ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.2, February 2010, pp.194−208
doi: 10.3724/SP.J.1001.2010.03781

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

一种资源敏感的Web应用性能诊断方法*

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

[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](中国科学院 软件研究所 计算机科学重点实验室,北京 100190)

[3](中国科学院 研究生院,北京 100049)

Resource-Aware Performance Diagnostic Method for Web Applications

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

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

[2](State Key Laboratory Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

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

+ Corresponding author: E-mail: wangwei@otcaix.iscas.ac.cn

Wang W, Zhang WB, Wei J, Zhong H, Huang T. Resource-Aware performance diagnostic method for Web applications. *Journal of Software*, 2010,21(2):194−208. http://www.jos.org.cn/1000-9825/3781.htm

Abstract: This paper proposes a resource-aware performance diagnostic method. For transactions in Web applications, the proposed method constructs performance profile chains based on the resource service time, which is stable for different workload characteristics. According to the anomaly of resource service time at application runtime, the proposed method provides an efficient solution to performance anomaly detection, location and diagnosis. Experimental results show that this method can effectively detect performance anomalies caused by different resource bottlenecks with changing workload characteristics.

Key words: Web application; performance diagnostic; anomaly detection; resource-aware

摘 要: 提出一种资源敏感的性能诊断方法.对于 Web 应用事务,该方法利用资源服务时间对于不同负载特征相对稳定的特点建立性能特征链,并依据运行时资源服务时间异常实现性能异常的有效检测、定位和诊断.实验结果表明,该方法可适应系统负载特征变化,诊断各种资源使用相关的性能异常.

关键词: Web 应用;性能诊断;异常检测;资源敏感

中图法分类号: TP311 **文献标识码**: A

多层架构的Web应用模式已经广泛应用到电子银行、网上支付等关键系统中,其性能保障非常重要,而在线的性能诊断是关键技术之一.随着Web应用的规模和复杂度的逐渐增加,对其进行在线性能诊断的难度和代价

* Supported by the National Natural Science Foundation of China under Grant No.90718033 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z134, 2007AA010301 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2009CB320704 (国家重点基础研究发展计划(973)); the Major Science and Technology Project of China under Grant No.2009ZX01043-001-05 (国家科技重大专项)

Received 2008-06-16; Revised 2009-09-11; Accepted 2009-12-07

大为增加.首先,开放的Internet环境下,Web应用的负载特征(负载大小以及事务混合比例等)是动态变化的,而Web应用在不同的负载特征下性能差异较大,这要求性能诊断方法具有良好的动态适应性[1,2].同时,多层架构的Web应用依赖于Web应用服务器、数据库等基础软件作为运行环境,这些软件自身的复杂性为Web应用的在线性能诊断带来严峻的挑战.

Web应用的自适应在线性能诊断是该领域的研究热点,如何遴选具有动态适应性的特征度量(profile metrics)是其关键所在[1,2].基于规则的阈值方法通过设置性能度量的阈值检测性能异常[3−5],当监测的性能度量超出阈值时,将根据规则执行预定动作.但是,由于阈值的大小是静态、依赖经验设置的,并且对于复杂系统需要人工设置大量阈值,因此,该方法难以适应负载特征变化较大的Web应用.异常检测(anomaly detection)方法利用预先构造的系统正常性能特征模型诊断性能异常.对于多层架构的Web应用,已有异常检测方法通常采用延迟时间等性能度量作为性能特征建立分阶段的性能特征模型[6−11].然而,性能度量容易受到负载特征变化的影响,因此严重影响该方法的有效性.

本文提出一种资源敏感的性能诊断方法.该方法利用相对稳定的资源服务时间作为性能特征,并根据其建立分阶段的性能特征链,依据运行时资源服务时间异常实现性能异常的检测、定位和诊断.实验结果表明,该方法可适应 Web 应用负载特征的动态变化,有效检测、定位和诊断 Web 应用中各种资源使用相关的性能异常.同时,由于采用平台无关的排队模型计算服务时间的近似值,与常用的字节码注入和本地代码调用等资源监测方法相比,该方法系统开销小,不影响 Web 应用的正常运行.

本文第 1 节分析引起 Web 应用性能异常的主要原因,并针对已有性能诊断方法存在的不足,给出基于资源服务时间的性能特征链.第 2 节给出基于性能特征链的性能异常诊断方法.第 3 节给出对本文方法的实验及结果.第 4 节比较相关研究工作.最后是全文的结论.

1 多层 Web 应用的性能特征建模

如图 1 所示,多层架构的 Web 应用通常包含前端的 Web 应用服务器和后端的数据库服务器,客户端请求在经过服务器端各层处理后返回响应.

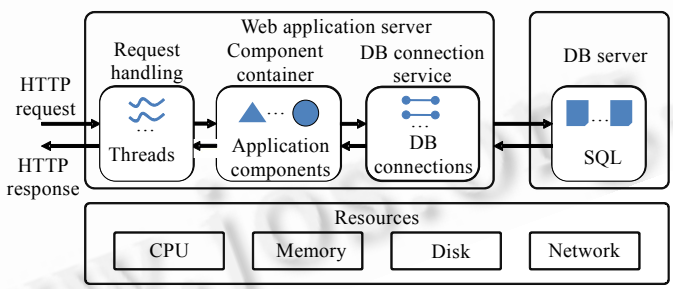


Fig.1 A typical Web application host environment: Web application server and DB server

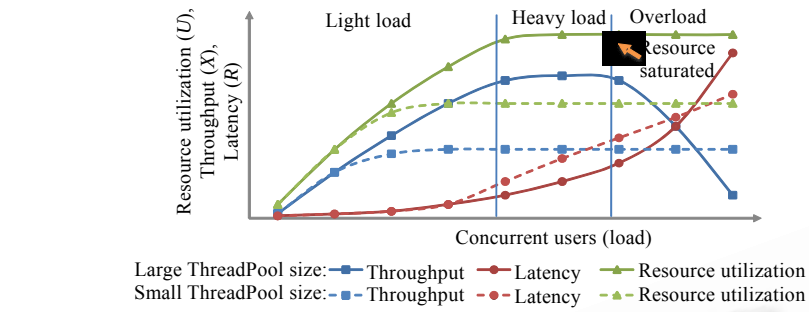
图 1 一个典型的 Web 应用运行环境:Web 应用服务器与数据库服务器

在处理客户端请求的过程中,多个处理任务结合在一起完成一个业务要求,称为事务.事务是多层架构Web应用性能特征的主要研究对象[6],本文主要关注服务器端事务处理的性能分析和诊断.为方便后文的讨论,对于一个包含M种事务的Web应用,T_i表示第i种事务,且 $1\le i\le M$.

1.1 性能异常分析

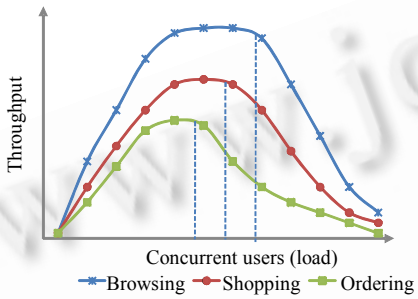
Web应用性能受到服务器资源的明显影响,我们将服务器端事务处理过程中涉及的资源分为逻辑资源(如进程、线程、数据库连接等)和物理资源(如CPU、内存、网络、硬盘等),其中,逻辑资源配置不当以及过载导致的物理资源饱和是引起性能异常的两个重要原因[2].下面通过一个典型的事务性Web应用基准测试TPC-W[12]分析上述两个原因.图 2(a)显示的是TPC-W基准测试中,Web应用在不同大小负载下,吞吐量X、平均延迟时间R

以及服务器CPU资源效用U^{cpu}(即CPU使用率)等性能度量的变化情况.



