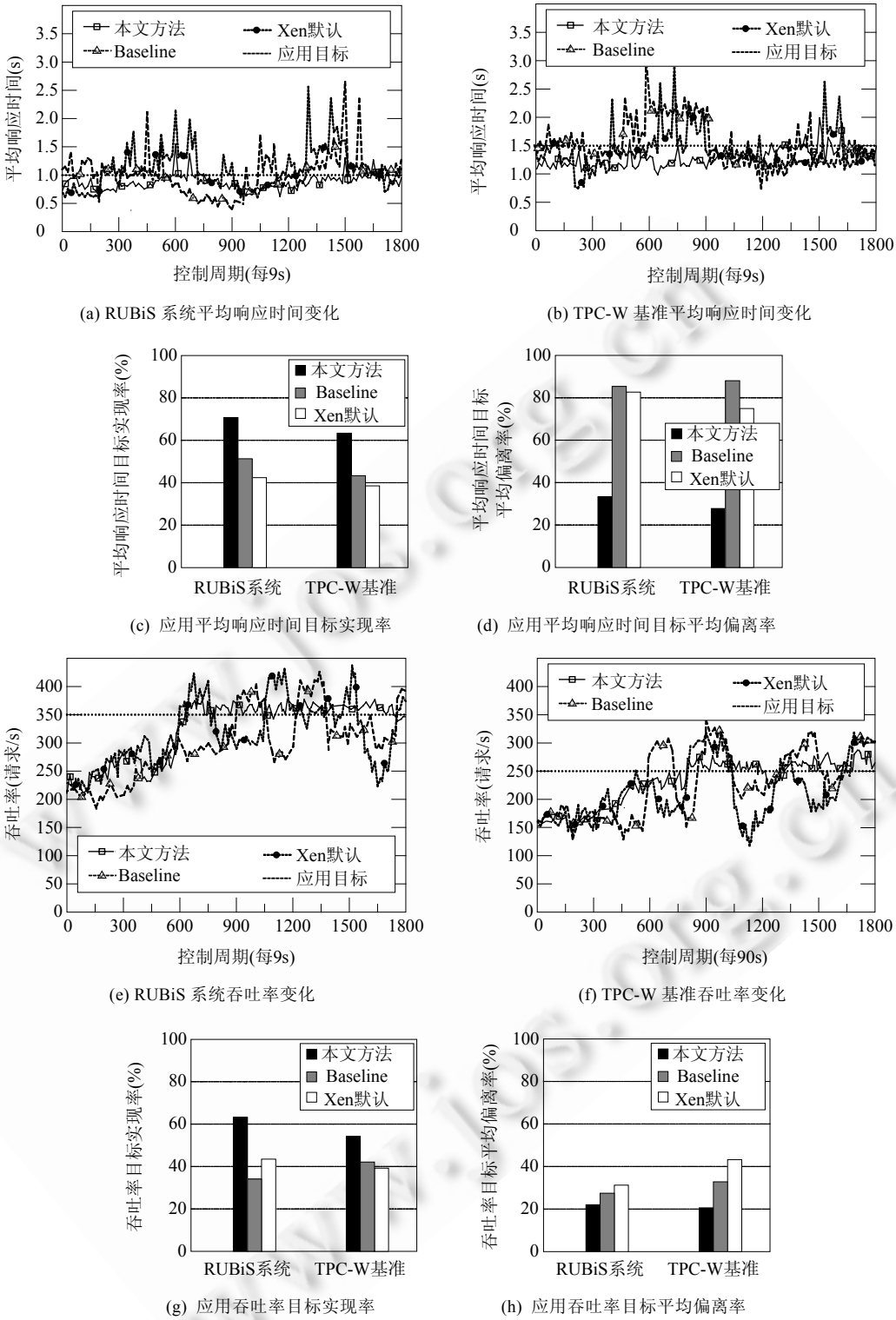
在应用吞吐率方面,我们设置 RUBiS 系统的吞吐率目标为 350 个请求/s,TPC-W 基准吞吐率目标为 250 个请求/s.实验结果显示:本文方法的应用吞吐率目标实现率平均比 Baseline 方法高 32.5%,比 Xen 默认方式高 28.9%;同时,应用吞吐率目标平均偏离率比 Baseline 方法低 30.5%,比 Xen 默认方式低 43.1%.

需要说明的是,因为模拟负载中存在较低请求到达率的情况,所以在这些时候,本文方法也不能很好地实现吞吐率目标.

总之,实验结果说明,本文方法能够更好地应对负载波动和应用变化,并通过资源分配将应用性能控制在目标附近.

