多租户 Web 应用的 CPU 资源动态评估方法

王 伟[1),2),3)] 黄 翔[1),3)] 张文博[1)] 魏 峻[1)] 钟 华[1)] 黄 涛[1)]

[1)](中国科学院软件研究所软件工程技术研究中心 北京 100190)

[2)](武汉大学软件工程国家重点实验室 武汉 430072)

[3)](中国科学院研究生院 北京 100039)

摘 要 中间件共享是云计算模式中一种重要的资源共享方式.但是,这种方式容易导致宿主在同一中间件服务器上的多个租户间产生性能干扰.因此,需要为租户提供性能隔离的服务实例.在线度量租户对系统资源的使用情况是实现性能隔离的前提条件,但是,在共享中间件服务器中直接度量 CPU 资源需要注入探针,将引起性能开销,并依赖于操作系统的支持.最近,一些工作利用回归分析进行资源使用情况的间接评估,但仍难以对动态 Web 系统的时变资源状态进行有效评估.文中针对普遍使用的 Java 中间件服务器,提出一种基于 Kalman 滤波的多租户 Web 应用 CPU 资源动态评估方法,并通过两个实验案例分析方法的评估效果、影响因素和面临的挑战.实验结果表明,通过适当的参数设置,该方法可动态适应持续变化的负载环境,并且与直接度量方法相比,具有可接受的评估误差.实验还表明该方法可用于检测侵占型租户,并避免共享中间件服务器 CPU 过载.

关键词 CPU 资源评估;性能隔离;Kalman 滤波;多租户

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.02292

Dynamically Estimating Approach for CPU Consumption of Multi-Tenancy Web Applications

WANG Wei[1),2),3)] HUANG Xiang[1),3)] ZHANG Wen-Bo[1)] WEI Jun[1)] ZHONG Hua[1)] HUANG Tao[1)]

[1)](*Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[2)](*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072)

[3)](*Graduate University of Chinese Academy of Sciences, Beijing* 100039)

Abstract Middleware sharing is one of the important resource sharing approaches in cloud computing. However, a shared middleware server easily causes interference in performance between multiple hosted tenants. This interference affects infrastructure resources as well as applications and services that are hosted on shared resources but that need to be made available in multiple performance isolated instances. A key requirement in performance isolation of the shared Java middleware server is the knowledge of the resource consumption of the various tenants. However, direct measurement of CPU resource consumption requires instrumentation, incurs overhead, and assumes OS support. Recently, regression analysis has been applied to indirectly approximate resource consumption, but challenges still remain in estimating time-varying states in dynamic systems. In this paper, we propose a Kalman filter-based approach to offer a solution to the problem of dynamically estimating the CPU consumption of a multi-tenancy Web application in a

收稿日期:2011-06-14;最终修改稿收到日期:2011-11-01.本课题得到国家自然科学基金(61100068,61173003)、国家"九七三"重点基础研究发展规划项目基金(2009CB320704)、国家"八六三"高技术研究发展计划项目基金(2011AA040504)、国家科技重大专项(2010ZX01045-001-010-4)和武汉大学软件工程国家重点实验室开放基金项目资助.**王 伟**,男 1982 年生,博士,助理研究员,主要研究方向为网络分布计算、资源管理等. E-mail: wangwei@otcaix.iscas.ac.cn. **黄 翔**,男,1982 年生,博士研究生,主要研究方向为网络分布计算、性能工程等. **张文博**,男,1976 年生,博士,副研究员,主要研究方向为网络分布计算、软件工程等. **魏 峻**,男,1970 年生,博士,研究员,博士生导师,主要研究领域为网络分布计算、软件工程等. **钟 华**,男,1971 年生,博士,研究员,博士生导师,主要研究领域为网络分布计算、软件工程等. **黄 涛**,男,1965 年生,博士,研究员,博士生导师,主要研究领域为网络分布计算、软件工程等.

shared Java middleware server, and we discuss the challenges involved in this approach. We investigate factors that impact the efficiency and accuracy of the approach in estimating time-varying states via two case studies. Experimental results show that, even under continuously changing workload conditions, estimation results are in agreement with the corresponding measurements with acceptable estimation errors, especially with appropriately tuned filter settings taken into account. Our experiments also demonstrate the utility of our approach in identifying the aggressive tenants and in avoiding shared middleware server CPU overloading.

Keywords　CPU consumption estimation; performance isolation; Kalman filter; multi-tenancy

1　引　言

多租户(multi-tenancy)让来自不同组织(租户)的并发用户共享使用同一基础设施资源,有利于成本降低和收益提高,是云计算的关键特征之一[1-2]. 如图 1 所示,目前存在 3 种典型的多租户部署方式:基于虚拟机(VM)的多租户部署、基于进程的多租户部署以及基于共享中间件的多租户部署. 基于 VM 的部署方式为每个租户配置一个 VM 实例,如 Amazon 的 EC2① 解决方案. 基于进程的部署方式为每个租户配置一个中间件服务器,如 Google 的 App Engine② 解决方案. 基于共享中间件(shared middleware)的部署方式通过单一中间件服务器为多个租户提供服务,如 Intalio③ 和 Salesforce④ 的解决方案. 共享中间件是一种高层次的多租户部署方式,与其它方式相比,资源共享程度更高,性能开销更低,因此租户的扩展性更强. 例如,在同样硬件环境下,使用基于 VM 的部署方式,1 个物理 CPU 最多支持 3 个虚拟 CPU(根据 VMWare 的配置建议),即支持 3 个租户,而基于共享中间件的部署方式则可以支持数十、甚至数百个租户[3]. 目前,共享中间件已经成为一种较为普遍的云计算部署方式,但在租户隔离方面(如性能、安全)仍存在诸多技术挑战. 本文主要关注性能隔离(performance isolation)相关技术研究.

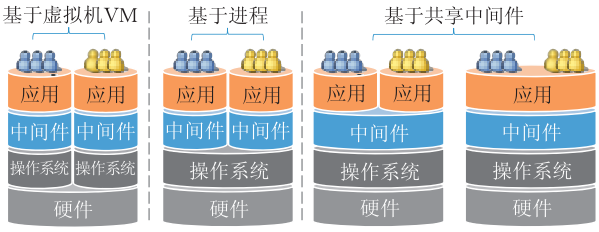


图 1　3 种典型的多租户部署方式

租户难以完全信赖,可能存在恶意侵占资源的行为,称之为侵占型租户(aggressive tenant). 即使同一组织内部、相互信赖的多个租户,仍有可能由于误操作或过载而导致个别租户侵占大量资源,影响其它租户性能. 性能隔离即防止租户间出现由于资源侵占而产生的性能干扰(performance interference). 对于电子商务、在线支付等性能关键(performance critical)服务,性能隔离至关重要,否则将导致客户损失和收益损失.

在共享中间件中实现性能隔离的关键是控制租户对 CPU、内存等资源的使用,其前提条件是在线获取租户的资源使用情况[4]. Java 服务器是最为广泛使用的中间件服务器之一,但是由于 Java 语言以及 Java 虚拟机(JVM)缺乏资源度量机制,在共享 Java 服务器中进行资源在线度量仍存在技术挑战,尤其是对 CPU 资源的在线度量[5]. 已有工作包括本地代码库(native code libraries)、程序转换(program transformations)等方法. 基于本地代码的方法需要对应用程序的源代码或二进制文件进行探针注入(probe instrumentation). 但是,在共享中间件环境下,中间件平台的提供者可能无权修改租户应用. 并且,方法需要调用操作系统级的本地代码,本质上是一种基于采样的度量方法,精确度依赖于采样时间戳(sampling timestamp)的精密程度. 但对于事务型应用(transactional application),每次请求处理所使用的 CPU 时间极短,对时间戳的精密度要求很高,甚至需要修改操作系统,因此存在系统兼容性问题. 基于程序转换的方法将程序字节码流量转换为 CPU 资源的使用量[5],方法不依赖于操作系统,但会产生较大性能开销(大于 30%)[6]. 对于在线系统而言,工业界可容忍的开销仅为 5%.

最近,一些研究工作利用多元回归分析方法估算资源使用量[7],在系统兼容性和性能开销方面具

① Amazon EC2, URL. https://aws.amazon.com/ec2/
② Google App Engine, URL. https://appengine.google.com/
③ Intalio, URL. http://www.intalio.com/
④ Salesforce, URL. http://www.salesforce.com/

有优势,但已有的研究工作表明此类方法难以对动态 Web 系统的时变资源状态进行有效评估,易产生较大误差[8],并且,方法的精确度依赖于长时间、高质量的观测样本作为输入,容易受到由于资源竞争造成的异常点(outliers)的影响.