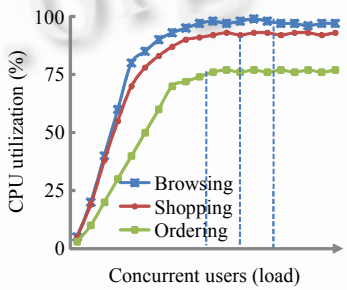
(a) Effects of load size and system configuration on resource utilization, throughput and latency time

(a) 负载大小及系统配置对资源效用、吞吐量、延迟时间的影响



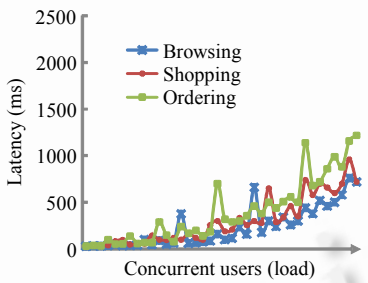
(b) Effects of three transaction mixes on throughput

(b) 3 种事务混合对吞吐量的影响



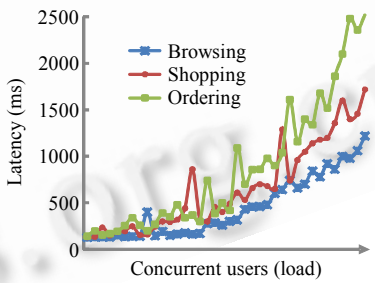
(c) Effects of three transaction mixes on CPU utilization

(c) 3 种事务混合对资源效用的影响



(d) Effects of three transaction mixes on “Home” transaction latency time

(d) 3 种事务混合对“Home”事务延迟时间的影响



(e) Effects of three transaction mixes on “Shopping Cart” transaction latency time

(e) 3 种事务混合对“Shopping Cart”事务延迟时间的影响

Fig.2

图 2

可以看出,负载大小及逻辑资源配置是影响上述度量变化的关键因素.当线程池配置过大(large ThreadPool size)时,随着负载的增加,X和U^{cpu}逐渐增大;当U^{cpu}到达最大值后,X也随之到达最大值;负载继续增加后,X迅速下降.此时,系统过载导致CPU资源的需求量大于其最大可供给量,即资源达到饱和,称为物理资源瓶颈.当线程池配置过小时(small ThreadPool size),随着负载的增加,U^{cpu}未能达到饱和即保持稳定,X也未能达到最大值即保持稳定,R由于等待队列长度的增大而增长.此时,线程池的设置限制了CPU资源的使用,称为逻辑资源瓶颈.

性能异常的检测和诊断是保障Web应用性能的关键技术,例如,系统过载检测、资源瓶颈诊断、系统配置诊断等.异常检测方法通过判断系统运行时的性能特征(performance profile,简称pp)与正常性能特征之间是否存在偏差来检测性能异常.在进行性能异常检测时,吞吐量、延迟时间、资源效用等性能度量常被用于刻画性能

特征[4,10,13−16].但是,通过TPC-W基准测试可以发现,受负载特征动态变化的影响,上述性能度量存在不稳定性.TPC-W包含14种事务,其中8种涉及数据库读、写操作,6种只涉及数据库读操作.图2(b)~图2(e)分别显示的是不同事务混合模式下(browsing,shopping,ordering),服务器CPU资源效用、吞吐量以及两种典型事务(home, shopping cart)的延迟时间等性能度量的变化情况.可以发现,即使负载大小相同,事务混合的变化也会引起吞吐量、资源效用的变化,进而导致物理资源瓶颈的变化,如图2(b)、图2(c)所示;同时,事务混合的变化也会引起事务延迟时间的变化,如图2(d)、图2(e)所示.上述变化造成性能度量的不稳定性使得已有工作难以有效检测物理资源瓶颈和逻辑资源瓶颈.如何遴选适当的度量(fitting metrics)刻画Web应用的性能特征,仍然具有挑战性[1,2].

1.2 资源的使用和控制建模

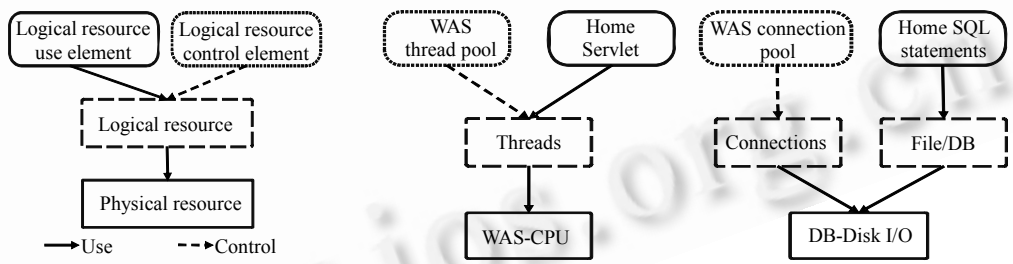
针对第1.1节提出的问题,首先分析Web应用的运行环境,给出一种资源使用和控制模型.在此基础上,在应用层分解Web应用的事务处理过程,以资源消耗链的形式构造模型实例,用于性能异常的检测和诊断.

Web应用服务器(Web application server,简称WAS)是Web应用的主要运行环境,如图1所示,WAS提供多种性能相关的资源管理服务(如线程池、数据库连接池等)[17].在图3(a)所示模型中,我们对资源的使用和控制关系进行定义和区分:

使用(use).Web应用及运行环境通过逻辑资源使用物理资源,例如通过线程(threads)使用CPU、内存等资源,通过文件(files)使用磁盘IO资源,通过数据库连接(connections)使用数据库资源等等.

控制(control).并发是Web应用事务处理的基本特征.Web应用的运行环境通过资源池(resource pooling)控制事务的并发规模[18,19],当并发事务超出资源池大小限制时,事务处理任务会进入资源池等待队列.

在Web应用及运行环境中,我们将Web应用的事务逻辑称为逻辑资源使用元素(logic resource use element,简称LRUE),将资源池及等待队列称为逻辑资源控制元素(logic resource control element,简称LRCE).领域专家可以建立Web应用的资源使用和控制模型,图3(b)显示了TPC-W中Home事务涉及的逻辑资源使用元素、逻辑资源控制元素、逻辑资源、物理资源及其之间的使用和控制关系("WAS-"表示Web应用服务器相关度量,"DB-"表示数据库服务器相关度量).



(a) Resource use and control model
(a) 资源使用和控制模型

(b) Model example: Home transaction in TPC-W
(b) 模型示例:TPC-W中的Home事务

Fig.3
图3

刻画资源的使用和控制关系的目的在于区分逻辑资源的使用和控制,为进一步区分物理资源瓶颈和逻辑资源瓶颈建立基础.Web应用的事务逻辑由应用组件及数据库操作组成,是主要的逻辑资源使用元素,其性能特征是检测物理资源瓶颈的关键.逻辑资源瓶颈与逻辑资源的配置相关,即与逻辑资源控制元素相关,资源池及等待队列的性能特征是检测逻辑资源瓶颈的关键.

事务处理过程可进行链式划分,如基于阶段的体系结构模式SEDA[20]和基于拦截器的体系结构模式Interceptor[21].在资源使用和控制模型基础上,我们通过事务跟踪方法在应用层分解Web应用的事务处理过程,使用组件方法、资源池、SQL语句等事务处理各阶段的逻辑资源使用元素或控制元素,建立链式的资源使用和控制模型实例.模型实例由若干阶段(P)组成,阶段内事务处理涉及的逻辑资源使用元素或控制元素是可知的,性

能特征是可度量的,这些信息构成了阶段的上下文信息.上下文信息为进一步实现性能异常的定位和隔离提供依据.图 4 显示了TPC-W中Home事务处理过程包含的 5 个阶段:P1) Web应用服务器接收到客户端请求后,从线程池(LRCE)分配一个空闲线程处理客户端请求,若不存在空闲线程,则进入等待队列;P2) 执行客户端请求的应用组件方法(LRUE);P3) 从数据库连接池(LRCE)获得一个空闲数据库连接,若不存在空闲连接,则进入等待队列;P4) 执行SQL语句(LRUE)并返回结果;P5) 应用组件(LRUE)处理SQL语句返回的结果,返回客户端响应.