图 8 是实验中 RUBiS 系统平均响应时间模型的部分参数的变化情况.可以看出,模型参数变化具有重复性和规律性的特点.

在下一节的实验中,我们将通过模型集管理过程来发现和记录这种规律性.



(a) RUBiS 系统平均响应时间变化

(b) TPC-W 基准平均响应时间变化

(c) 应用平均响应时间目标实现率

(d) 应用平均响应时间目标平均偏离率

(e) RUBiS 系统吞吐率变化

(f) TPC-W 基准吞吐率变化

(g) 应用吞吐率目标实现率

(h) 应用吞吐率目标平均偏离率

Fig.7 Comparison of our method to Baseline method and Xen default method for application's SLO

图 7 对比本文方法与 Baseline 方法和 Xen 默认方式实现应用服务级目标的效果

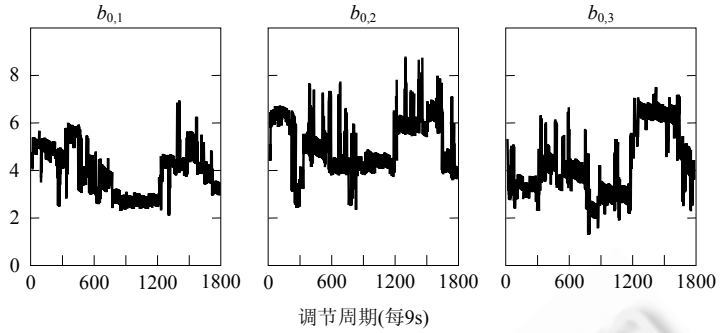


Fig.8 Variation of partial parameters of RUBiS response time model

图 8 RUBiS 系统平均响应时间模型的部分参数变化

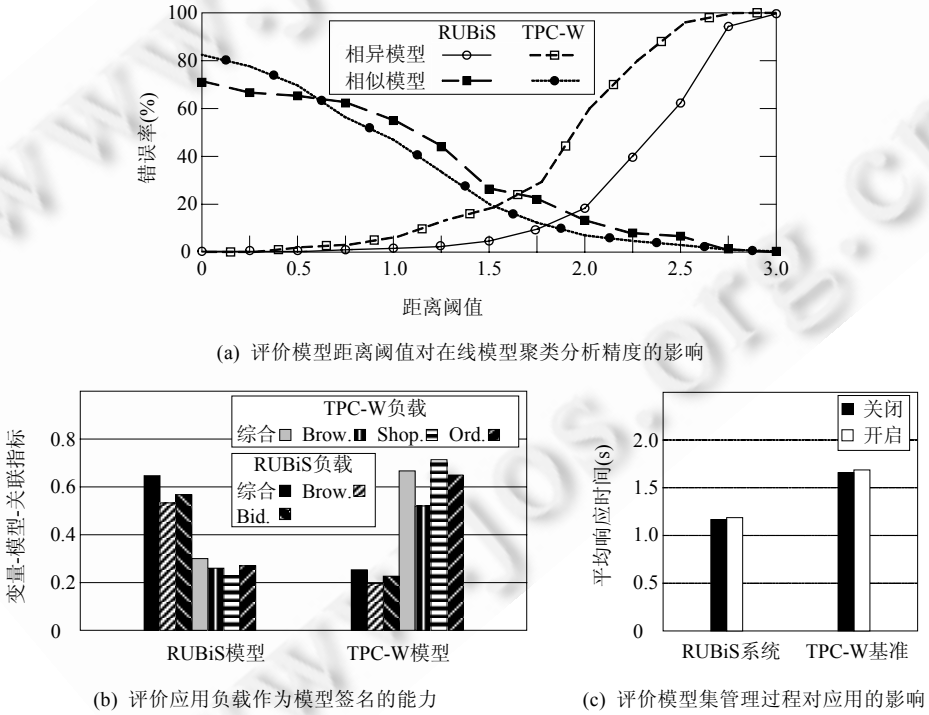
3.5 模型集管理过程

在本节实验中,我们使用第 3.3 节实验得到的应用模型,通过第 2.5.1 节的模型管理过程来建立应用有效模型的集合,同时检验了影响模型集效果的 3 个关键因素:模型距离阈值、模型签名和开销.

3.5.1 距离阈值对模型集的影响

第 2.5.1 节中提到,模型距离阈值是在线模型聚类分析的关键.距离阈值设置过高或过低都将影响模型集效果.为了检验阈值对在线聚类操作的影响,我们将离线聚类方法 *k*-means[10]分析结果作为我们方法的参照.*k*-means 算法能够在全局范围内优化聚类过程,因此不需要依赖距离阈值.