在本文中,我们针对 Java 中间件服务器,提出一种基于 Kalman 滤波[9]的多租户 Web 应用 CPU 资源动态评估方法.方法通过近似最优的方式利用可观测值估算不可观测值,并且可随着新的观测值的到来更新之前的估算值,更适用于时变资源状态的在线评估.文中基于该方法设计实现了一个租户资源评估引擎①.通过两个实验案例分析方法的评估效果、影响因素和技术难点:

(1)电子商务应用测试基准 TPC-W②;

(2)在线软件开发环境 Trustie Forge③.

在第 1 个应用案例中,我们针对持续变化的负载环境,设计了一系列实验,分析滤波参数设置对于资源评估精确度的影响,并与回归分析方法进行比较.在第 2 个应用案例中,我们在支撑多个软件开发团队的 Forge 生产环境中,基于本文方法设计侵占型租户检测和过载保护策略,对方法的应用效果进行了验证.本文贡献总结如下:

(1)我们提出了基于 Kalman 滤波的多租户 Web 应用 CPU 资源动态评估方法.实验结果表明,通过设置适当的参数,方法可动态适应持续变化的负载环境,并且与直接度量方法相比,具有可接受的评估误差.

(2)在实验案例中,我们对本文方法的应用效果进行经验式分析.实验结果表明,方法可以有效检测出侵占型租户,并避免共享中间服务器 CPU 过载.

(3)方法利用生产环境中常用的监测数据进行资源评估,因此不会产生性能开销.并且,方法具有非侵入性和操作系统独立性.

本文第 2 节对 CPU 资源动态评估问题进行形式化描述,并讨论技术挑战;第 3 节给出基于 Kalman 滤波的动态评估方法;第 4 节针对 TPC-W 电子商务应用验证方法的有效性;第 5 节针对在线软件开发环境验证方法的应用效果;第 6 节讨论本文方法的不足与未来工作;第 7 节比较相关研究工作;最后是全文结论.

2 问题描述

本节首先针对多租户 Web 应用 CPU 资源动态评估问题进行形式化描述,接着讨论利用回归分析方法追踪时变资源状态的技术挑战.

2.1 问题形式化

对多租户 Web 应用的 CPU 资源进行动态评估的前提条件是在线收集中间件服务器的日志信息,包括租户的吞吐量以及中间件服务器的 CPU 利用率.上述运行时信息在固定的时间间隔被监测记录,这种时间间隔称为监测窗口.为方便后文的讨论,对于一个宿主了 N 个租户的中间件服务器,我们给出以下记号:

T 表示监测窗口长度;

N_i 表示第 i 个租户($1\leq i\leq N$)在监测窗口内完成的事务数;

U_{CPU} 表示中间件服务器在监测窗口内的 CPU 平均利用率;

S_i 表示第 i 个租户($1\leq i\leq N$)所有事务的平均服务时间(即所有事务的 CPU 平均占用时间).

根据效用法则(Utilization Law)[10],资源利用率等于吞吐量乘以服务时间,我们可以得到等式:

$$U_{\mathrm{CPU}}T=\sum_i N_i S_i \tag{1a}$$

由于难以精确度量服务时间 S_i,我们使用 C_i 表示其近似值,进而得到资源利用率近似值 U'_{CPU} 的计算等式:

$$U'_{\mathrm{CPU}}=\frac{\sum_i N_i C_i}{T} \tag{1b}$$

可以利用统计分析方法求解 C_i.对于此类间接逼近问题的求解,$U_{\mathrm{CPU},k}$ 和 $U'_{\mathrm{CPU},k}$ 的误差是衡量精确度的典型指标,其中 k 表示监测窗口的标示号.在本文中,我们的目标是如何降低真实服务时间 S_i 与近似服务时间 C_i 的误差.

2.2 追踪服务时间变化

一些工作利用多元回归分析方法求解 C_i.但是动态 Web 系统的资源状态具有时变性,严重影响方法的精确度和适应性.动态 Web 系统包含用于完成各种业务逻辑的多种事务,负载组成(workload mix)则表示系统当前负载中包含的各种事务的组成比例.在开放环境中,租户的负载组成不断变化,导致租户所有事务的平均服务时间也不断变化.例如,工业级的电子商务测试基准 TPC-W 定义了 14 种事务

① TRE4J, URL. http://www.trustie.net/projects/project/show/TREforJ

② TPC-W Benchmark: URL http://www.tpc.org

③ Trustie forge, URL: http://www.trustie.net/

操作，包括网上书店的浏览、查询以及订单等事务操作. 这 14 种事务可被分为浏览型和订单型两类，并根据不同的组成比例，定义了如表 1 所示的 3 种负载组成模式(浏览、购物、订单). 下面通过一个简单实验说明负载组成与事务服务时间的关系. 针对 3 种负载组成模式，我们分别在 10～600 的单租户并发规模下进行测试，并统计事务吞吐率和 CPU 平均利用率，随后利用等式(1a)计算租户的事务平均服务时间. 实验结果如图 2 所示，对于特定的负载组成模式，租户的事务平均服务时间在不同并发规模下基本相同，但是不同负载组成模式的平均服务时间在同一并发规模下却存在着较大差异.

表 1　TPC-W 基准测试中的事务类型和组成模式

事务类型	浏览模式	购物模式	订单模式
浏览相关	95.00%	80.00%	50.00%
订单相关	5.00%	20.00%	50.00%

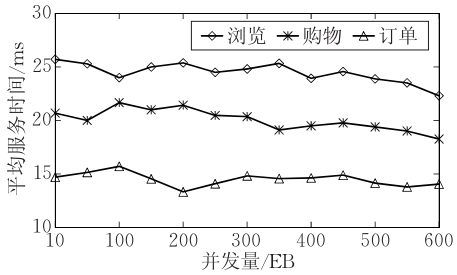


图 2　3 种负载组成模式下的事务平均服务时间

另外，回归分析方法的精确度还依赖于长时间、高质量的样本作为输入，容易受到由于资源竞争造成的异常点的影响. 下面通过一个简单实验说明事务吞吐率与 CPU 利用率的关系. 在单个租户的 3 种负载组成模式下，租户的访问并发量随机变化，在一组长度为 1 min 的监测窗口中，我们将吞吐率按照 CPU 利用率进行归类统计，得到如图 3 所示的吞吐率剖面(profile). 可以看出，3 种负载组成模式下的吞吐率剖面存在差异，这与我们的预计相同. 但是，我们也发现曲线中存在较多的异常点. 这些异常点通常是由于资源竞争而导致在异常 CPU 利用率情况下出现较小的吞吐率而产生的. 由于回归分析方法需要最小化所有样本点的绝对误差，当存在较多异常样本点时，方法精确度将受到严重影响.

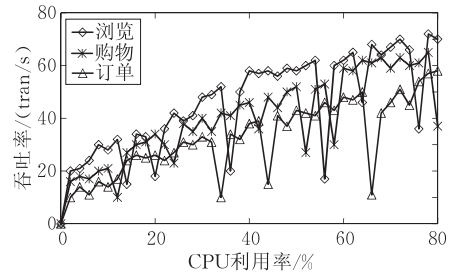


图 3　3 种负载组成模式下的吞吐率剖面

一种改进的方法是对最近的监测样本进行异常点过滤和回归分析. 但是，对于存在 N 个租户的共享中间件服务器而言，通常需要 $N+1$ 个监测窗口的时间来收集多元回归的样本. 考虑到存在异常样本，则需要更长的采样时间，如果在此期间负载组成出现较大变化，则会影响回归分析的结果.

综上分析，回归分析方法难以满足共享中间件服务器环境下、具有时变特征的多租户 CPU 资源的评估需求. 我们由此需要研究新的方法用于追踪服务时间的变化.

3　基于 Kalman 滤波的 CPU 资源动态评估

3.1　Kalman 滤波器

Kalman 滤波[9]在自动控制和辅助导航领域得到了广泛的使用和研究，其最大的特点是可以用一种近似最优的方式基于可观测值估算不可观测值，并且可随着新的观测值的到来更新之前的估算值，因此更适用于时变资源状态的在线评估.

Kalman 滤波提供了一个在离散时间点，估算不可观测状态 $\boldsymbol{x}$ 的通用方法. 第 k 时刻状态 $\boldsymbol{x}_k$ 可以定义为一个线性随机差分方程：

$$\boldsymbol{x}_k=\boldsymbol{A}\boldsymbol{x}_{k-1}+\boldsymbol{w}_{k-1} \tag{2a}$$

第 k 时刻测量值 z_k 定义为

$$z_k=\boldsymbol{H}_k\boldsymbol{x}_k+v_k \tag{2b}$$

其中，$\boldsymbol{A}$ 是从 $k-1$ 时刻到 k 时刻状态转换矩阵，$\boldsymbol{w}_{k-1}$ 为过程误差，其协方差矩阵为 $\boldsymbol{Q}_{k-1}$. $\boldsymbol{H}_k$ 是 $\boldsymbol{x}_k$ 到 z_k 的转换矩阵，v_k 是观测误差，其协方差矩阵为 $\boldsymbol{R}_k$.

