软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(4):786−801 [doi: 10.3724/SP.J.1001.2012.04027]

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

面向性能剖析的 Web 应用自动性能建模方法*

黄 翔[1,2,3,4+], 王 伟[1,4], 张文博[1], 魏 峻[1,2], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术研究开发中心,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)

[4](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

Automatic Performance Modeling Approach to Performance Profiling of Web Applications

HUANG Xiang[1,2,3,4+], WANG Wei[1,4], ZHANG Wen-Bo[1], WEI Jun[1,2], HUANG Tao[1,2]

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

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

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

[4](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

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

Huang X, Wang W, Zhang WB, Wei J, Huang T. Automatic performance modeling approach to performance profiling of Web applications. *Journal of Software*, 2012,23(4):786−801. http://www.jos.org.cn/1000-9825/4027.htm

Abstract: This paper proposes an automatic performance modeling approach for performance profiling of Web applications. In addition, the study proposes an automatic approach to build performance model. Both the user behaviors and their corresponding internal service relations are modeled, and the CPU time consumed by each service is also obtained through Kalman filter, which can "absorb" some level of noise in real-world data. Experimental results show that this approach can adapt to the change in both the inner and outer environments of Web applications and provide valuable information for capacity planning and bottleneck detection.

Key words: Web application; performance profiling; performance modeling; Kalman filter

摘 要: 提出了一种面向性能剖析的Web应用自动性能建模方法.该方法考虑了用户行为与系统中不同服务之间的关联,动态地构造与应用实际状态相符的性能模型,并利用 Kalman 滤波所具备的过滤"噪声"和适应变化的特性,精确估算各服务所需 CPU 时间.实验结果表明,该方法可以适应 Web 应用内、外部环境的变化,分析结果可为瓶颈定位和容量规划等性能保障技术提供高质量数据.

关键词: Web 应用;性能剖析;性能建模;Kalman 滤波

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(61100068); 国家重点基础研究发展计划(973)(2009CB320704); 核高基重大专项(2009ZX01043-001-05, 2009ZX01043-003-001, 2010ZX01045-001-010-4); 武汉大学软件工程国家重点实验室开放基金(SKLSE20100821)

收稿时间: 2010-10-11; 修改时间: 2011-01-20; 定稿时间: 2011-04-02

Web 系统已成为当前主流的网络应用形式,其性能作为衡量用户满意度的一个重要属性受到了越来越多的关注,特别是对一些关键的 Web 系统(如电子商务、网上支付等),性能问题会导致客户流失、收益受损等严重后果.但是,随着发布与管理模式的转变[1,2],性能保障不再是单纯的以满足服务质量约束(service level agreement)为目标,而更需要兼顾资源利用率的提高.与此同时,Web 系统面临的环境也变得日益开放和动态,导致系统管理和控制难度大为增加[3,4].因此,在开放和动态的环境下,通过提高资源利用率保障系统的性能已成为一种挑战.

资源调度是提高资源利用率的主要途径[1,2],因此对资源利用情况进行剖析和评估就变得至关重要[5,6].传统的性能剖析方法通常基于探针技术监测应用的行为,但这类方法或者因为无法准确监测 Web 应用这类短任务的 CPU 消耗[7,8],或者因为会引入大量的性能开销[9,10](30%以上),从而无法适用于在线系统的性能剖析.此外,这类方法也很难评估系统未来的性能表现.

另一方面,由于资源的变化并不能线性地映射到性能的改变[5,6],所以对于系统未来性能的评估,通常采用基于性能模型的方法.目前,一些建模方法[11,12]引入了对用户行为的分析.相比于传统的单一负载的性能模型,这类方法可以更好地描述负载的变化[13],但仍不能解释应用内部的执行行为,难以用于剖析系统内部的性能表现.

在已有性能建模方法的基础上,本文进一步关注了应用行为和用户行为之间的关联.利用日志信息,在系统正常运行的情况下自动构造出体现应用内部执行特征的细粒度的性能模型,以解决在线 Web 应用性能剖析和评估的问题.