图 9(a)是不同距离阈值影响在线聚类操作精度的实验结果.



(a) 评价模型距离阈值对在线模型聚类分析精度的影响

(b) 评价应用负载作为模型签名的能力

(c) 评价模型集管理过程对应用的影响

Fig.9 Evaluation of the procedure of model set management

图 9 评价模型集管理过程

在图 9(a)中,相异模型曲线刻画了在不同距离阈值时,在线聚类算法区分不同模型的能力.例如,当距离阈值

设为 2 时,对于 RUBiS 系统平均响应时间模型,在线聚类算法识别出 78%的 *K*-means 分析得到的不同模型,另外 22%的不同模型被误认为与其他模型类似.相似模型曲线刻画了在距离阈值不同时,在线聚类方法归类相似模型的能力.同样,当距离阈值为 2 时,对于 RUBiS 系统平均响应时间模型,在线聚类方法识别出了 83%的 *k*-means 方法得到的相似模型,而将另外 17%相似模型误认为不同.由此可知,合适的距离阈值能够使得在线聚类分析方法具有可接受的精度.TPC-W 基准的实验结果类似.

在后面的实验中,我们在模型调度方式中选用了距离阈值为 2 时建立的 RUBiS 系统模型集和距离阈值为 1.5 时建立的 TPC-W 基准模型集.表 1 是 RUBiS 系统的模型集结果,共包括了 9 类模型.实验中,在线聚类分析的规模是 1 800 个应用模型,其中,78.2%的模型被归入这 9 类模型.

Table 1 Set of models for RUBiS average response time objective

表 1 面向 RUBiS 系统平均响应时间目标的模型集结果

模型 ID	模型参数													模型签名	
	a_1	向量 $\boldsymbol{b}_0$						向量 $\boldsymbol{b}_1$						中心	半径
1	−0.16	4.82	4.09	3.18	−2.96	−0.82	−0.89	−0.42	−2.29	−2.42	−1.69	−1.38	0.36	1 134	73
2	−0.87	4.05	5.53	6.55	−2.05	0.15	−0.25	1.54	−0.47	−1.56	−0.58	−1.53	−1.38	1 252	32
3	−0.48	3.96	5.92	4.26	−1.32	−0.66	0.92	1.56	1.31	−1.46	1.26	1.64	−1.46	1 375	83
4	−0.38	2.61	3.46	4.12	−2.61	−0.61	−1.06	−1.54	−1.76	−1.29	−1.71	−1.45	−1.28	1 568	89
5	−0.15	4.82	5.92	3.18	−1.32	−0.82	0.92	−2.62	−1.36	−1.16	−1.37	−1.32	−1.31	1 875	191
6	−0.56	5.12	3.17	4.82	−2.12	1.12	−1.42	−0.51	−1.52	−0.71	−1.61	−0.25	−0.35	2 053	42
7	−0.17	4.05	3.82	6.55	−2.82	0.15	−0.08	1.68	1.39	0.52	1.38	1.49	0.58	2 171	106
8	−0.96	4.02	4.12	4.02	−2.08	−0.84	−1.02	−0.38	0.71	1.79	−0.36	−1.47	−1.23	2 318	58
9	−0.77	2.82	3.73	2.92	−2.64	−0.94	−1.96	1.61	−0.61	2.59	1.43	2.62	1.46	2 475	36

3.5.2 系统变量作为模型签名的能力

第 2.5.2 节中提到,模型签名是模型集里用于模型选择和描述模型有效范围的某系统变量.为了挑选合适的系统变量,我们提出变量-模型-关联指标来评价系统变量作为模型签名的能力.该指标量化了系统变量与模型变化的关联性.本文主要考虑使用应用负载作为模型签名.

图 9(b)是各种应用负载分别与 RUBiS 系统模型和 TPC-W 基准模型的变量-模型-关联指标结果(变量-模型-关联指标量化描述了系统变量与模型变化的关联性).根据结果,我们选择 RUBiS 系统的综合负载和 TPC-W 基准的 shopping 负载分别作为 RUBiS 系统和 TPC-W 基准的模型签名.

3.5.3 模型集管理过程的开销

我们还考察了模型管理过程对应用的影响.图 9(c)对比了模型集管理过程分别在开启和关闭状态时的应用平均响应时间.可以看出,模型集管理操作对应用的性能影响基本上可以忽略.RUBiS 系统和 TPC-W 基准的平均响应时间的变化分别在 2.6%和 3.1%.