3.2　基于 Kalman 滤波的多租户 CPU 资源动态评估

图 4 显示了基于 Kalman 滤波的多租户 Web 应用资源评估引擎的逻辑架构. 我们首先将不可观测状态 $\boldsymbol{x}$ 建模为包含 N 个租户的事务平均服务时间的 N 维向量 $\boldsymbol{x}_k=(C_1^k,C_2^k,\cdots,C_N^k)$，表示 k 时刻各租户的事务平均服务时间. 然后，根据等式(1b)对观测得到的总 CPU 利用率 $\boldsymbol{z}_k$ 进行建模，得到

$$z_k=\frac{\sum_i N_i^k C_i^k}{T}+v_k \tag{3}$$

其中，N_i^k 表示监测得到的租户 i 的吞吐率，$\boldsymbol{H}_k$ 则定义为 $\left(\frac{N_1^k}{T},\frac{N_2^k}{T},\cdots,\frac{N_N^k}{T}\right)$.

Kalman 滤波器算法在每个监测窗口的结束时

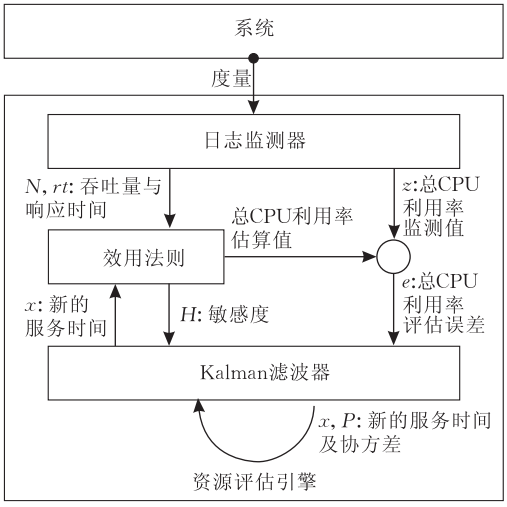


图 4 基于 Kalman 滤波的资源评估引擎的逻辑架构

进行服务时间的迭代评估,初始值包括状态初始值 $\hat{\boldsymbol{x}}_0$ 以及初始的误差协方差矩阵 $\boldsymbol{P}_0$. 迭代过程如下:

(1) 向前推算 $\boldsymbol{x}$ 的状态:

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1} \tag{4a}$$

(2) 向前推算状态先验估计误差的协方差矩阵 $\boldsymbol{P}_k^-$:

$$\boldsymbol{P}_k^- = \boldsymbol{A}\boldsymbol{P}_{k-1}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{Q}_k \tag{4b}$$

(3) 计算 Kalman 增益 $\boldsymbol{K}_k$:

$$\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}_k^{\mathrm{T}}(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^{\mathrm{T}} + \boldsymbol{R}_k)^{-1} \tag{4c}$$

(4) 由观测变量 z_k 更新 $\boldsymbol{x}$ 的状态:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(z_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-) \tag{4d}$$

(5) 更新状态后验估计误差的协方差矩阵 $\boldsymbol{P}_k$:

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_k^- \tag{4e}$$

迭代过程中,第 4 步修正 $\boldsymbol{x}$ 的状态是估算值更新的关键,该等式可以简化为 $\boldsymbol{x}_{\mathrm{new}} = \boldsymbol{x}_{\mathrm{old}} + \boldsymbol{K}\cdot\boldsymbol{e}$ 的形式,即 Kalman 增益 $\boldsymbol{K}$ 可以看作修正 $\boldsymbol{x}$ 的权重矩阵,利用误差 $\boldsymbol{e}$ 和相应的权重修正 $\boldsymbol{x}_{\mathrm{old}}$ 的数据. 同时,在第 4 步计算中,还需要考虑设置每个租户的事务平均服务时间的估值范围,即非负且小于某个上界:

$$0 \leq C_i^k < u_k,$$

其中,u_k 表示估值上界. 在本文中,我们设置状态 $\boldsymbol{x}$ 的最大值小于这一上界:

$$(0, (\mu u_k + (1-\mu)\boldsymbol{x}_{k-1})),$$

其中设置 $\mu = 0.9$,且 $u_k = U_{\mathrm{CPU}}^k T / N_i^k$. 进而第 4 步的计算修正为

$$\hat{\boldsymbol{x}}_k^{\mathrm{trial}} = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(z_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-) \tag{4f}$$

$$\hat{\boldsymbol{x}} = \min(\hat{\boldsymbol{x}}_k^{\mathrm{trial}}, (\mu u_k + (1-\mu)\boldsymbol{x}_{k-1})) \tag{4g}$$

3.3 滤波器参数影响

滤波器的参数设置是影响方法精确度和适应性关键. 参数设置包括 $\boldsymbol{Q}$ 和 $\boldsymbol{R}$ 矩阵、初始状态向量及误差协方差以及监测窗口长度等.

3.3.1 设置 $\boldsymbol{Q}$ 和 $\boldsymbol{R}$

$\boldsymbol{Q}$ 和 $\boldsymbol{R}$ 矩阵影响 Kalman 增益 $\boldsymbol{K}$,因此影响滤波器对新观测数据的反应. 实际中,$\boldsymbol{Q}$ 矩阵是不可知的,但如果 $\boldsymbol{Q}$ 设置过大,则会增大迭代的误差协方差矩阵 $\boldsymbol{P}$,进而增大 Kalman 增益 $\boldsymbol{K}$,导致滤波器对评估误差反应过大,最终造成评估结果抖动;如果 $\boldsymbol{Q}$ 设置过小,则会降低 Kalman 增益 $\boldsymbol{K}$,最终降低滤波器对评估误差的响应性,难以追踪服务时间变化. 在本文中,我们的应对策略是将 $\boldsymbol{Q}_k$ 设置为对角矩阵,且对角线元素利用前三次迭代产生的 $\boldsymbol{x}$ 值动态计算获得,即 $\boldsymbol{Q}_k = \mathrm{diag}(\xi_1, \xi_2, \cdots, \xi_N)$,且 $\xi_i = \left(C_{i,k-1} - \frac{C_{i,k-2} + C_{i,k-3}}{2}\right)^2$. $\boldsymbol{R}$ 矩阵是总 CPU 利用率的测量误差的协方差矩阵,因此可以在服务器空载状态下统计获得. 4.2.3 节的实验结果显示 $\boldsymbol{Q}$ 和 $\boldsymbol{R}$ 矩阵设置的正确与否对评估精确度存在较大影响.

3.3.2 设置初始状态向量及误差协方差

一些研究工作认为初始状态向量及误差协方差的设置对评估结果影响较小. 但是,我们发现设置适当的初始状态向量,并利用转换矩阵 $\boldsymbol{A}$ 对每次迭代的输入状态向量进行动态修正,可以提高方法收敛性(见 4.2 节). 我们根据排队论,在各监测窗口内,对租户的事务平均服务时间进行估算[11],设置初值状态向量和 $\boldsymbol{A}$ 矩阵的取值,等式(4a)修正为

$$\hat{\boldsymbol{x}}_k^- = \begin{cases} \hat{\boldsymbol{x}}_{k-1}, & k=1 \\ \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1}, & k>1 \end{cases} \tag{5}$$

当 $k=1$ 时,初始状态向量 $\hat{\boldsymbol{x}}_0$ 的各参数设置为 $C'_{i,1} = rt_{i,1}(1-U_{\mathrm{CPU},1})$,其中 $rt_{i,1}$ 表示在第一个监测窗口内租户 i 的事务平均响应时间. 另外,因为各租户的事务平均服务时间是独立的,所以设置初始状态向量的误差协方差 $\boldsymbol{P}_0$ 为一个对角矩阵:$\boldsymbol{P}_0 = \mathrm{diag}((C_1^0)^2, (C_2^0)^2, \cdots, (C_N^0)^2)$,即取状态向量各参数初始值的平方. 当 $k>1$ 时,设置转换矩阵 $\boldsymbol{A}_k = (C'_{1,k}/C'_{1,k-1}, C'_{2,k}/C'_{2,k-1}, \cdots, C'_{i,k}/C'_{i,k-1}, \cdots, C'_{N,k}/C'_{N,k-1})$,其中 $C'_{i,k} = rt_{i,k}(1-U_{\mathrm{CPU},k})$,表示在第 k 个监测窗口第 i 个租户的事务平均服务时间的预测值. 利用转换矩阵动态修正滤波器的输入状态,本质上是增加了观测参数,有利于提高方法收敛性[12].