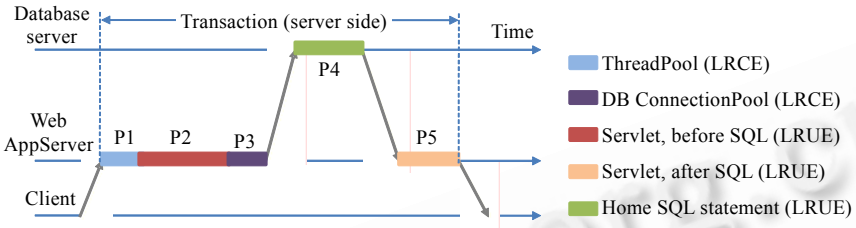


Fig.4 Home transaction in TPC-W

图 4 TPC-W 中的 Home 事务

1.3 基于资源服务时间的特征度量

在获得链式模型实例后,选择可动态适应 Web 应用负载特征变化的性能度量并刻画性能特征,是进一步需要解决的问题.我们提出一种基于资源服务时间的特征度量.资源服务时间表示 Web 应用事务处理过程中,逻辑资源对物理资源的实际使用时间,不包含操作系统在处理任务时因排队、调度所耗费的管理时间.延迟时间由服务时间与管理时间组成,当事务处理并发量(负载大小)或事务混合变化时,系统资源的竞争将造成管理时间的变化,进而造成延迟时间的不稳定性.与延迟时间不同,服务时间的大小仅与事务逻辑以及系统物理资源相关.当事务逻辑和系统物理资源能力确定时,服务时间不受系统负载大小及事务混合等变化特征的影响.因此,利用服务时间刻画性能特征具有动态适应性.

服务时间不易直接测量,需要使用平台相关的系统底层方法[22,23].这些方法在系统开销、兼容性方面仍存在不足.我们利用排队模型模拟Web应用事务处理过程,建立延迟时间、服务时间以及资源效用的关联关系.首先考虑一个简单的队列系统,S代表系统中某一任务的平均服务时间,L代表新任务到达时系统队列长度,R代表任务在系统中的平均延迟时间.可知,任务的延迟时间等于其自身的服务时间与队列等待时间之和:

$$R=S+S\times L \tag{1}$$

在闭合队列模型中,当系统中存在N个客户时,任务的队列长度L等于系统中有$N-1$ 个客户时的平均队列长度,可近似为$L\approx Q\times(N-1)/N$[24],其中,Q表示平均队列长度.当N不断增大时,$(N-1)/N$趋近于 1,因此队列长度L可以近似为平均队列长度Q,即$L\approx Q$.由此得到等式:

$$R=S+S\times Q \tag{2}$$

设X代表系统平均吞吐量,应用Little's法则[24]$Q=X\times R$,可得到等式:

$$R=S+S\times(X\times R) \tag{3}$$

进一步地,根据效用法则(utilization law)[24],资源效用等于吞吐量乘以服务时间,即$U=X\times S$,推导出等式:

$$Q=U/(1-U) \tag{4}$$

式(4)说明了队列平均任务数与资源效用之间的关系.典型的计算机操作系统在处理多个任务时使用分时规则,如果系统平均有N个并发任务,则每个任务被分配到 $1/N$的平均服务时间,因此对于事务T_i的第j阶段,存在等式:

$$R_{i,j}=S_{i,j}\times N \tag{5}$$

由于 $N=Q+1$,根据式(4)和式(5)可得到计算服务时间的等式:

$$S_{i,j}=R_{i,j}\times(1-U) \tag{6}$$

式(6)说明了服务时间、延迟时间以及资源效用之间的关系.由第 1.1 节的分析可推测,随着负载的增加,当

系统出现过载造成资源饱和时,资源效用保持稳定,但延迟时间持续增加,因此,由式(6)计算得到的服务时间增加.上述推测在 TPC-W 测试中得到验证,图 5 显示的是两种典型事务(home,shopping cart)在不同负载大小情况下由式(6)计算得到的服务时间的变化情况.当 CPU 资源未达到饱和时,计算得到的服务时间基本保持不变,这与前述服务时间具有动态适应性是相符的;而当 CPU 资源饱和时,计算得到的服务时间与实际服务时间出现明显的偏差.由此,我们可以利用式(6)计算得到服务时间在物理资源非饱和时相对稳定的特点,进行物理资源瓶颈和逻辑资源瓶颈的检测和诊断:当服务时间变化时,物理资源达到饱和,发生物理资源瓶颈;当服务时间稳定时,物理资源没有达到饱和,如果逻辑资源控制元素出现等待队列,则发生逻辑资源瓶颈.

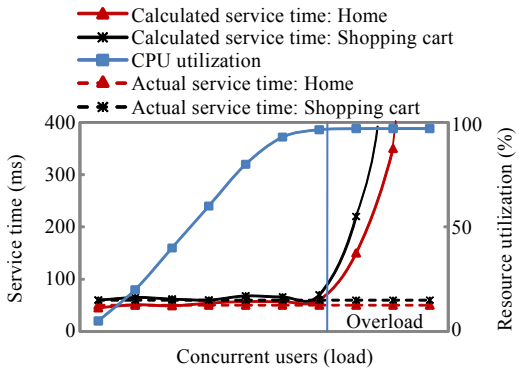


Fig.5 Relation between actual service time and calculated service time

图 5 实际服务时间与计算服务时间的关系

2 异常检测与诊断方法

本节利用基于资源服务时间的特征度量,构造链式模型实例的性能特征,并给出进行异常检测和诊断的相关算法.

2.1 性能特征链构造算法

性能特征链基于第 1.1 节提出的资源使用和控制模型,反映事务处理过程中服务器端各种资源使用和控制元素的性能特征.对于事务处理的各阶段,性能数据及上下文信息易从服务器端获得[4,25],因此不难构造性能特征链,实现事务跟踪.与以往方法采用延迟时间作为特征度量不同,我们采用资源服务时间来描述性能特征.

在事务T_i处理过程中,每个阶段的性能状态在固定的时间间隔都会被监测记录,这种时间间隔称为监测窗口.多个连续的监测窗口形成特征采样窗口(profile sampling interval).为便于描述,我们给出以下记号:

- $Phase_i$表示事务T_i处理过程中的阶段数量.
- $U_{i,j}^r$表示在事务T_i第j阶段($1\le j\le Phase_i$),物理资源r的平均效用值.
- $ele_{i,j}$表示在事务T_i第j阶段($1\le j\le Phase_i$)的LRCE或LRUE.
- $QueueU_{i,j}$表示LRCE的等待队列长度不为 0 的监测窗口占所有监测窗口的比例,即特征采样窗口内等待队列的使用率.
- $N_{i,j}$表示$ele_{i,j}$在单位监测窗口内被使用的次数,即完成该阶段事务处理的计数.
- $R_{i,j}$表示$ele_{i,j}$的平均延迟时间.
- $S_{i,j}$表示$ele_{i,j}$的服务时间.
- $pp_{i,j}$表示$ele_{i,j}$的性能特征,选择$S_{i,j}$作为LRUE的性能特征, $QueueU_{i,j}$作为LRCE的性能特征.