3.6 基于模型调节结合模型调度的资源管理

本节实验中,我们分别在模拟负载 I 和模拟负载 II 的情形下对比了混合方式、模型调节方式和模型调度方式在应用目标实现率指标上的实验结果.其中,模型调度方式使用第 3.4 节实验的模型集结果.混合方式指模型调节结合模型调度的方式.

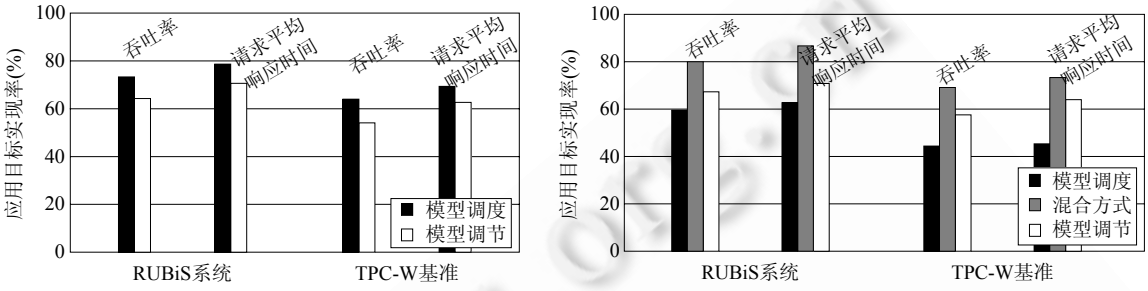
图 10(a)是在模拟负载 I 的情形下,模型调度方式与模型调节方式的应用目标实现率的对比(混合方式指模型调节结合模型调度的方式).

实验结果显示,模型调度方式无论是在平均响应时间还是吞吐率方面,都不仅到达了模型调节方式的效果,而且提高了应用目标实现率,分别平均提高 14.8%和 21.5%.这主要是因为当应用发生变化时,特别是突发性的变化,模型调节方式通常需要经过模型收敛过程才能找到一个稳定而有效的应用模型.模型调度方式可以直接通过模型集配置应用模型,避免了模型收敛过程,从而减少了模型的无效的临时状态.此外,模型集是在模拟负载 I 的情形下建立的,所以很好地覆盖了模拟负载 I 时应用的各种变化.

图 10(b)是在模拟负载 II 的情形下,混合方式、模型调节方式和模型调度方式的应用目标实现率的比较.

从实验结果可知,混合方式的应用目标实现率比模型调度方式平均高 43.5%,比模型调节方式平均高 18.5%.另外需要注意到,图 10(b)中的模型调度方式与模型调节方式相比,没有出现类似于图 10(a)的优势.这主要是因为模拟负载 II 与模拟负载 I 存在较大差异,在模拟负载 I 情形下建立的模型集不能很好地覆盖应用在模拟负载 II 情形下的变化.所以,通过模型调度配置应用模型,会出现因为配置错误导致反馈控制器处于失效状态的情况.这说明,混合方式使得两种自适应方法互相弥补了对方的不足.

- 模型调节方式能够动态地适应应用变化,但是必须付出模型收敛开销的代价;
- 而模型调度方式能够直接配置应用模型,避免了模型收敛过程,却不能覆盖应用的所有变化,因此需要基于比较完备的模型集.



(a) 模拟负载 I (b) 模拟负载 II

Fig.10 Comparison of three adaptation mechanisms of our method for application's SLO

图 10 对比分别基于 3 种自适应方式时,本文方法实现应用服务级目标的效果

3.7 虚拟机性能隔离

在本节实验中,我们使用模拟负载 III 的情形来检验本文方法控制虚拟机竞争磁盘资源的效果.