3.3.3 设置监测窗口长度

文中滤波器的迭代步长即为监测窗口长度. 在监测窗口中,方法需要获得各租户的吞吐量 N_i^k、平均响应时间 $rt_{i,k}$ 以及总 CPU 利用率 z_k,这些数据是在监测窗口内采样并计算平均值获得. 因此,较长的监测窗口内会提高监测数据的精确度,但也会降低

方法对服务时间变化的适应性. 相反，较短的监测窗口可以快速追踪服务时间的变化，但代价是增大了监测数据误差，最终影响方法的评估精确度. 在 4.2.4 节中，我们将在持续变化的负载环境下，分析监测窗口长度对方法精确度的影响.

3.4　收敛性与复杂度

上述 Kalman 滤波迭代计算的收敛条件是，等式(3)是线性的，且满足可识别条件(identifiability condition)[12]. 可识别条件要求观测数据至少与状态向量 $\boldsymbol{x}$ 的参数数量相等，且各观测数据间是不相关的(noncorrelated). 在本文中，易知等式(3)是线性的，同时，方法对每个租户的事务吞吐率进行监测，观测数据等于状态向量 $\boldsymbol{x}$ 中的参数数量，并且，各租户的事务吞吐率之间是独立的，因此满足 Kalman 滤波的收敛条件.

本方法一次迭代的复杂度为 $O(N^3)$，N 为宿主在同一中间件服务器上的租户数量. 因为观测值 z 定义为 CPU 总利用率，而 $\boldsymbol{H}$ 和 $\boldsymbol{K}$ 是向量，所以式(4c)中求逆退化对一个数求倒数. 剩下复杂度最高的计算是式(4e)所对应的步骤，其复杂度为 $O(N^3)$. 因此方法总的复杂度为 $O(N^3)$.

4　实验案例 1：TPC-W 电子商务应用

本节针对 TPC-W 电子商务应用设计了一系列实验，分析滤波参数设置对资源评估精确度的影响，并且，在 4.2.5 节与文献[7]中使用的回归分析方法进行了比较.

4.1　实验方法与环境

选用 TPC-W 基准测试作为实验基础. TPC-W 是一个广泛使用的电子商务测试基准. 实验选用了中国科学院软件研究所软件工程技术中心研发的符合 TPC-W 基准规范的 Bench4Q 测试套件①，可模拟复杂、可控的多租户访问负载.

如图 5，实验环境是一个典型的两层架构系统，包括：1 个 Tomcat 中间件服务器和 1 个 DB2 数据库服务器；3 个客户端模拟标示为“租户 1”、“租户 2”、“租户 3”的 3 个租户负载，访问部署在 Tomcat 上的电子商务应用；资源评估引擎收集 Tomcat 的运行时日志信息. 表 2 给出了软硬件环境的具体信息. 我们将 Tomcat 分别运行在 Windows Server(2003)和 Linux(2.6.18)两个操作系统环境下进行测试，评估结果具有相似特征，表示方法具有操作系统独立性. 受篇幅所限，下面我们仅给出 Windows 环境下的测试结果.

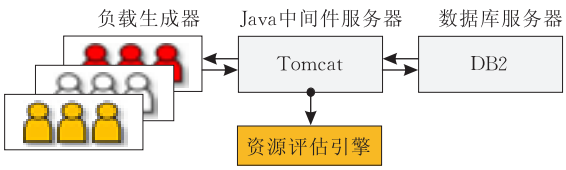


图 5　TPC-W 实验架构

表 2　软硬件环境

服务器	处理器	内存/GB	数量
租户负载生成器	Pentium IV/1.8GHz	2	3
Java 中间件服务器(Tomcat6.0)	Pentium IV/2.8GHz	2	1
数据库服务器(DB2-V9.5)	Pentium IV/3.8GHz	2	1
资源评估引擎	Pentium IV/1.8GHz	2	1

实验中方法评估误差的计算方式如下：在每次测试中，每隔 30 s 计算租户的事务平均服务时间的相对误差绝对值 $e_i=\dfrac{|C_i^{\text{new}}-S_i^{\text{new}}|}{S_i^{\text{new}}}$，其中，$C_i^{\text{new}}$ 表示最新的评估结果，S_i^{new} 表示通过 Windows 系统提供的事件追踪机制得到的服务时间直接度量值②，接着计算每次测试的平均相对误差绝对值(Mean Absolute Relative Error，MARE).

图 6 表示所生成的 3 个租户负载. X 轴表示时间，Y 轴表示并发用户数，可以看到，每个租户的负载大小随时间持续变化. 负载组成分为固定和变化两种，前者表示所有租户负载起始于相同的负载组成，且测试过程中保持不变，后者则表示租户的负载组成以一定长度的 T_{change} 为变化周期，交替生成 3 种负载组成模式.

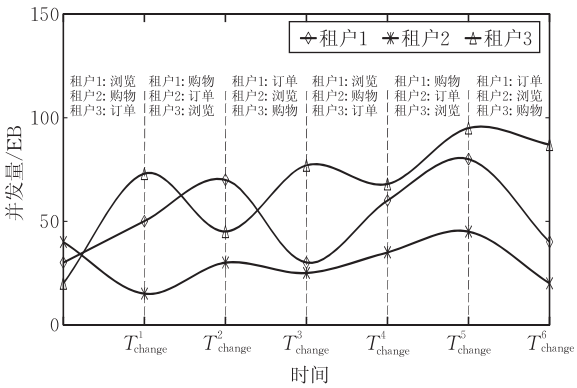


图 6　租户负载变化

4.2　实验结果及分析

4.2.1　追踪服务时间变化

在该实验中，我们分析本文方法在负载组成变化情况下，对租户的事务平均服务时间的评估效果.

① Bench4Q: URL http://forge.ow2.org/projects/jaspte
② Microsoft. Event tracing for windows: URL http://www.microsoft.com/whdc/devtools/tools/EventTracing.mspx

如图 6 所示，每个租户的事务平均服务时间随着浏览、购物以及订单 3 种负载组成模式的交替而变化. 实验中，设置负载组成的变化周期 $T_{\text{change}}=1800$ s，评估方法的监测窗口长度设置为 $T=T_{\text{change}}/10=180$ s(远小于 T_{change}). 滤波器初始状态向量及误差协方差矩阵根据 3.3 节提供的方法进行设置，$Q_0=\text{diag}(1,1,1)$.

图 7 显示了方法追踪服务时间变化的实验数据. X 轴表示时间，Y 轴表示租户的事务平均服务时间. 图中曲线“租户 N 评估值”表示实验中每个监测窗口内租户 N 的事务平均服务时间评估值的变化，曲线“租户 N 监测值”表示相应的直接度量值的变化. 实验数据显示，与直接度量值相比，当负载组成变化时，方法需要经过最多 4 个监测窗口的延迟，然后评估值会收敛到直接度量值附近，反映出良好的收敛效果和对服务时间变化的适应性.

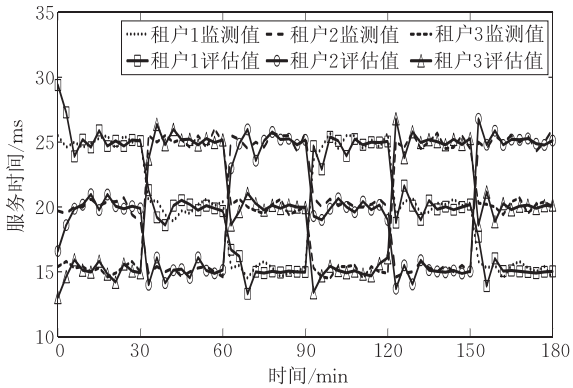


图 7 追踪服务时间变化

4.2.2 Q 和 R 的设置与调整

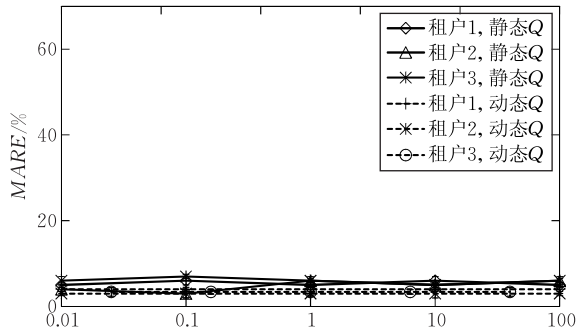
在该实验中，分析 Q 和 R 矩阵设置对评估结果的影响. 首先，将 Q 矩阵根据 α 因子增大或减小，α 的取值范围是 0.01 到 100. 图 8(a)、(b)分别显示在负载组成固定(购物模式)和变化环境下的方法精确度. X 轴表示不同 Q 矩阵设置，Y 轴表示 $MARE$ 及其标准偏差. 对比实验数据，可以观察到：

(1) 如图 8(a)所示，在负载组成固定时，曲线“静态 Q” 表示在 α 因子变化得到的不同静态 Q 矩阵情况下，3 个租户的事务平均服务时间的评估误差；曲线“动态 Q”表示采用 3.3 节中动态 Q 矩阵设置时的评估误差. 实验数据显示，评估误差均小于 7%.