自动构造此类模型最大的挑战不仅在于 Web 应用的规模,而且还在于其动态和异构性.首先,负载会随着用户行为的改变而不断变化.比如,对于一个在线商城系统,当有新商品发布时,用户行为很可能从浏览型改变为购物型,从而使系统内部执行状况发生改变,影响系统性能;其次,服务通常会隐藏其内部实现,其执行所需 CPU 时间不仅与平台密切相关,而且还具有时变性的特点.因而,已有性能建模方法采用的静态参数(CPU 时间)设置显然不能适应系统状态的变化和平台的迁移,而在系统运行阶段,又很难精确地监测 CPU 时间.

为了解决上述问题,本文以广泛使用的分层排队网(layered queue network)[14−16]为基础,利用 Kalman 滤波动态估算模型参数,提出了一种自动的性能建模方法.Kalman 滤波不仅可以很好地适应参数变化,还可以过滤监测噪声,为应对软、硬件环境变化提供了支持.此外,由于用户请求的执行轨迹也被刻画在本模型当中,所以用户行为的变化可以体现为用户请求调用执行轨迹的频率变化.因而,通过日志信息对性能模型进行动态更新,可使性能模型适应负载和软、硬件环境的变化;最后,本文以 TPC-W 基准测试为例[17,18],在模拟负载类型和软、硬件环境变化的情况下对比性能模型的分析结果与实际测量结果,验证本文方法的准确性与适应性.

本文第 1 节给出我们的方法的概述及所关注的应用类型.第 2 节给出具体的建模方法.第 3 节给出工具实现及方法验证.第 4 节给出基于模型的性能剖析案例.第 5 节讨论方法的局限和优势.第 6 节对相关工作进行比较.最后,第 7 节总结本文的工作与主要贡献.

1 方法概述

本文所讨论的性能剖析问题针对于 Web 系统的在线运行阶段.此时,系统已经部署于特定平台,并向外提供服务,因而可以通过收集日志信息获取系统运行的状态.

Web 应用的性能首先会受到负载的并发规模(并发用户数)与混合比例(用户行为)的影响,比如 100 个浏览型的并发用户与 100 个订单型的并发用户所需的资源显然是不同的.为了支持负载规模与混合的特性,本文采用用户行为模型图(customer behavior model graph,简称 CBMG)[19,20]来刻画负载.

影响系统性能的另一个因素是应用自身的资源消耗.对于 Web 系统这类事务性的应用,一个请求通常会由若干服务协作共同完成(如界面渲染、数据处理与存储等等),而这些服务又会部署在不同的层上.对于简单的情况,请求由 T_i 层上的服务处理后,会转发到 T_{i+1} 层上的服务,直到最后的 T_n 层处理结束,然后结果逐步由 T_{n-1}, T_{n-2},…返回给用户.对于更为普通的情况,位于 T_i 层的服务可能向多个位于 T_{i+1} 层的服务发出请求.整个处理流程称为一个事务,其中每个服务都会有各自的资源消耗,它们共同构造应用总体上的资源开销.后文将同一页面

的不同请求称为一个事务,因而同一个事务可能具有多条执行路径.

本文将事务型的 Web 应用(如 JavaEE 应用[21])抽象为如图 1 所示的形式,以描述负载和应用的特征.图 1 的左侧是采用 CBMG 描述的负载,右侧则是采用执行图(execution graph,简称 EG)描述的各事务的执行流程.CBMG 中的每一个节点表示对某个事务的一次访问.EG 则由服务和服务间的关联组成,起始节点是直接接收用户请求的服务.每个服务含有一个描述 CPU 消耗(服务时间)的属性.服务时间的实际值与服务所在的服务器相关,但因为这种部署关系非常直接,所以图中并未对其加以描述.