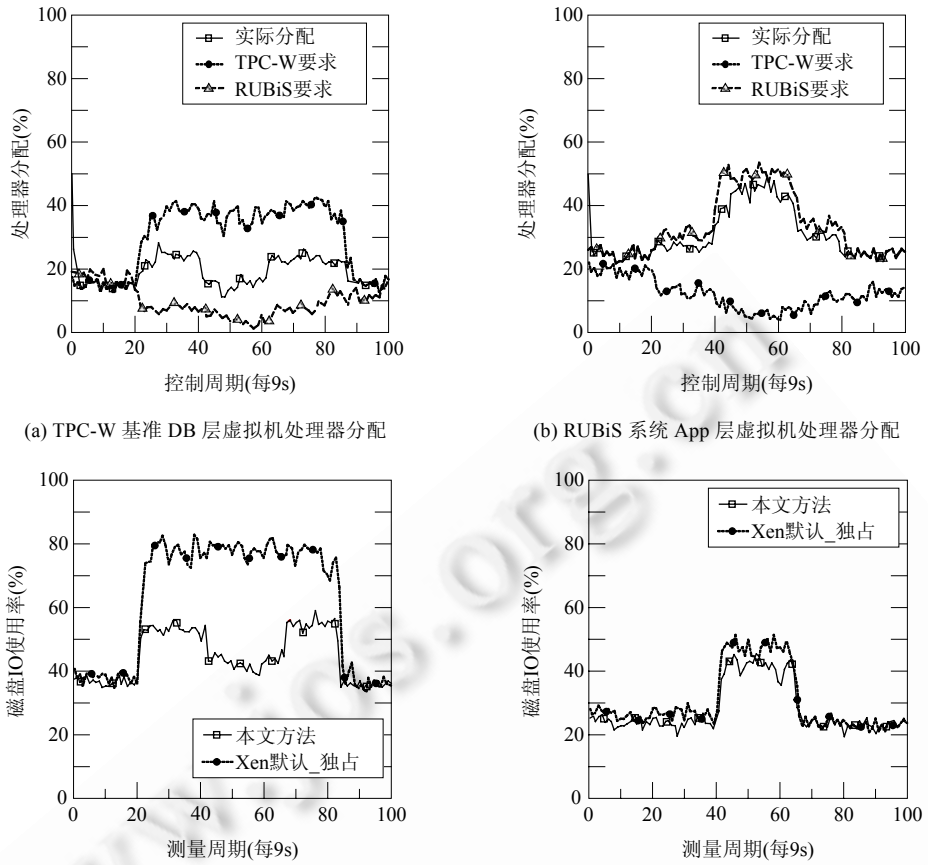
前面提到,在模拟负载 III 中,我们通过增加 TPC-W 基准的磁盘密集型负载 ordering 和 RUBiS 系统的负载来对节点 2 的磁盘 I/O 施加压力.在实验中,我们观察了应用在节点 2 的处理器资源分配情况和磁盘 I/O 使用率情况.

图 11(a)和图 11(b)分别是 TPC-W 基准和 RUBiS 系统在节点 2 上的处理器需求与实际分配的对比结果(比较应用处理器资源需求与实际分配的差异,以及应用磁盘 I/O 需求与实际使用率的差异).

可以看到,TPC-W 基准和 RUBiS 系统的处理器资源要求没有超出服务器能力,最高时分别为 38.5%和 51.4%.但是,实际的处理器分配并没有满足应用需求,尤其对于 TPC-W 基准,实际分配比它的需求平均低了 28.7%,而 RUBiS 系统则只有 4.6%.

另一方面,图 11(c)和图 11(d)分别是 TPC-W 基准和 RUBiS 系统对节点 2 磁盘资源需求与实际使用情况的对比结果.由于在本文中磁盘资源是非虚拟化资源,所以不能直接获得应用的磁盘 I/O 需求.因此,我们通过虚拟机独占节点方式,观察 Xen 默认资源分配方式时应用的磁盘 I/O 使用率来估计应用的磁盘 I/O 需求.

从图中可以看到,两个应用对磁盘 I/O 的总需求超出了服务器的磁盘能力.第 20 周期~第 82 周期,TPC-W 基准的磁盘 I/O 使用率平均在 78.4%.RUBiS 系统的磁盘 I/O 需求,前 40 个周期和后 40 个周期平均在 27.4%,第 40 周期~第 63 周期平均在 48.1%.而在使用我们的方法时,RUBiS 系统在节点 2 的磁盘 I/O 需求几乎被完全满足,实际分配比需求平均低 4.7%,而 TPC-W 的实际分配平均比需求低 37.1%.这说明,当应用在节点 2 的磁盘资源上发生竞争时,我们的方法能够通过减少 TPC-W 基准在该节点的处理器分配来抑制它对磁盘资源的消耗,从而保证 RUBiS 系统的资源需求得到满足,从而实现了在非虚拟化资源上的虚拟机性能隔离.