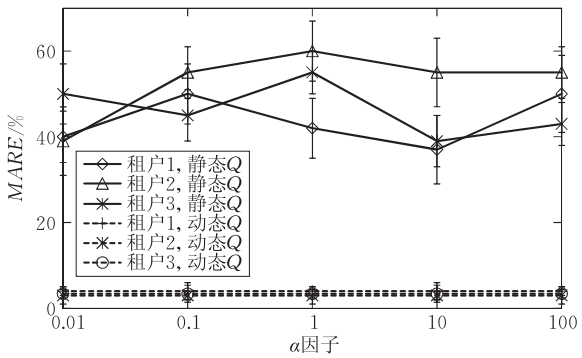
(2) 如图 8(b)所示，在负载组成变化时，静态 Q 矩阵设置的评估误差大于 40%，而动态 Q 矩阵设置的评估误差则小于 5%.

上述观察表明，当负载组成固定时，动态更新 Q 矩阵并不会影响方法精确度，反之，则需要在方法迭代时根据近期追踪数据对 Q 矩阵进行更新.

接下来分析 R 矩阵设置对方法精确度的影响. R 矩阵根据 β 因子增大或减小，β 的取值范围是 0.01 到 100. 图 9(a)、(b)分别显示了方法在负载组

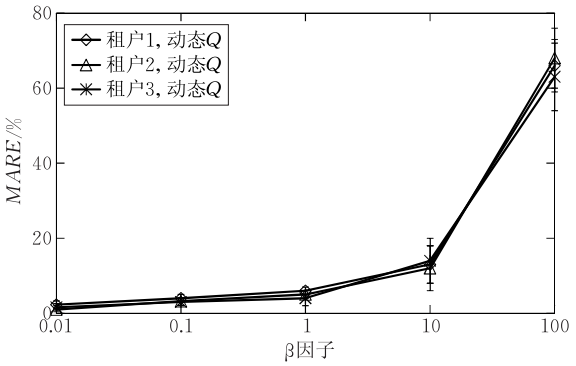


(a) 负载组成固定

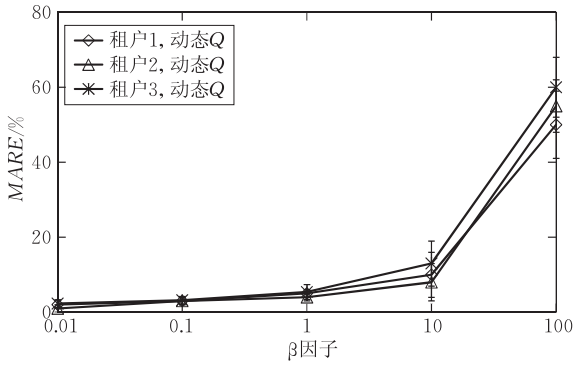


(b) 负载组成变化

图 8 不同 Q 矩阵设置下的 $MARE$



(a) 负载组成固定



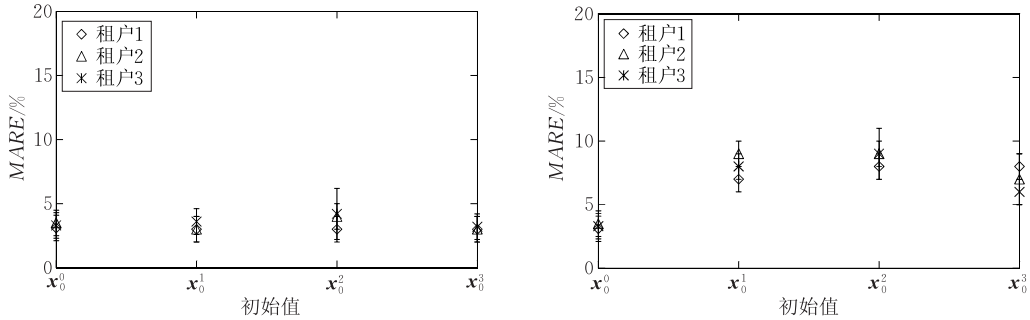
(b) 负载组成变化

图 9 不同 R 矩阵设置下的 $MARE$

成固定(购物模式)和变化环境下，不同 $\boldsymbol{R}$ 矩阵设置的方法精确度. 实验数据显示，在上述环境下，方法精确度均受到 $\boldsymbol{R}$ 矩阵设置的影响，随着 $\boldsymbol{R}$ 矩阵值增大时，方法对观测数据的适应性降低，评估误差持续增长.

4.2.3　状态向量的初始设置与调整

在该实验中，分析状态向量的初始设置以及动态调整对评估结果的影响. 图 10(a)、(b)分别显示了在负载组成(购物模式)固定和变化环境下的方法精确度. X 轴表示 4 个不同 $\hat{\boldsymbol{x}}_0$ 设置：$\hat{\boldsymbol{x}}_0^0=(C'_{1,1},C'_{2,1},C'_{3,1})$；$\hat{\boldsymbol{x}}_0^1=(C'_{1,1},C'_{1,1},C'_{1,1})$；$\hat{\boldsymbol{x}}_0^2=(C'_{2,1},C'_{2,1},C'_{2,1})$；$\hat{\boldsymbol{x}}_0^3=(C'_{2,3},C'_{2,3},C'_{2,3})$，并且对于后 3 种设置不进行状态向量的动态调整. Y 轴表示 $MARE$ 及其标准偏差. 实验结果显示，负载组成固定时，评估误差小于 5%，负载组成变化时，评估误差也小于 10%. 上述观察显示状态向量的初始设置及动态调整对评估结果的影响很小. 但是，产生这一现象的原因可能是由于各租户的事务平均服务时间存在的差异并不显著(如图 2 所示，浏览模式大约为 25 ms，购物模式大约为 20 ms，订单模式大约为 15 ms)，因此，状态向量的初始设置及动态调整对方法的收敛效果影响较小.



(a) 负载组成固定　　(b) 负载组成变化

图 10　不同 $\hat{\boldsymbol{x}}_0$ 设置下的 $MARE$

为验证这一推断，我们修改了 TPC-W 电子商务应用，在其中两个租户的事务中注入 CPU 密集型操作. 图 11 显示了修改后，在负载组成变化环境下的实验数据. 可以看到，在对状态向量设置适当的初始值并进行动态调整的情况下，评估误差小于 5%，而其它 3 种情况的评估误差则大于 50%. 图 11 还显示了状态向量各参数的初始设置值与直接度量值的余弦相似度(cosine similarity). 余弦相似度是一种用于比较两个 n 维向量的相似度的度量，相似度的值域为 0 到 1. 相似度接近 1 则表示两个向量越近似. 如图 11 所示，可以发现 $\hat{\boldsymbol{x}}_0^0$ 的相似度明显高于其它 3 种设置. 上述观察表明，通过设置适当的初始状态向量，并利用转换矩阵动态修正状态向量，可以提高滤波器的收敛速度，进而提高了方法精确度和适应性.