在一个包含p个单位监测窗口的特征采样窗口内:1) 对于LRUE,在每个单位监测窗口内收集$[N_{i,j},R_{i,j},U_{i,j}^r]$.将资源效用分类,表示为 $\{U_1^r=1\%,U_2^r=2\%,...,U_k^r=k\%,...,U_{100}^r=100\%\}$,其中$k$表示类别.将$R_{i,j}$和$N_{i,j}$按资源效用归

类统计,即特征采样窗口内所有 $U_{i,j}^r$ 相同的监测窗口,其对应的$R_{i,j}$和$N_{i,j}$被归为一类.例如,在当前监测窗口内,$ele_{i,j}$存在$N_{i,j}$次使用,平均延迟时间为$R_{i,j}$, $U_{i,j}^{\text{cpu}}$ 为 10%,则($N_{i,j}$,$R_{i,j}$)作为值对被归为事务T_i的类别 k=10 中.归类的目的是为了将延迟时间与资源效用相关联,同时,利用阶段内的事务处理计数体现该监测窗口内收集得到的延迟时间是否具有代表性;2) 对于 LRCE,记录等待队列非空的单位监测窗口计数 q.

在特征采样窗口结束时,对LRUE和LRCE分别进行性能特征分析,构造性能特征链.算法以伪代码形式描述如图 6 所示.对于LRUE(算法第 2 行~9 行),计算每一个资源效用类别下所有($N_{i,j}$,$R_{i,j}$)值对的延迟时间加权平均值,然后利用等式(6)计算得到资源效用为 U_k^r 时的服务时间$R_{i,j}$,最后利用累计概率分布函数(cumulative distribution function,简称CDF)[26]处理所有资源效用分类对应的服务时间样本,得到性能特征采样窗口内LRUE的服务时间近似值.对于LRCE(算法第 10 行~13 行),计算性能特征采样窗口内的等待队列使用率.

Algorithm 1. Build the performance profile chain associated with resource r in a profile sampling window.
Input: Set of performance metrics P collected in a profile sampling window:

$$P=\{[N_{i,1},R_{i,1},U_{i,1}^r],...,[N_{i,j},R_{i,j},U_{i,j}^r],...,[N_{i,Phase_i},R_{i,Phase_i},U_{i,Phase_i}^r]\},$$

where $r\in\{CPU,Memory,DiskIO,\ldots\}$;

Output: Performance profile chain $ppc_i(r)$: $T_i \xrightarrow{r} \{pp_{i,1}, pp_{i,2},..., pp_{i,j},..., pp_{i,Phase_i}\}$.

```
1.  for each ele_ij in T_i do
2.      if ele_ij is LRUE
3.          for each U_k^r , 0≤k≤100 do
4.              R_ij = Σ(R_ij × N_ij) / ΣN_ij ;   /* Calculate weighted means of all (N_ij,R_ij) pairs which have the same U^r */
5.              S_ij,k = R_ij × (1−U_k^r) ;   /* Calculate service time R_ij by Equation (6) when resource utilization is U_k^r */
6.          end for
7.          S_ij = CDF(0.5,{S_ij,1,S_ij,2,...,S_ij,k,...,S_ij,100}) ;   /* Generate performance profile by CDF */
8.          pp_ij = S_ij ;
9.      end if
10.     if ele_ij is LRCE   /* Calculate utilization of waiting queue of LRUE within a profile sampling window */
11.         QueueU_ij=q/p;
12.         pp_ij=QueueU_ij;
13.     end if
14. end for
```

Fig.6 Algorithm for construction of performance profile chain

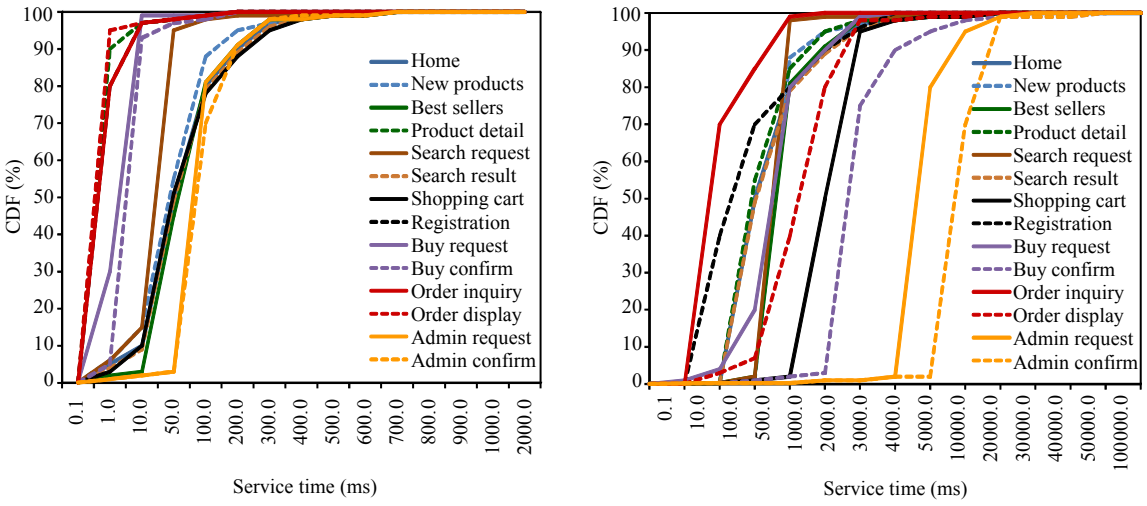
图 6 性能特征链构造算法

在利用式(6)计算服务时间时,代入不同种类资源的效用值,可以得到事务T_i关联不同资源的性能特征链(performance profile chain,简称ppc),表示为$ppc_i(r)$:

$$T_i \xrightarrow{r} \{pp_{i,1}, pp_{i,2},..., pp_{i,j},..., pp_{i,Phase_i}\},$$

其中,$1\le j\le Phase_i$,$1\le i\le M$,$r\in\{CPU,Memory,DiskIO,\ldots\}$.

在性能特征链构造过程中,我们使用累计分布函数处理服务时间样本集合,而非线性回归分析或平均值方法目的是为了降低采样值中异常点的影响.图 7(a)、图 7(b)分别显示了 TPC-W 中 14 种事务在轻载和过载情况下的服务时间 CDF 分布.其中,除了各曲线的头部和尾部存在一些异常点以外,大量的相似点都集中在 CDF 曲线中间.据此特征,选择 CDF 为 50%的值作为服务时间的近似值.



(a) Tiny load (b) Over load

(a) 轻载情况 (b) 过载情况

Fig.7 CDF of the service time for 14 transactions in TPC-W

图 7 TPC-W14 种事务的服务时间 CDF 分布

2.2 基于性能特征链的异常检测与诊断算法

利用性能特征链构造方法,分别构造Web应用性能特征链的参考实例和运行时实例,通过检查运行时实例与参考实例的偏差是否大于设定阈值,判断是否出现异常.定义异常标记$anomaly_{i,j}$表示$ele_{i,j}$是否出现异常,其取值包括{−1,0,1},0 表示没有出现异常,1 表示出现异常,−1 表示$ele_{i,j}$与资源r不存在关联.由此,我们得到经过异常标记的性能特征链$ppc_i(r)$:

$$T_i \xrightarrow{r} \{...,anomaly_{i,j},...\}.$$

表 1 是TPC-W的性能特征链示例,分别与Web应用服务器CPU、数据库服务器Disk I/O等资源关联(“WAS-”和“DB-”分别表示与性能特征链相关联的Web应用服务器资源和数据库服务器资源),其中,$ele_{i,j}$相同的阶段(线程池和数据库连接池)被合并.