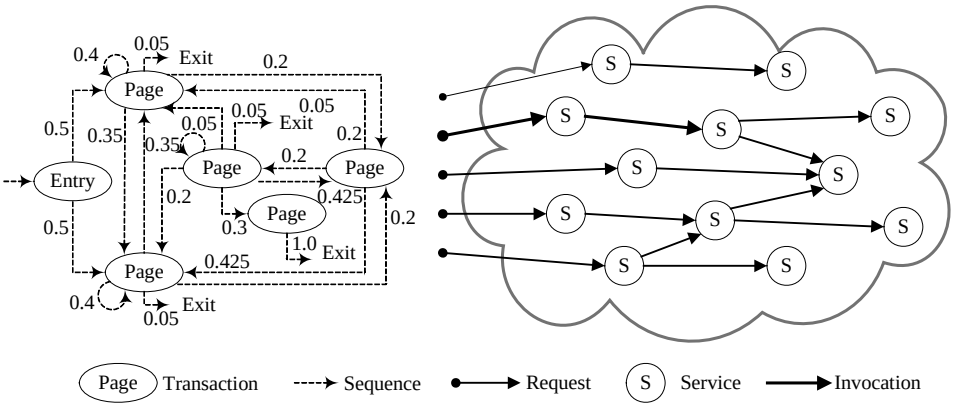


Fig.1 Performance characteristics of Web system

图 1 Web 系统性能特征

2 模型构造

在运行阶段自动构造上述模型存在如下几个挑战:首先,因为用户行为是有状态的,即下一个请求依赖于上一个请求的状态,所以使得排队论模型的有效性受到限制.比如,对于 CBMG 用户行为模型,其中的循环不能有效地转化为 LQN 这类非循环的模型;其次,对于在线的应用,执行图和服务时间都很难获取,特别是 CPU 时间,它不仅是平台相关的,而且具有时变性的特点.现有的监测技术或者无法准确获取 CPU 时间,或者会带来巨大的额外开销.针对上述问题,本节将从负载、执行图和服务时间这 3 个方面讨论性能模型的自动构造方法.

2.1 负载模型构造

Web 系统的负载适合使用基于会话的方式来描述[20,22],由到达模式和访问模式两部分组成.到达模式是指新用户的达到方式,分为封闭和开放两种基本类型[23],主要影响系统并发规模.访问模式是指用户使用系统的行为,通常描述为用户行为图(CBMG)[19],主要影响并发请求中不同类型请求的混合比例.

CBMG 可以表示为一个 $P=[p_{i,j}]$的 $n\times n$ 矩阵.其中,$p_{i,j}$ 表示在一个会话里,在事务 i 之后请求事务 j 的概率,$0\leqslant i,j\leqslant N+1$.其中,事务 0 表示会话开始,事务 N+1 表示会话终止.CBMG 的本质是一种马尔可夫链,表示用户每一步访问状态变化的概率.严格地说,CBMG 是一种带有吸收状态的马尔可夫链.也就是说,事务总会从事务 0 开始,然后到事务 N+1 结束,因而可以推导出一个会话里各事务被访问到的平均次数.

设 V 表示 CBMG 中各事务被访问到的平均次数,v_i 表示事务 i 在一次会话中被访问到的次数.如果假设 v_0 的次数是 1,即会话开始的次数为 1,那么各个服务被访问的次数可定义为公式(1)的形式,即各服务被访问的次数等于其前驱服务被访问次数与访问该服务概率的乘积.

$$v_j=\sum_{i=0}^{n+1}v_i\times p_{i,j},\ j=1,...,n+1 \tag{1}$$

公式(1)可以写为如下矩阵形式:

$$\vec{V}-\vec{1}=\vec{V}\times p \tag{2}$$

其中,$\vec{1}=(1,0,...,0)$, $p_{n+1,k}=0,\forall k=0,\ldots,n+1$,因为会话的开始和结束事件必然存在,且会话结束后不会再访问其他

服务.于是,求解公式(2)对应的线性方程组,即可获得向量 V.