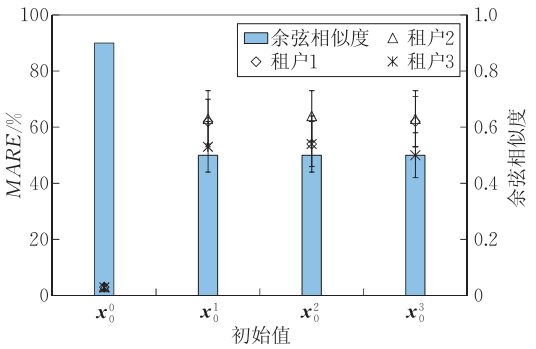


图 11　不同 $\hat{\boldsymbol{x}}_0$ 设置下的 $MARE$(修改后的 TPC-W 电子商务应用)

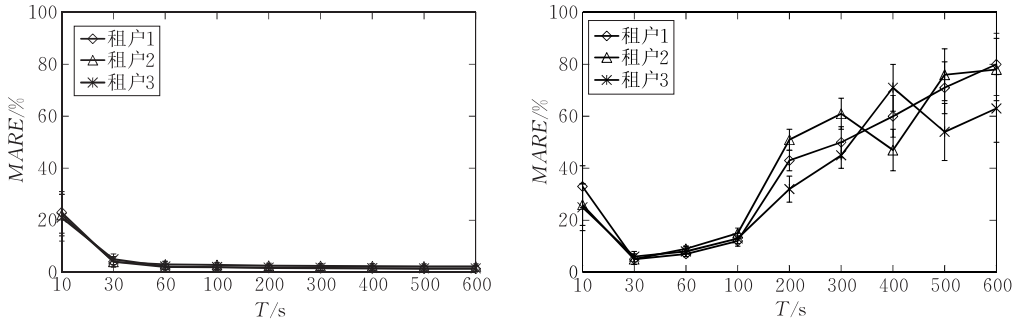
4.2.4　监测窗口长度的设置与调整

在前文实验中，我们设置负载组成的变化周期 $T_{\text{change}}=1800$ s，监测窗口长度则远小于 T_{change}，设置为 $T=300$ s. 在本节实验中，我们减小负载组成的变化周期为 $T_{\text{change}}=300$ s，分析不同监测窗口长度对评估结果的影响. 作为比较，我们还在负载组成(购物模式)固定环境下进行了相同的实验.

图 12(a)、(b)分别显示在负载组成固定和变化环境下的方法精确度. X 轴表示不同的监测窗口长度设置，$T=10$ s，30 s，60 s，100 s，200 s，300 s，400 s，500 s，600 s，Y 轴表示 $MARE$ 及其标准偏差. 对比实验数据，可以观察到：

(1) 在负载组成固定环境下，如图 12(a)所示，较长的监测窗口可以获得更高的精确度. 但是，当负载组成变化时，较长的监测窗口设置降低了方法适应性. 如图 12(b)所示，当 $T>100$ s 时，由于方法需要经过 1 至多步的收敛过程，较长的监测窗口将显著延长收敛时间，因此评估误差逐渐增大.

(2) 如图 12(b)所示，较短的监测窗口并没有受异常点的影响，可以很好地追踪服务时间变化，并获得可接受的评估误差. 当 $T=30$ s 时，评估误差小于



(a) 负载组成固定 (b) 负载组成变化

图 12 不同 T 设置下的 $MARE$

5%.这可能是由于滤波器经过连续多次迭代后,消除了异常点的影响.但是,当监测窗口过小,无法获得足够精确的观测值时,评估误差逐渐增大.例如,当 $T=10$ s,评估误差超过了 20%.

上述观察说明,通过设置适当的监测窗口长度,我们的方法可以适应负载组成快速变化的环境,获得较为精确的评估结果.

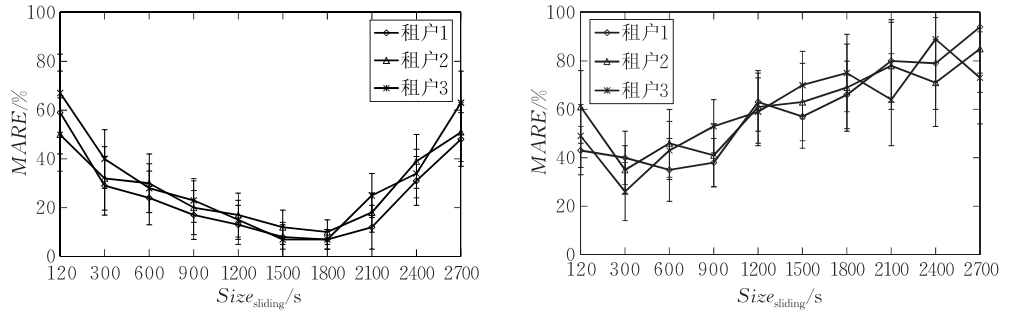
4.2.5 与回归分析方法比较

在该实验中,我们在负载组成变化环境下,利用文献[7]提出的回归分析方法对多租户 CPU 资源进行评估.方法使用滑动窗口进行数据采样,然后使用 MATLAB 提供的非负的最小二乘法进行回归分析,得到各租户的事务平均服务时间的评估值 C_i.该方法的目标是最小化误差 $\varepsilon=\sqrt{\sum_j (U'_{\mathrm{CPU},k}-U_{\mathrm{CPU},k})_j^2}$,并且 $C_i\geq 0$,其中 j 用于标示滑动窗口内的样本.通常,回归分析方法的采样周期对精确度影响较小[7],为了更好地追踪服务时间变化,我们使用一个较小的采样周期 $T=30$ s.然后,我们通过设置不同的滑动窗口大小 $Size_{\mathrm{sliding}}$ 来分析回归分析方法的评估效果.

图 13(a)、(b)分别显示了在负载组成慢速变化($T_{\mathrm{change}}=1800$ s)和快速变化($T_{\mathrm{change}}=300$ s)两种环境下的评估误差,滑动窗口大小取值范围从 120 s(小于快速变化周期)到 2700 s(大于慢速变化周期).对比实验数据,可以观察到:

(1) 负载组成慢速变化时,较大的滑动窗口可以获得更高的精确度.如图 13(a)所示,可以看到,在负载组成变化周期附近,评估效果最优.但是,当滑动窗口过大或过小时,评估误差迅速增大.

(2) 负载组成快速变化时,回归分析方法难以追踪服务时间变化.如图 13(b)所示,虽然可以在负载组成变化周期附近找到一个滑动窗口大小的最优取值,但评估误差已经超过了 30%.



(a) $T_{\mathrm{change}}=1800$ s (b) $T_{\mathrm{change}}=300$ s

图 13 不同负载组成变化周期和滑动窗口大小设置下的 $MARE$

上述观察表明,难以权衡回归分析方法的精确度和适应性.实验结果显示,负载组成变化周期较大时,如 $T_{\mathrm{change}}\gg Size_{\mathrm{sliding}}$,可以获得较好的评估效果,这表明回归分析方法的精确度受样本数量和质量的影响较大.当负载组成快速变化时,需要降低滑动窗口的大小.但是,受异常点的影响,较小的滑动窗口难以收集到足够多的回归分析样本来保证方法精确度.

5 实验案例 2:在线 Forge 应用

在该实验案例中,我们在支撑多个软件开发团队的 Forge 生产环境中验证方法的精确度,并基于本文方法设计侵占型租户检测和过载保护策略,对

方法的应用效果进行验证.

5.1　实验方法与环境

Trustie Forge 是在国家高技术研究发展计划(863)课题“可信的国家软件资源共享与协同生产环境”支持下,研发的一个基于互联网的软件协同开发环境.我们在本地部署了一个企业版 Trustie Forge 系统,支持了包括 12 个软件项目团队(租户)、上百人的软件项目管理和协同开发活动.如图 14 所示,Trustie Forge 系统是一个三层架构系统,包括部署 Trustie 门户的 Nigix 服务器、部署 Trustie Forge 系统的 Tomcat 中间件服务器以及若干数据库服务器、邮件服务器等.资源评估引擎收集 Tomcat 的运行时日志信息,所有系统运行在 Windows Server 2003.

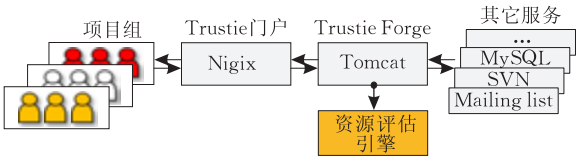


图 14　Trustie Forge 系统架构

5.2　实验结果及分析

5.2.1　租户 CPU 资源评估

通过对 Trustie Forge 系统进行分析,我们发现该系统包含两百多种事务,并可分类为项目浏览(browsing)和项目执行(processing)两类,前者主要访问缓存内容,后者需要动态生成富客户端页面,是 CPU 密集型的事务.图 15 显示了在一个典型的工作日中这两类事务组成比例的变化情况,由此可见,事务的平均服务时间也是动态变化的.图 16 显示了方法的评估结果,监测窗口大小设置为 $T=30$ s,图中曲线表示 12 个租户的评估误差的累积概率分布(Cumulative Distribution Function,CDF).如图 16 所示,12 个租户的 90% 以上的评估误差都小于 5%,并且资源评估引擎对 12 个租户的每次迭代计算耗时仅为 2 ms.