Table 1 Performance profile chain in TPC-W

表 1 TPC-W 的性能特征链

Phase	Element	WAS_CPU	DB_Disk_IO
WAS ThreadPool	ThreadPool	1	−1
Application component	Home	0	−1
	Product_Detail	0	−1
	…	…	…
	Admin_Confirm	0	−1
DB ConnectionPool	ConnectionPool	−1	0
SQL statement	Home_SQL	−1	0
	Product_Detail_SQL	−1	0
	…	…	…
	Admin_Confirm_SQL	−1	0

对于Web应用中M个事务的性能特征链$\{ppc_1(r),ppc_2(r),\ldots,ppc_i(r),\ldots,ppc_M(r)\}$,结合第 1.3 节服务时间在物理资源非饱和时的稳定特点,假设不同物理资源之间不存在相互影响,性能异常检测和诊断算法以伪代码形式

描述如图 8 所示.对于事务T_i的性能特征链$ppc_i(r)$,算法首先根据$ppc_i(r)$中的LRUE是否出现异常来判断物理资源r是否存在瓶颈;接着,根据$ppc_i(r)$中的LRCE是否出现异常来判断相应的$ele_{i,j}$是否存在逻辑资源瓶颈.例如在表 1 示例中,逻辑资源控制元素线程池出现异常,而逻辑资源使用元素未出现异常,根据算法,可以诊断CPU,Disk I/O等物理资源未出现瓶颈,但线程池设置过小导致逻辑资源瓶颈.

参考实例反映 Web 应用的正常性能特征,可在轻负载下对 Web 应用进行模拟访问获得,与关联分析方法的学习过程相似,但由于异常检测不依赖历史"病例",并且参考实例不受负载特征变化的影响,因此学习过程代价较低.

Algorithm 2. Detect and diagnose the performance anomaly.
Input: Set of performance profile chains $ppcs=\{ppc_1(r),ppc_2(r),\ldots,ppc_i(r),\ldots,ppc_M(r)\}$;
Output: Root cause of the performance anomaly:
physical resource bottleneck r or logical resource bottleneck $ele_{i,j}$.

```
1.  for each ppc_i(r) in ppcs do
2.      for each anomaly_{i,j} in ppc_i(r)
3.          if (anomaly_{i,j} && ele_{i,j} is LRUE)
4.              Physical resource r is a bottleneck;
5.          end if
6.          if (anomaly_{i,j} && ele_{i,j} is LRCE)
7.              Logic resource ele_{i,j} is a bottleneck;
8.          end if
9.      end for
10. end for
```

Fig.8 Algorithm for performance anomaly detection and diagnosis
图 8 性能异常检测和诊断算法

3 系统实验与结果

上述方法已经在中国科学院软件所软件工程中心开发的Java EE应用服务器OnceAS[27]平台进行了原型实现.本节在一个典型的Web应用环境下进行本文方法的有效性测试,并与基于字节码注入和本地代码调用的性能诊断工具进行系统开销对比测试.

3.1 实验方法

1. 测试基准

我们采用的测试环境是基于事务性Web测试基准TPC-W规范实现的在线书店(online bookstore)[28],共包含 14 种事务,其中 8 种涉及数据库读、写操作,属于订单相关(order-related)的事务;6 种只涉及读操作,属于浏览相关(browse-related)的事务.TPC-W可按不同比例混合事务,构成不同的事务混合模式,常用的 3 种模式包括浏览(browsing)、购买(shopping)和订购(ordering),见表 2.客户端采用负载生成器模拟用户请求事务,负载的大小可通过改变并发用户数量来调整.

Table 2 Transactions types and mixes in TPC-W
表 2 TPC-W 基准测试中的事务类型和混合模式

Web transaction	Browsing mix	Shopping mix	Ordering mix
Browse-Related	95.00	80.00	50.00
Order-Related	5.00	20.00	50.00

2. 方法有效性测试

本实验的目的是在不同负载特征及系统配置情况下,测试本文方法能否有效检测和诊断由资源瓶颈引起的性能异常,并与基于阈值的经验法则(rule of thumbs)相比较.测试中,客户端将依次产生浏览(browsing)、购买(shopping)和订购(ordering)这 3 种模式的负载(见表 3),其中,各阶段负载的最大值都超过了系统的容量,将造成系统过载.

考虑到测试中存在逻辑资源的参数配置空间过大的问题,我们利用自优化方法对参数配置空间进行裁剪.

自优化方法运用控制论或运筹学的理论、方法建立数学模型,依据不断改变的资源和环境状态自主决定系统参数的调整,使得系统性能保持在期望的范围内.已有工作(文献[29,30])将 Web 应用的服务质量保障转化为一个多约束的效用函数问题,采用爬山算法进行优化配置的搜索,具有很好的准确性和收敛性.由于在非优化配置下存在物理资源瓶颈或逻辑资源瓶颈,因此我们可以利用该方法进行本文模型的测试.我们将效用函数简化为吞吐量,优化目标是寻找吞吐量最大的参数配置.

Table 3 Changing input workload for TPC-W

表 3 TPC-W 负载变化情况

	Browsing	Shopping	Ordering
10 Minutes	0→100	0→100	0→100
20 Minutes	100	100	100
10 Minutes	100→0	100→0	100→0

定义效用函数*Utility*=*X*,其中,*X*为吞吐量.定义*P*维向量 $\vec{C}=\{c_1,c_2,...,c_p\}$ 表示Web应用服务器中可动态调整的配置参数,其中,每个参数c_i存在取值范围 $(c_i^{\min},c_i^{\max})$.自优化方法利用爬山算法搜索 $\vec{C}$ 相邻的参数配置空间,寻找可以提高效用函数值的参数配置.$\vec{C}$ 的相邻参数配置空间为 $M_C=\{\vec{C}+\Delta\overrightarrow{C_i}\}\cup\{\vec{C}-\Delta\overrightarrow{C_i}\}$, 其中, $\Delta\overrightarrow{C_i}$ 表示爬山算法在对于每个参数的搜索步长.实验中,针对效用函数定义 $\vec{C}=\{c_{tp},c_{cp}\}$,其中,c_{tp},c_{cp}分别表示Web应用服务器的线程池和数据库连接池配置,每个参数的取值范围均为(1,100),初始值均为 5.由于爬山算法是局部最优搜索算法,而表 2 中的不同事务混合将造成效用函数的局部最优解,因此,当实验中的事务混合发生变化时,爬山算法及所有参数配置将被重置为初始值.同时,为了便于测试数据的表示,我们将反馈控制的时间间隔(control interval)设置为特征采样窗口大小.

3. 系统开销测试

JProfiler和NetBeans Profiler是两种常用的Web应用性能分析工具[31,32],它们通过字节码注入以及本地代码调用方式跟踪Web应用的事务并捕获资源消耗.对于TPC-W的 3 种事务混合模式,我们分别在不同负载大小情况下对比测试本文方法与JProfiler以及NetBeans Profiler的系统开销.

4. 实验环境及参数设置

如图 9 所示,实验环境包括一台模拟 TPC-W 客户端的服务器(DELL PowerEdge 2000 PC,Intel XEON CPU3.04GHz×2,2GB 内存)、一台运行 OnceAS 的服务器(DELL PowerEdge 2600 PC,Intel XEON CPU3.06GHz×4,2GB 内存)、一台运行 DB2 9 数据库的服务器(IBM X255,Intel XEON CPU MP2.5G×8,4GB 内存)以及一台性能异常诊断服务器(Dell OptiPlex GX620,Intel Pentium D CPU 3.4G,2GB 内存),网络连接采用 100Mbps 的以太网交换机.服务器操作系统均采用 Window 2000 Server SP4.

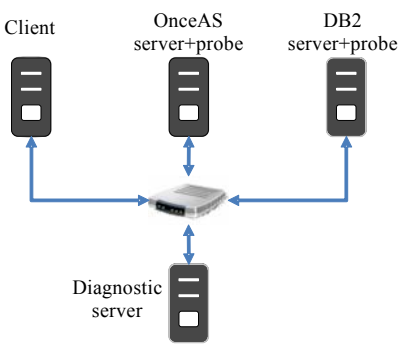


Fig.9 Experimental system

图 9 实验系统

Web 应用服务器和数据库服务器被植入监测器,用于监测运行时信息,监测窗口大小为 2s,特征采样窗口大小为 2min.服务时间的偏差阈值设置为 20%,等待队列使用率的阈值设置为 60%.实验关注在线书店的 14 种应用组件、SQL 语句等逻辑资源使用元素,Web 应用服务器线程池、数据库连接池等逻辑资源控制元素,以及 Web 应用服务器 CPU、数据库服务器 Disk I/O 两种服务器物理资源,其关联关系见表 1.