(a) TPC-W 基准 DB 层虚拟机处理器分配

(b) RUBiS 系统 App 层虚拟机处理器分配

(c) TPC-W 基准 DB 层虚拟机磁盘 I/O 使用率变化

(d) RUBiS 系统 APP 层虚拟机磁盘 I/O 使用率变化

Fig.11 Evaluation of our method for VM isolation

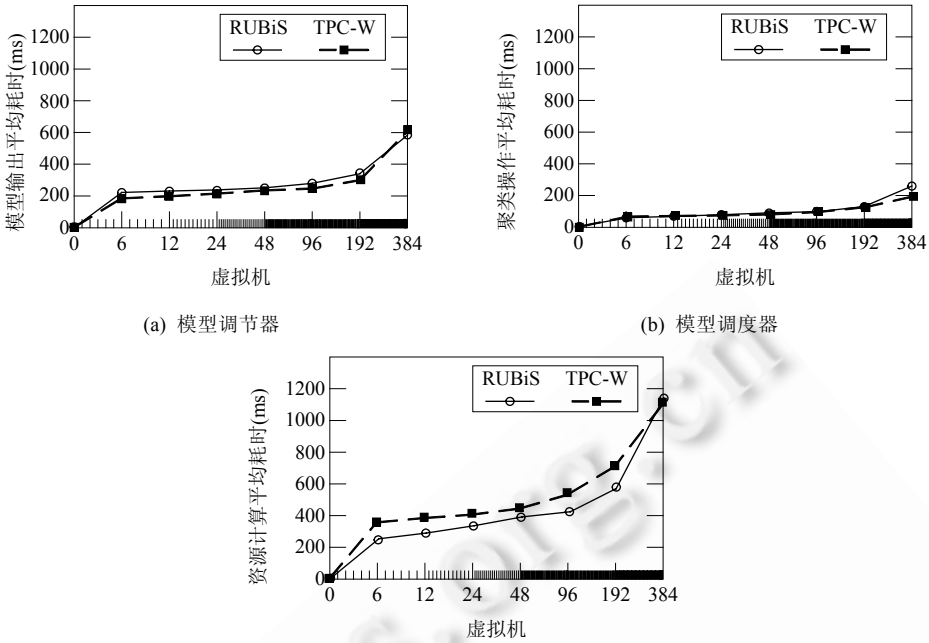
图 11 评价本文方法实现虚拟机隔离的效果

3.8 可扩展性

本节实验中,我们在 7 种系统规模中检验了图 2 所示的模型调节器、模型调度器和反馈控制器这几个本文方法关键部件的可扩展性.这 7 种系统规模分别是:

- 在 3 台服务器上分别启动 6 个虚拟机、12 个虚拟机、24 个虚拟机、48 个虚拟机、96 个虚拟机;
- 在 6 台服务器上启动 192 个虚拟机;
- 在 12 台服务器上启动 384 个虚拟机.

图 12 是 3 个部件可扩展性实验结果.其中,图 12(a)显示了在不同系统规模下,模型调节器输出模型的平均耗时.可以看到,随着系统规模的扩大,模型输出时间的变化不大.当系统规模达到 384 个虚拟机时,模型输出的平均时间在0.6ms左右.图12(b)是在不同系统规模下,模型调度器在线模型聚类操作的平均耗时.可以看到,聚类操作耗时随着系统规模的扩大,变化不大,当系统达到实验的最大规模时,聚类操作的平均耗时在 0.2s 左右.图12(c)显示了在不同系统规模下,反馈控制器计算应用资源需求的平均耗时.可以看到,与模型调节器和模型调度器相比,反馈控制器的开销随着规模的扩大,变化较为明显.当规模达到 96 个虚拟机时,TPC-W 基准的反馈控制器耗时有较为明显的增幅.但是即便如此,当系统规模达到384个虚拟机时,资源计算耗时仍然不大,在1.1s左右.



(a) 模型调节器

(b) 模型调度器

(c) 反馈控制器

Fig.12 Evaluation of scalability of our method

图 12 评价本文方法的可扩展性

4 相关工作

4.1 动态资源分配

在近年来面向共享环境的资源动态管理研究中,具有代表性的工作有文献[1,3,12]等.其中,Aron 等人提出一种在机群系统中实现分布式的应用隔离的方法[3].该方法通过资源分配确保不同应用的资源使用率来实现它们的性能隔离,将资源分配问题形式化为一个在线优化问题的求解过程.该方法将单节点的应用隔离机制 Resource Container[13]扩展到机群规模.Jeffrey 等人面向数据中心环境设计和实现了一个具有能耗效率意识的资源管理系统 Muse[1].该方法通过一个竞价模型,在 SLA 允许范围内动态地调整应用资源分配,实现了能耗开销与资源收益的平衡.Shen 等人在 Neptune[14]中实现了一个机群系统的应用资源管理框架[12].该方法以最大化系统的整体资源效率为目标,并支持对不同服务的区分.它通过两层结构的请求分发和调度机制实现.在机群层面,每类服务的请求被平均地发送给该服务的所有副本服务器;在服务器节点层面,通过一个自适应的请求调度策略,力求在不同负载情况下最大化服务器资源的整体效率.