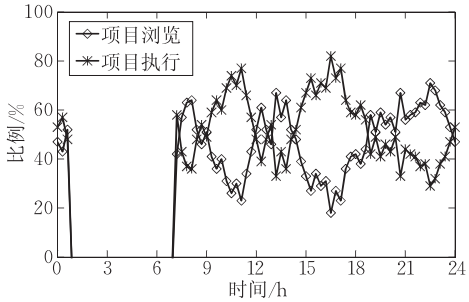


图 15　Trustie Forge 负载组成的变化

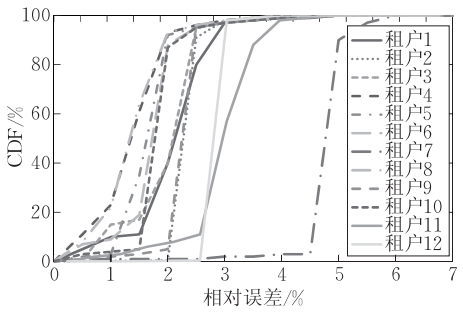


图 16　评估误差的 *CDF*

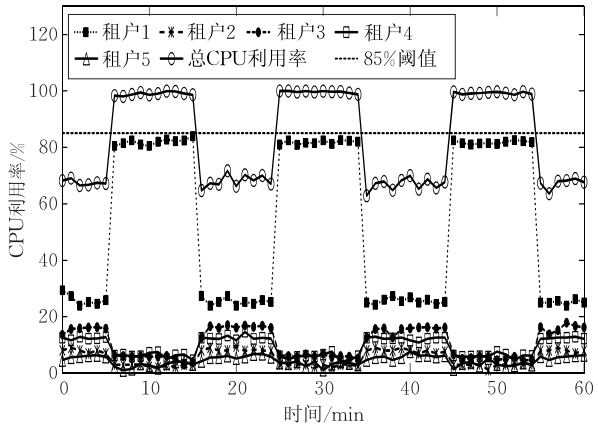
5.2.2　侵占型租户检测与 CPU 过载保护

Trustie Forge 采用 Eclipse RAP① 技术动态生成项目执行相关的富客户端页面(RAP 页面).在 Tomcat 中间件服务器中,RAP 页面的生成过程是 CPU 密集型的任务,当进行项目执行的租户负载增高时,会导致中间件服务器的 CPU 过载.在该实验中,我们通过常用的并发控制(concurrency control)技术来避免 CPU 过载.对于多租户应用,限制所有租户的并发量对于非侵占型的租户而言是不公平的,因此需要根据各租户的 CPU 资源状态进行细粒度的并发控制.在已有工作中,我们曾提出一种细粒度的中间件并发控制框架(QoS-enabled WorkManager,QWM)[13].在该实验中,我们基于 QWM 框架和本文提出的 CPU 资源评估方法,设计实现一种面向多租户的 CPU 过载保护策略,其基本思路如下:利用 CPU 资源评估方法,在线监测各租户的 CPU 资源状态以及 Tomcat 中间件服务器的总 CPU 利用率;当总 CPU 利用率大于预先设定的过载阈值时,则根据各租户的 CPU 资源状态,检测占用资源最多的租户,标示为侵占型租户,然后,依据排队论模型,求解对侵占型租户的并发量限制,并利用 QWM 框架进行并发控制;当总 CPU 利用率小于预设过载阈值时,并且 QWM 框架中存在由于前期并发限制而出现的等待队列时,则逐步降低并发限制,直到总 CPU 利用率接近过载阈值.

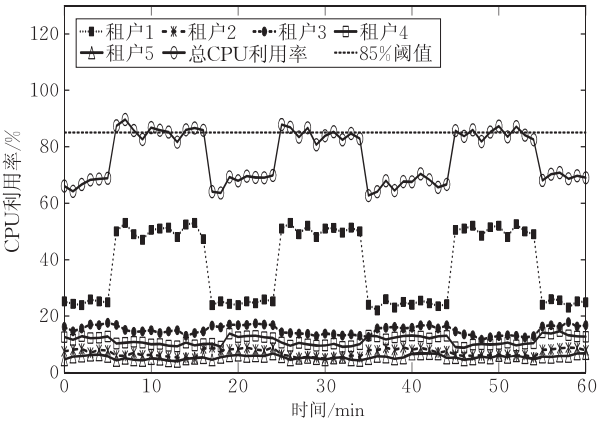
为了验证上述策略的效果,我们建立了一个包含 5 个租户的 Trustie Forge 实验环境,并通过 Bench4Q 录制并模拟租户负载.在该实验中,租户 1(Tenant 1)被设计为侵占型租户,进行更频繁的项目执行操作,并且负载大小进行高低交替.CPU 过载阈值根据已有经验被设置为 85%[14].图 17(a)、(b)分别显示了未使用过载保护策略和使用过载保护策略时各租户的 CPU 利用率以及总 CPU 利用

① Eclipse RAP, URL: http://www.eclipse.org/rap/

率的变化情况.在图 17(a)中,当租户 1 出现高负载时,可以观测到其它租户的 CPU 利用率出现明显的下降,并且总 CPU 利用率近 100%.上述现象表明,未使用过载保护策略时,租户 1 在高负载时侵占了其它租户的资源,并严重威胁了系统可用性.在图 17(b)中,由于使用了过载保护策略,租户 1 在高负载时被检测为侵占型租户,受到了并发限制,对其它租户的影响明显降低,并且总 CPU 利用保持在 85%以下.由此可见,基于本文的多租户 CPU 资源评估方法,上述过载保护策略可以有效检测侵占型租户,并通过对侵占型租户的并发限制,将总 CPU 利用率保持在合理范围内.



(a) 未使用过载保护



(b) 使用过载保护

图 17 租户 CPU 利用率

6 方法局限及未来工作

6.1 滤波器参数设置

文中提出的方法可以在给定的硬件环境下实现多租户 Web 应用 CPU 资源的有效评估.但是,精确的评估结果需要准确而充分地理解方法使用的 Kalman 滤波器的参数设置.

(1)我们发现,在负载组成变化环境下,需要在滤波器迭代过程中根据近期的追踪数据动态更新 $\boldsymbol{Q}$ 矩阵;在负载组成固定和变化环境下,较大的 $\boldsymbol{R}$ 矩阵取值都会产生较大的评估误差.

(2)我们发现,通过设置适当的初始状态向量 $\hat{\boldsymbol{x}}_0$,并利用转换矩阵 $\boldsymbol{A}$ 在滤波器迭代过程中动态修正状态向量,可以加快迭代收敛,提高方法的精确度和适应性.

(3)我们根据负载组成的变化周期 T_{change} 设置监测窗口长度 T,当 $T_{\text{change}} \gg T$ 时,可以提高方法的精确度和适应性.在本文中,我们根据经验,手动设置监测窗口长度.在未来工作中,我们将监测负载组成,预测变化趋势,进而实现监测窗口长度的自动设置和动态调整.

6.2 负载复杂度

在本文第 1 个实验案例中,我们使用 TPC-W 电子商务测试基准测试本文方法的精确度和适应性.在上述负载环境下,各租户的 CPU 资源状态是不相关的,满足使用 Kalman 滤波的可识别条件.但是,如果要进一步提高多租户性能隔离的精确度和效果,需要以事务为粒度,将各个事务的平均服务时间作为状态向量参数,进行评估分析.在这种情况下,由于同一会话的各事务间存在依赖关系,因此违背了可识别条件,将影响评估效果.在未来工作中,我们将针对这一问题展开研究.

在第 2 个实验案例中,我们选择 Trustie Forge 系统作为实验对象,对宿主在同一中间件服务器上的 12 个租户进行资源评估,取得了较好的评估效果.在未来工作中,我们将在更高密度的租户负载环境下,研究本文方法的评估效果.

7 相关工作

在共享 Java 服务器中进行 CPU 资源的在线度量仍存在技术挑战[5].已有工作主要通过本地代码库或程序转换等方法进行 CPU 资源的在线度量.

基于本地代码的方法被广泛使用[15-17].该方法本质上是一种基于采样的度量方法,其精确度依赖于采样时间戳的精密度.由于事务型应用每次请求处理所使用的 CPU 时间极短,因此对方法使用的时间戳精密度要求很高.例如,Magpie[15]使用 Windows 事件追踪机制实现 Windows 操作系统环境下 CPU 资源的精确度量.类似的方法还包括 Linux 操作系统环境下的 Trace 工具包[16]以及 Solaris 操作

系统环境下的 DTrace[17]. 方法依赖于操作系统的特性，甚至需要修改操作系统，因此存在系统兼容性问题. 另外，方法需要对应用程序的源代码或二进制文件进行探针注入，但是，中间件平台的提供者可能无权修改租户应用. 相比之下，本文方法不需要进行探针注入，不依赖于操作系统特性，具有非侵入性和操作系统独立性的优点.