3.2 实验结果及分析

3.2.1 对变化负载特征的适应能力

图 10~图 12 显示方法有效性测试的实验数据.这 3 个图右侧的 *Y* 轴均表示 TPC-W 的测试负载;*X* 轴均表示

特征采样窗口的时间间隔,同时也表示反馈控制的时间间隔,其中,间隔 1~20 为浏览模式的负载,21~40 为购买模式的负载,41~60 为订购模式的负载.图 10 左侧的 *Y* 轴表示吞吐量,图中曲线表示实验中性能异常的检测和诊断情况以及吞吐量的变化;图 11 左侧的 *Y* 轴表示 Web 应用服务器的线程池和数据库连接池配置的最大值(“WAS-TP”,“WAS-CP”分别表示 Web 应用服务器线程池和数据库连接池的最大配置);图 12 左侧的 *Y* 轴表示 Web 应用服务器 CPU 和数据库服务器 Disk I/O 的资源效用(“WAS-”表示 Web 应用服务器相关度量,“DB-”表示数据库服务器相关度量).

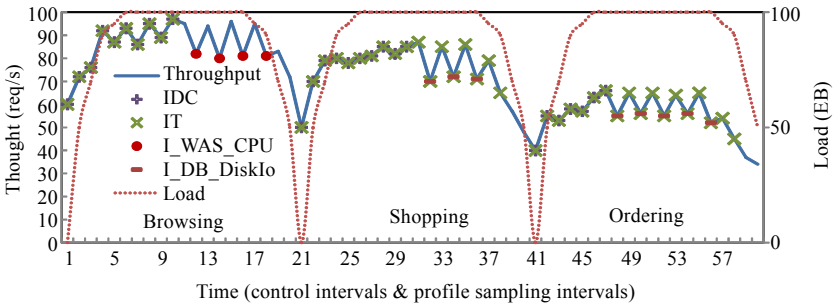


Fig.10 Detected and diagnosed performance anomalies under three transaction mixes

图 10 3 种事务混合下性能异常的检测和诊断情况

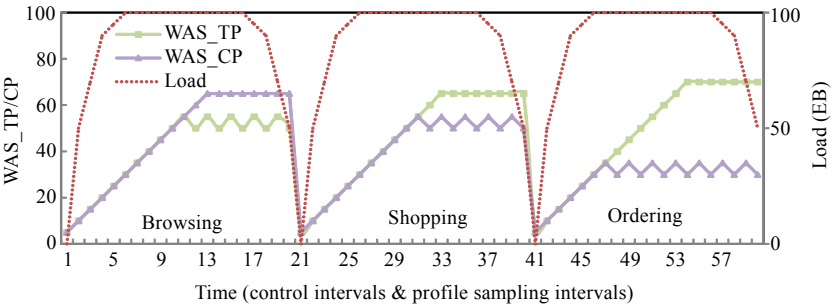


Fig.11 Configurations of thread pool and connection pool in WAS under three transaction mixes

图 11 3 种事务混合下 Web 应用服务器线程池及数据库连接池的配置

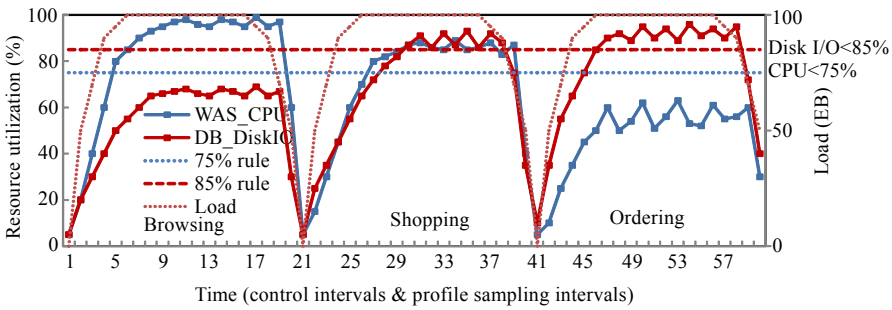


Fig.12 Resource utilization under three transaction mixes

图 12 3 种事务混合下的资源效用

在浏览模式的前 11 个时间间隔内,随着负载的增加,图 11 中 Web 应用服务器线程池和数据库连接池配置的最大值持续增长.由此可见,自优化方法的性能模型评估当前负载下资源没有达到饱和.此时,图 10 中的吞吐量也持续增长,本文方法则检测诊断出线程不足(insufficient thread,简称 IT)、数据库连接不足(insufficient database connection,简称 IDC)等原因引起的逻辑资源瓶颈.从第 12 个时间间隔开始至浏览模式的负载结束,线

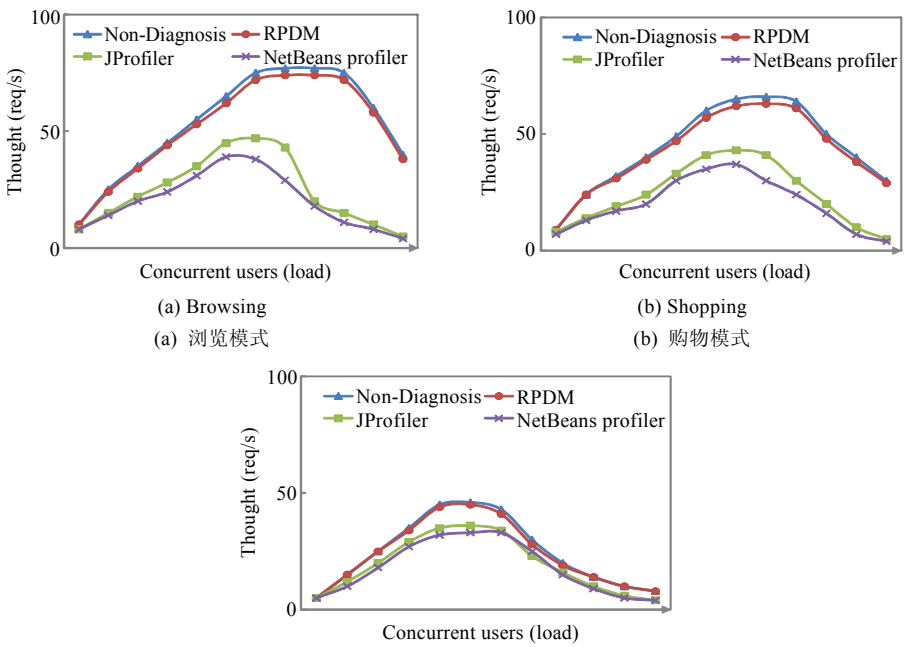
程池配置的最大值持续“振荡”.图 12 显示,此时 Web 应用服务器 CPU 资源效用达到 95%以上.由于自优化方法的性能模型无法预测资源饱和,导致线程池的最大值继续增长,造成 Web 应用服务器 CPU 资源饱和,吞吐量下降.吞吐量下降进一步促使爬山算法降低线程池的最大值.这一过程在高负载下反复持续,造成吞吐量“振荡”.在此过程中,本文方法检测诊断出 Web 应用服务器 CPU 资源不足(insufficient WAS CPU,I_WAS_CPU)引起的物理资源瓶颈.