与这些工作不同的是:

- 本文的工作面向虚拟环境中多层应用的服务级目标,而以上工作面向应用隔离、能耗效率和资源效率等目标.
- 其次,我们的方法基于控制论原理.
- 最后,在虚拟机层次,我们的方法考虑了应用内部模块资源需求的相关性.文献[1,3]针对的是多实例的单层应用,没有考虑这种内部关系;而文献[12]可以作为本文方法的补充,实现应用内部的请求服务区分.

4.2 控制论应用

基于控制论的反馈控制方法近来被广泛应用于计算机系统的管理任务,如性能管理[15−19]、资源分配[4,20]等.这些工作面向不同管理目标的研究,都提出一种基于反馈控制机制,通过系统参数调节来适应计算机系统内部和外部动态性的方法.Kamra 等人使用自调节 PI 控制器设计了一个请求接纳控制机制,实现了多层应用的过载保护和响应时间限定[15].Padala 等人提出一项基于反馈控制机制的多层应用资源控制技术[4].Lu 等人面向不同 Web 内容服务的性能区分目标,基于反馈控制技术设计了一种 Web cache 资源分配方法[20].Abdelzaher 等人提出使用反馈控制方法解决 Web 应用性能管理问题,主要包括服务性能隔离、服务质量区分、服务质量自适应等[16].Karlsson 等人在存储系统管理中设计了一个基于反馈控制的请求吞吐率控制机制来实现不同客户端请求的性能隔离[17].江滢等人使用 PI 控制器设计了一个基于接纳时间比控制和比例积分调节器的接纳控制机制[18],用于 Web 服务器过载保护.王晓川等人提出一种基于模糊控制理论的应用 QoS 量化控制方法,用于 Web 应用性能管理[19].

在这些工作中,除了文献[15,17,19]以外,都通过离线的系统辨识过程并采用静态确定性模型作为系统模型.本文使用在线的模型调节方法更适合于时变的多层应用系统.与文献[15,17,19]不同的是,本文提出一种前摄式的基于模型配置的自适应方式,与模型调节方式结合,提高了我们方法的自适应能力.

5 结 论

本文探讨了一种适用于虚拟环境的资源管理方法.该方法通过动态的虚拟机资源分配和虚拟机性能隔离来确保数据中心多层应用的服务级目标.我们为多层应用设计了一个基于控制论原理的应用资源控制器.资源控制器根据应用服务级目标,周期性地计算出应用虚拟机的资源需求.然后,通过一个两层结构的自适应机制,应用资源控制器能够动态地捕捉虚拟机资源分配与应用服务级目标的时变的非线性关系.最后,我们为服务器设计了资源分配器.服务器资源分配器通过仲裁应用资源请求来实现虚拟机性能隔离.实验中,我们在基于 Xen 的虚拟环境中测试了本文方法在 RUBiS 系统和 TPC-W 基准上的效果.实验结果显示,本文方法的应用服务级目标实现率比两种对比方法平均高 29.2%,而应用服务级目标平均偏离率比它们平均低 50.1%.另一方面,当 RUBiS 系统和 TPC-W 基准竞争非虚拟化的磁盘 I/O 资源时,本文方法通过抑制 TPC-W 基准 28.7%的处理器资源需求来优先满足 RUBiS 系统的磁盘 I/O 需求.实验结果说明,我们的方法能够应对应用负载的波动,按需分配虚拟机资源,同时,通过虚拟化资源分配控制虚拟机在非虚拟化资源上的相互影响,从而确保应用服务级目标.

References:

[1] Chase JS, Anderson DC, Thakar PN, Vahdat AM. Managing energy and server resources in hosting centers. In: Proc. of the 18th Symp. on Operating Systems Principles (SOSP). 2001. 103−116. [doi: 10.1145/502034.502045]

[2] Crovella ME, Bestavros A. Self-Similarity in World Wide Web traffic: Evidence and possible causes. IEEE/ACM Trans. on Networking, 1997,5(6):835−846. [doi: 10.1109/90.650143]

[3] Aron M, Druschel P, Zwaenepoel W. Cluster reserves: A mechanism for resource management in cluster-based network servers. In: Proc. of the ACM SIGMETRICS Conf. 2000. 90−101. [doi: 10.1145%2f339331.339383]

[4] Wood T, Shenoy P, Venkataramani A, Yousif M. Black-Box and gray-box strategies for virtual machine migration. In: Proc. of the 4th USENIX Conf. on Networked Systems Design and Implementation (NSDI). 2007. 17.