Binder 等人利用程序转换技术，将 Java 程序字节码流量转换为 CPU 资源的使用量[5]，方法不存在兼容性问题，但存在较大性能开销（大于 30%）[6]. 林海略等人[2]使用一种渐进式的方法来设置程序转换的检查点，从而得到开销较低且保证精度要求的程序检查点集合. 方法首先采用保守的策略找到一组能够保证精度要求的检查点，随后在不损害精度的前提下逐渐缩减检查点的个数. 但是，这种方式需要经过较长时间的检查点配置和选择. 本文方法利用生产环境中常用的监测数据进行资源评估，因此不会产生性能开销.

最近，一些研究工作利用统计分析方法估算资源使用量. 例如 Zhang 等人[7]利用多元回归分析估算事务的平均服务时间. 但是，回归分析方法的精确度依赖于长时间、高质量的样本作为输入，容易受到时变资源状态以及样本异常点的影响. 已有工作[8]以及本文的实验结果表明此类方法难以对动态系统的时变资源状态进行有效评估，易产生较大误差.

Woodside 等人[18]提出基于 Kalman 滤波的性能模型参数在线调整方法. 方法通过监测响应时间并构造系统的性能模型，滤波器的优化目标是减小模型预测得到的响应时间与观测得到的响应时间之间的误差. 方法使用非线性的方法，通过近似地对性能模型求偏导数获得 $\boldsymbol{H}$，即计算每一个参数的微小变化对结果的影响. 因此，方法计算的时间复杂度是参数个数与性能模型求解时间的乘积，通常需要较长的计算时间[19]. 本文提出一种线性算法，不依赖于复杂的性能模型，复杂度降为 $O(N^3)$，可以即时算出结果，更适用于多租户 Web 应用的在线性能管理.

8　结束语

中间件共享是云计算模式中一种重要的资源共享方式. 但是，这种方式容易导致宿主在同一中间件服务器上的多个租户间产生性能干扰. 因此，需要为租户提供性能隔离的服务实例. 在线度量租户对资源的使用情况是实现性能隔离的前提条件，但是，在共享中间件服务器中直接度量 CPU 资源使用，需要注入探针，将引起性能开销，并依赖于操作系统的支持. 文中针对普遍使用的 Java 中间件服务器，提出一种基于 Kalman 滤波的多租户 Web 应用 CPU 资源动态评估方法，提供了多租户 Web 应用 CPU 资源评估的新途径.

本文通过两个实验案例分析方法的评估效果、影响因素和挑战. 在第 1 个应用案例中，我们设计了一系列实验，分析滤波参数设置对资源评估精确度的影响. 实验结果表明，通过设置适当参数，本文方法可动态适应持续变化的负载环境，并且与直接度量方法相比，具有可接受的评估误差. 在第 2 个应用案例中，我们在支撑多个软件开发团队的 Forge 生产环境中验证方法的精确度，实验还表明方法可用于检测侵占型租户，并避免共享中间件服务器 CPU 过载.

参　考　文　献

[1] Zhang Q, Cheng L, Boutaba R. Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 2010, 1(1): 7-18

[2] Lin Hai-Lue, Han Yan-Bo. Performance management for multi-tenant web applications. Chinese Journal of Computers, 2010, 33(10): 1881-1895(in Chinese)
(林海略，韩燕波. 多租户应用的性能管理关键问题研究. 计算机学报，2010，33(10)：1881-1895)

[3] Guo C J, Sun W, Huang Y, Wang Z H, Gao B. A framework for native multi-tenancy application development and management//Proceedings of the 9th International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-commerce and E-Services. 2007: 551-558

[4] Li X H, Liu T C, Li Y, Chen Y. SPIN: Service performance isolation infrastructure in multi-tenancy environment//Proceedings of the 6th International Conference on Service-Oriented Computing. Sydney, Australia, 2008: 649-663

[5] Binder W, Hulaas J. A portable CPU-management framework for Java. IEEE Internet Computing, 2004, 8(5): 74-83

[6] Hulaas J, Binder W. Program transformations for portable CPU accounting and control in Java//Proceedings of ACM SIGPLAN Symposium on Partial Evaluation & Program Manipulation. Verona, Italy, 2004: 169-177

[7] Zhang Q, Cherkasova L, Mathews G, Greene W, Smirni E. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads//Proceedings of the Middleware. Newport Beach, CA, 2007: 244-265

[8] Cherkasova L, Ozonat K. Automated anomaly detection and performance modeling of enterprise applications. ACM Transactions on Computer Systems, 2009, 27(3): 1-32

[9] Kalman R E. A new approach to linear filtering and prediction problems. Transactions of the ASME-Journal of Basic Engineering, 1960, 82(D): 35-45

[10] Lazowska E D, Zahorjan J, Graham G S, Sevcik K C. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Upper Saddle River: Prentice-Hall, Inc., 1984

[11] Wang Wei, Zhang Wen-Bo, Wei Jun, Zhong Hua, Huang Tao. A resource-aware performance diagnostic method for Web applications. Journal of Software, 2010, 21(2): 194-208(in Chinese)
(王伟, 张文博, 魏峻, 钟华, 黄涛. 一种资源敏感的 Web 系统性能诊断方法. 软件学报, 2010, 21(2): 194-208)

[12] Jazwinski A H. Stochastic Processes and Filtering Theory. New York: Academic Press, 1970

[13] Wang W, Zhang W B, Wei J, Huang T. A QoS-enabled WorkManager model for Web application servers//Proceedings of the 7th International Conference of Quality Software. Portland, Oregon, USA, 2007: 40-49

[14] Gunther N J. Mind Your Knees and Queues. CMG Measure IT Issue 7.08.

[15] Barham P, Donnelly A, Isaacs R, Mortier R. Using magpie for request extraction and workload modeling//Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'04). Berkeley, CA, USA, 2004: 18-18

[16] Yaghmour K, Dagenais M R. Measuring and characterizing system behavior using kernel-level event logging//Proceedings of the Annual Conference on USENIX Annual Technical Conference. Berkeley, CA, USA, 2000: 2-2

[17] Cantrill B M, Shapiro M W, Leventhal A H. Dynamic instrumentation of production systems//Proceedings of the Annual Conference on USENIX Annual Technical Conference. Berkeley, CA, USA, 2004: 2-2

[18] Woodside M, Zheng T, Litoiu M. The use of optimal filters to track parameters of performance models//Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems(QEST'05). Torino, Italy, 2005: 74-84

[19] Woodside M, Franks G, Petriu D C. The future of software performance engineering//Proceedings of the 2007 Future of Software Engineering, Washington, DC, USA, 2007: 171-187

WANG Wei, born in 1982, Ph. D.. His research interests include distributed computing and resource management.

HUANG Xiang, born in 1982, Ph. D. candidate. His research interests include distributed computing and performance engineering.

ZHANG Wen-Bo, born in 1976, Ph. D., associate professor. His research interests include distributed computing and software engineering.

WEI Jun, born in 1970, Ph. D., professor, Ph. D. supervisor. His research interests include distributed computing and software engineering.

ZHONG Hua, born in 1971, Ph. D., professor, Ph. D. supervisor. His research interests include distributed computing and software engineering.

HUANG Tao, born in 1965, Ph. D., professor, Ph. D. supervisor. His research interests include distributed computing and software engineering.

Background

The main research interests of the author include distributed computing and resource management. This paper is primarily supported by the National Natural Science Foundation of China (Nos. 61100068, 61173003).

Middleware sharing is one of the important resource sharing approaches in cloud computing. However, a shared middleware server easily causes interference in performance between multiple hosted tenants. This interference affects infrastructure resources as well as applications and services that are hosted on shared resources but that need to be made available in multiple performance isolated instances. A key requirement in performance isolation of the shared Java middleware server is the knowledge of the resource consumption of the various tenants. However, direct measurement of resource consumption requires instrumentation, incurs overhead, and assumes OS support. Recently, regression analysis has been applied to indirectly approximate resource consumption, but challenges still remain in estimating time-varying states in dynamic systems. This paper proposes a Kalman filter-based approach to offer a solution to the problem of dynamically estimating the CPU consumption of a multi-tenancy application in a shared Java middleware server, and we discuss the challenges involved in this approach. The authors investigate factors that impact the efficiency and accuracy of the approach in estimating time-varying states via two case studies. Experimental results show that, even under continuously changing workload conditions, estimation results are in agreement with the corresponding measurements with acceptable estimation errors, especially with appropriately tuned filter settings taken into account. This paper also demonstrates the utility of our approach in identifying the aggressive tenants and in avoiding shared middleware server CPU overloading.