在购物和订单模式下,吞吐量同样表现出“振荡”现象.与浏览模式不同,购物和订单模式包含更多的数据库写操作.因此在高负载下,当数据库服务器 Disk I/O 成为瓶颈时,数据库连接池配置的最大值持续“振荡”.图 12 显示,在数据库服务器 Disk I/O 效用达到 90%之前,本文方法检测诊断出 IT 和 IDC 等原因引起的逻辑资源瓶颈.在吞吐量和数据库连接池配置最大值出现“振荡”时,方法检测诊断出数据库服务器 Disk I/O 资源不足(insufficient DB DiskIO,简称 I_DB_DiskIO)引起的物理资源瓶颈.

根据常用的CPU,Disk I/O经验法则(rule of thumbs)[33],分别设置CPU效用阈值为 75%,Disk I/O效用阈值为 85%.图 12 显示的实验数据表明,上述资源效用阈值会受事务混合变化的影响而产生假警报.如图 12 所示,如果按照CPU资源效用不大于 75%的经验法则,在浏览、购物模式下会产生大量Web应用服务器CPU资源不足的假警报;而按照Disk I/O资源效用不大于 85%的经验法则,则会在购物、订单模式下产生数据库服务器Disk I/O资源不足的假警报.

3.2.2 系统开销

图 13(a)~图 13(c)分别显示 TPC-W 的 3 种事务混合下系统开销测试的实验数据,其中,*X* 轴均表示负载大小,*Y* 轴均表示吞吐量.图中曲线分别表示未进行性能诊断(non-diagnosis)、使用本文方法(resource-aware performance diagnostic method,简称 RPDM)进行性能诊断以及使用 JProfiler 和 NetBeans Profiler 进行性能诊断这 4 种情况下吞吐量的变化.如图 13 所示,由于本文方法(RPDM)在获取性能特征过程中采用可直接监测的延迟时间、资源效用等性能度量模拟计算服务时间,因此产生的系统开销很低,3 种事务混合下的系统开销均小于 5%,不影响 Web 应用的正常运行;而 JProfiler 和 NetBeans Profiler 等采用字节码注入以及本地代码调用技术的监测工具则产生明显的系统开销,吞吐量下降最高达 87%.



(a) Browsing
(a) 浏览模式

(b) Shopping
(b) 购物模式

(c) Ordering
(c) 订单模式

Fig.13 System overhead under three transaction mixes

图 13 3 种事务混合下的系统开销

4 相关工作

Web 应用及运行环境的动态复杂性促使性能异常的自主分析和诊断成为该领域研究的热点,已有的研究工作主要包括基于规则的阈值、异常检测以及关联分析等方法.

基于规则的阈值方法通过设置应用层、系统层的性能度量阈值来检测性能异常,当性能度量的测量值超出阈值时,发出警报或按照设定策略执行预定动作.由于方法简单,基于阈值的检测方法被商业工具广泛采用[3−5].在该方法中,阈值的大小是静态、依赖经验设置的,并且对于大规模、复杂系统需要人工设置大量阈值.

异常检测方法通常利用事务跟踪方法分析多层Web应用的请求处理过程,文献[6–9]等工作使用端到端(end-to-end)的请求延迟监测方法分析客户端网络延迟造成的性能问题,文献[10,11]等工作关注服务器端的性能问题,将服务器端的请求处理过程进行阶段划分和性能监测.这些工作采用延迟时间作为特征度量,易受系统负载特征变化的影响.Barham等人的工作[34]记录每一请求的通信路径中的操作系统底层事件(如线程上下文切换、I/O中断等),捕捉分布式系统中请求处理的资源消耗,构造请求行为的概率统计模型.由于使用操作系统级接口监测资源消耗,方法存在兼容性问题.另外,该方法缺乏应用级的上下文信息,模型难以反映事务处理过程中应用组件交互等细粒度交互行为,为进一步的问题分析和定位带来困难.一些工作利用字节码注入以及本地代码调用等方法监测资源消耗[31,32],但会造成严重的系统开销.而有研究表明,工业界可容忍的开销仅为 5%[35].

关联分析方法利用机器学习技术和统计学方法分析历史数据,建立性能异常检测和诊断的不确定性推理模型(如贝叶斯网络、决策树等).Cohen等人的工作[36,37]利用模式分类技术挖掘系统历史数据,将系统级度量(CPU,Disk I/O等)与应用层的性能状态相关联.当性能状态违反服务水平协议(service level agreement)时,利用不确定性推理模型(贝叶斯网络)检测与性能状态相关的系统级度量.在此基础上,文献[38,39]等工作通过匹配历史“病例”进行问题诊断.Chen和Kcman等人[40,41]提出一种面向分布式组件的异常检测和诊断框架,通过建立系统“正常”行为的参考模型,检测当前系统状态是否出现组件失效等异常;同时,利用故障诊断决策树定位失效组件.关联分析方法具有较高的准确性,并且对领域知识的要求较低;但机器学习和分析过程比较耗时,除了需要针对系统负载大小、事务混合模式以及运行环境设置等大量变化特征进行训练以外,还需要为诊断提供“病例”数据,造成该方法的实施代价很高.

本文的研究工作借鉴了事务跟踪方法的思想,与Barham等人的工作[34]相比,本文提出的性能特征链利用应用级的上下文信息以及Web应用事务处理过程中可直接监测的性能度量建立,可以描述应用组件交互以及数据库访问等细粒度交互行为的性能特征.从性能特征的度量方法上看,本文通过平台无关的排队模型计算资源服务时间的近似值,与基于操作系统级的方法[34]相比,具有更好的平台兼容性;而与基于字节码注入以及本地代码调用等方法[31,32]相比,本文方法产生的系统开销低,不影响Web应用的正常运行.从性能异常检测和诊断的方法上看,Barham等人的工作[34]未涉及检测和诊断方法的研究,本文的方法建立性能特征参考实例的过程与关联分析方法的学习过程相似,但与文献[36–41]等工作相比,由于本文方法不依赖于历史“病例”,并且在构造性能特征参考实例时不受负载特征变化的影响,因此方法的实施代价较低.

5 结束语

性能诊断是 Web 应用性能保障的关键技术,如何动态地适应 Web 应用负载特征的变化,有效检测和诊断各种资源使用引起的性能异常,是本文研究和解决的问题.

本文提出一种资源敏感的 Web 应用性能诊断方法,针对 Web 应用及运行环境的复杂性问题,在应用层分解事务处理过程,使用组件方法调用、数据库连接、SQL 语句等事务处理的组成元素建立细粒度的资源消耗链式模型.在此基础上,针对动态变化的负载特征,利用服务器端的物理资源实际使用时间(服务时间)度量 Web 应用的性能特征,并给出性能异常检测和诊断的方法.实验结果表明,本文方法可适应系统负载大小、事务混合等变

化特征,诊断各种资源使用引起的性能异常.同时,在获取性能特征过程中,由于采用可直接监测的延迟时间、资源效用等性能度量,本文方法产生的系统开销低于 5%,不影响 Web 应用的正常运行.

下一步的研究工作将利用本文方法改进 Web 应用的性能保障机制,主要包括:(1) 研究适应系统负载特征变化的自优化机制;(2) 差分事务的资源使用时间,提高准入控制和调度机制的有效性.另外,本文提出的方法需要构造资源使用和控制模型,模型的构造过程需要领域专家的参与,下一步我们将研究利用代码分析技术建立程序片段与资源使用的关联关系,进而实现模型的自动构造.

References:

[1] Kephart JO. Research challenges of autonomic computing. In: Proc. of the 27th Int'l Conf. on Software Engineering (ICSE 2005). New York: ACM Press, 2005. 15−22.

[2] Cook B, Babu S, Candea G, Duan S. Toward self-healing multitier services. In: Proc. of the ICDE Workshops (SMDB 2007). Washington: IEEE Computer Society Press, 2007. 424−432.