[5] Li DS, Huang F, Wang XH. XML-Based modeling and implementation of distributed virtual execution environment. Journal of Computer Research and Development, 2009,46(Suppl.):277−282 (in Chinese with English abstract).

[6] RUBiS: Rice University bidding system. 2009. http://rubis.ow2.org/

[7] TPC-W benchmark. 2005. http://www.tpc.org/tpcw/

[8] Anderson BDO, Moore JB. Optimal Control: Linear Quadratic Methods. Englewood Cliffs: Prentice Hall, Inc., 1989. 2−4.

[9] Astrom KJ, Wittenmark B. Adaptive Control. 2nd ed., Boston: Addison-Wesley, 2008. 156−184.

[10] MacQueen J. Some methods for classification and analysis of multivariate observations. In: LeCam LM, Neyman J. In: Proc. of the 5th Berkeley Symp. on Mathematical Statistics and Probability. Berkeley: University of California Press, 1967. 281−297.

[11] Arlitt M, Jin T. Workload characterization of the 1998 World Cup Web site. Technical Report, HPL-1999-35R1, HP Laboratories, 1999.

[12] Shen K, Tang H, Yang T, Chu LK. Integrated resource management for cluster-based Internet services. ACM SIGOPS Operating Systems Review, 2002,36(SI):225−238. [doi: 10.1145/844128.844150]

[13] Banga G, Druschel P, Mogul JC. Resource containers: A new facility for resource management in server systems. In: Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI). 1999. 45−58.

[14] Shen K, Yang T, Chu LK, Holliday JL, Kuschner DA, Zhu HC. Neptune: Scalable replication management and programming support for cluster-based network services. In: Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems. 2001. 197−208.

[15] Kamra A, Misra V, Nahum E. Yaksha: A self tuning controller for managing the performance of 3-tiered Web sites. In: Proc. of the Int'l Workshop on Quality of Service (IWQoS). 2004. 47−56. [doi: 10.1109%2fIWQOS.2004.1309356]

[16] Abdelzaher T, Shin KG, Bhatti N. Performance guarantees for Web server end-systems: A control-theoretical approach. IEEE Trans. on Parallel and Distributed Systems, 2001,13(1):80−96. [doi: 10.1109/71.980028]

[17] Karlsson M, Karamanolis C, Zhu XY. Triage: Performance isolation and differentiation for storage systems. In: Proc. of the 12th IEEE Int'l Workshop on Quality of Service (IWQoS). 2004. 67−74. [doi: 10.1109/IWQOS.2004.1309358]

[18] Jiang Y, Meng D. Enforcing admission control using admission-time-ratio and PI controller. Journal of Computer Research and Development, 2007,44(1):65−70 (in Chinese with English abstract).

[19] Wang XC, Jin SY, Xia MB. Distributed quantitative QoS control based on control theory in Web cluster. Ruanjian Xuebao/Journal of Software, 2007,18(11):2810−2818 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2810.htm [doi: 10.1360/jos182810]

[20] Lu Y, Abdelzaher TF, Saxena A. Design, implementation, and evaluation of differentiated caching services. IEEE Trans. on Parallel and Distributed Systems, 2004,15(5):440−452. [doi: 10.1109/TPDS.2004.1278101]

附中文参考文献:

[5] 李东升,黄峰,王小海.基于 XML 的分布式虚拟运行环境建模与实现.计算机研究与发展,2009,46(增刊):277−282.

[18] 江滢,孟丹.基于接纳时间比控制和比例积分调节器的接纳控制机制.计算机研究与发展,2007,44(1):65−70.

[19] 王晓川,金士尧,夏明波.Web 集群中基于控制论的分布式 QoS 量化控制.软件学报,2007,18(11):2810−2818. http://www.jos.org.cn/1000-9825/18/2810.htm [doi: 10.1360/jos182810]

文雨(1976－),男,重庆人,博士,高级工程师,主要研究领域为云计算,数据中心系统管理,系统行为分析.
E-mail: rainix.oo@gmail.com

詹剑锋(1976－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为负载分析,操作系统,云计算系统.
E-mail: zhanjianfeng@ict.ac.cn

孟丹(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为高性能计算机体系结构和系统软件,高效通信协议,分布式文件系统,存储服务器,机群操作系统.
E-mail: mengdan@iie.ac.cn