[3] Mercury diagnostics. http://www.mercury.com/us/products/diagnostics/

[4] IBM Corporation. Tivoli Web management solutions. http://www-01.ibm.com/software/tivoli/

[5] Computer Associates Co. Wily Introscope. http://www.ca.com/us/application-management.aspx

[6] Cherkasova L, Fu Y, Tang W. Vahdat A. Measuring and characterizing end-to-end Internet service performance. ACM/IEEE Trans. on Internet Technology (TOIT), 2003,3(4):347−391.

[7] Olshefski D, Nieh J. Understanding the management of client perceived response time. ACM SIGMETRICS Performance Evaluation Review, 2006,34(1):309−322.

[8] Olshefski D, Nieh J, Agrawal D. Using certes to infer client response time at the Web server. ACM Trans. on Computer Systems (TOCS), 2004,22(1):49−93.

[9] Agarwala S, Schwan K. Sysprof: Online distributed behavior diagnosis through fine-grain system monitoring. In: Proc. of the ICDCS. Washington: IEEE Computer Society Press, 2006.

[10] Nimsoft Co. End user response monitoring. http://www.nimsoft.com/solutions/ete/index.php

[11] Chen MY, Accardi A, Kiciman E, Patterson DA, Fox A, Brewer EA. Path-Based failure and evolution management. In: Proc. of the 2004 Networked Systems Design and Implementation. Berkeley: UC Berkeley, 2004. 309−322.

[12] The Transaction Processing Council (TPC). TPC-W. 2002. http://www.tpc.org/tpcw

[13] Chen H, Mohapatra P. Session-Based overload control in QoS-aware Web servers. In: Proc. of the IEEE INFOCOM 2002. New York: ACM Press, 2002. 516−524.

[14] Chen X, Chen H, Mohapatra P. An admission control scheme for predictable server response time for Web accesses. In: Proc. of the 10th World Wide Web Conf. New York: ACM Press, 2001. 545−554.

[15] Abdelzaher TF, Lu C. Modeling and performance control of Internet servers. In: Proc. of the 39th IEEE Conf. on Decision and Control. Washington: IEEE Computer Society Press, 2000. 2234−2239.

[16] Diao Y, Gandhi N, Hellerstein J, Parekh S, Tilbury D. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In: Proc. of the Network Operations and Management Symp. New York: ACM Press, 2002. 219−234.

[17] Fan GC, Zhong H, Huang T, Feng YL. A survey on Web application servers. Journal of Software, 2003,14(10):1728−1739 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1728.htm

[18] Barghouti NS, Kaiser GE. Concurrency control in advanced database applications. ACM Computing Surveys (CSUR), 1991,23(3): 269−317.

[19] Agrawal R, Carey MJ, Livny M. Concurrency control performance modeling: Alternatives and implications. ACM Trans. on Database Systems (TODS), 1987,12(4):609−654.

[20] Welsh M, Culler D, Brewer E. SEDA: An architecture for well-conditioned, scalable Internet services. In: Proc. of the 18th Symp. on Operating Systems Principles (SOSP). New York: ACM Press, 2001. 230−243.

[21] Fleury M, Reverbel F. The JBoss extensible server. In: Endler M, Schmidt DC, eds. Proc. of the ACM/IFIP/USENIX Int'l Middleware Conf. (Middleware 2003). LNCS 2672, Riode Janeiro: Springer-Verlag, 2003. 344−373.

[22] Harkema M, Quartel D, van der Mei R, Gijsen B. A Java performance monitoring tool. In: OOPSLA'98: Proc. of the 13th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 1998. 21−35.

[23] Sun Microsystems. Sun studio 11: Performance analyzer. 2005. http://docs.sun.com/app/docs/doc/819-3687

[24] Lazowska ED, Zahorjan J, Graham GS, Sevcik KC. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Upper Saddle River: Prentice-Hall, Inc., 1984.

[25] Oracle Co. Oracle enterprise manager. 2009. http://www.oracle.com/technology/products/oem/pdf/ds_as_dp.pdf

[26] Casella G, Berger RL. Statistical Inference. Pacific Grove: Wadsworth Group, 2002.
[27] Huang T, Chen NJ, Wei J, Zhang WB, Zhang Y. OnceAS/Q: A QoS-enabled Web application server. Journal of Software, 2004, 15(12):1787−1799 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1787.htm
[28] Menasce DA. TPC-W: A benchmark for e-commerce. IEEE Internet Computing, 2002,6(3):83−87.
[29] Menascé DA. Automatic QoS control. IEEE Internet Computing, 2003,7(1):92−95.
[30] Menascé DA, Dodge R, Barbará D. Preserving QoS of e-commerce sites through self-tuning: A performance model approach. In: Wellman MP, Shoham Y, eds. Proc. of the 3rd ACM Conf. on Electronic Commerce. New York: ACM Press, 2001. 224−234.
[31] EJ Technologies. JProfiler ej-technologies. http://www.ej-technologies.com/products/jprofiler/overview.html
[32] NetBeans Community. NetBeans profiler, Version 5.5. 2007. http://www.netbeans.org/products/profiler/index.html
[33] Murray H, Engineer S, Associates I. Rules-of-Thumb for monitoring windows NT/2000 and domino statistics. http://www.ibm.com/developerworks/lotus/library/ls-Rules_WinNT2000/
[34] Barham P, Donnelly A, Isaacs R, Mortier R. Using magpie for request extraction and workload modeling. In: Proc. of the 6th Symp. on Operating Systems Design and Implementation (OSDI 2004). Berkeley: USENIX Association, 2004. 18.
[35] Bodden E, Hendren LJ, Lam P, Lhoták O, Naeem NA. Collaborative runtime verification with tracematches. In: Proc. of the 7th Int'l Workshop on Runtime Verification (RV). LNCS 4839, Berlin: Springer-Verlag, 2007. 22−37.
[36] Cohen I, Goldszmidt M, Kelly T, Symons J, Chase J. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In: Proc. of the 6th Symp. on Operating Systems Design and Implementation (OSDI 2004). Berkeley: USENIX Association, 2004. 16.
[37] Zhang S, Cohen I, Symons J, Fox A. Ensembles of models for automated diagnosis of system performance problems. In: Proc. of the 2005 Int'l Conf. on Dependable Systems and Networks (DSN 2005). Washington: IEEE Computer Society Press, 2005. 644−653.
[38] Cohen I, Zhang S, Goldszmidt M, Symons J, Kelly T, Fox A. Capturing, indexing, clustering, and retrieving system history. In: Proc. of the 20th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2005. 105−118.
[39] Kelly T. Detecting performance anomalies in global applications. In: Proc. of the 2nd Workshop on Real, Large Distributed Systems (WORLDS 2005). Berkeley: USENIX Association, 2005. 42−47.
[40] Chen M, Kcman E, Fratkin E, Brewer E, Fox A. Pinpoint: Problem determination in large, dynamic Internet services. In: Proc. of the Symp. on Dependable Networks and Systems (IPDS Track). Washington: IEEE Computer Society Press, 2002. 595−604.
[41] Kiciman E, Fox A. Detecting application-level failures in component-based Internet services. IEEE Trans. on Neural Networks, 2005,16(5):1027−1041.

附中文参考文献:

[17] 范国闯,钟华,黄涛,冯玉琳.Web 应用服务器研究综述.软件学报,2003,14(10):1728−1739. http://www.jos.org.cn/1000-9825/14/1728.htm
[27] 黄涛,陈宁江,魏峻,张文博,张勇.OnceAS/Q:一个面向 QoS 的 Web 应用服务器.软件学报,2004,15(12):1787−1799. http://www.jos.org.cn/1000-9825/15/1787.htm

王伟(1982－),男,山西运城人,博士生,主要研究领域为网络分布计算,软件工程.

钟华(1971－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布计算,软件工程.

张文博(1976－),男,博士,高级工程师,CCF 高级会员,主要研究领域为网络分布计算,软件工程.

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程,网络分布计算.

魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布计算,软件工程.