CBMG 附带的另一个属性是思考时间(think time),可由矩阵 $D=[d_{i,j}]$表示,其中,$d_{i,j}$ 表示用户在接收到事务 i 的响应后到发起事务 j 的时间间隔.设 W 表示等待时间的集合,w_i 为收到事务 i 响应之后的平均等待时间,总思考时间可以表示为 $\sum_{i=1}^{n} w_i$,其中,w_i 表示为如下形式:

$$w_i = \sum_{j=1}^{n+1} v_i \times d_{i,j},\ i = 1,...,n \tag{3}$$

V 与 P 中单个服务被访问的概率是一样的,但不存在环,所以可以用 LQN 建模.图 2 给出了 LQN 模板.EB (emulated browser)用于模拟用户行为,对应于 LQN 模型中模拟用户的特殊“任务(task)”,可以采用开放和封闭两种方式生成负载[24].EB 以 V 中的权重访问与各个事务对应的起始“任务”(从 T_1 到 T_n,“任务”的构造方法将在第 2.2 节中加以介绍).不过,并不存在 T_0 和 T_{n+1} 这两个起始“任务”,因为它们只是会话开始和结束的标志.

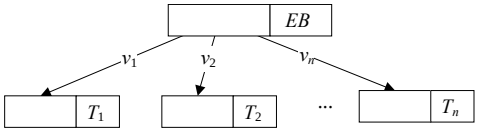


Fig.2 An example of workload model

图 2 负载模型示例

2.2 执行图构造

事务通常存在多条执行路径,不同执行路径上的概率并不一样,有些路径会被经常调用,而有些则可能很少被调用.本节将介绍如何根据日志中收集的轨迹信息构造事务的带权执行图.本节所需的日志信息采用开源的监控框架 InfraRED[25]进行收集.其他研究方案[13,26]也可用于轨迹的监测,但利用哪种监控技术更优并不属于本文的讨论范围.

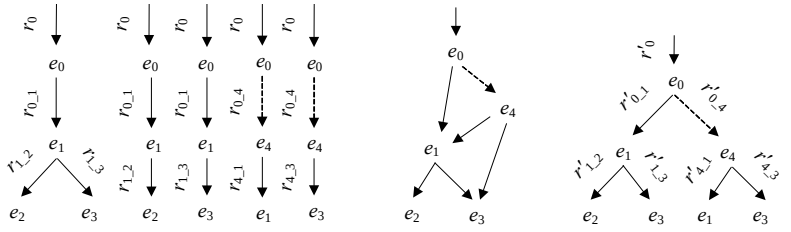
图 3(a)给出了对一个事务进行监测所得到的轨迹示例,该服务具有 5 条不同的执行轨迹.图中节点代表服务,边代表了服务之间的调用关系,实线表示同步请求,虚线表示异步请求,边上的权值表示调用的次数.比如,e_0 表示了该事务的起始任务,r_0 表示用户请求 e_0 的次数,r_{0_1} 表示 e_0 调用 e_1 的次数.

执行图描述的是系统总体上的执行特征,可以通过统计一段时期内的执行轨迹近似获得.但如果只是简单地按节点统计执行轨迹,则会导致结构上的不一致.如图 3(b)中 $e_0 \to e_4 \to e_1 \to e_3$ 的执行轨迹并不存在.导致不一致的原因在于 $e_4 \to e_1$ 与 $e_0 \to e_1$ 并不对等,如果统计过程不作区分,则会使路径交叉,出现实际并不存在的路径.

为了解决这个问题,本文定义了对等节点的概念.如果节点α和β是对等的,那么满足:

- α和β表示同一个服务,且
- 或者α的父节点和β的父节点是对等节点,且父节点到α和β的请求类型相同;
- 或者α=β且α和β的父节点为空.

本文用 $E(x)$表示节点 x 的对等类.如果两个对等的节点有$\alpha \to \beta$,那么在合并后的执行图上有 $E(\alpha) \to E(\beta)$.图 3(c)描述了合并后的执行图,其中保留了 $e_0 \to e_1 \to e_3$ 和 $e_0 \to e_4 \to e_1$ 的调用关系,所以不会导致不一致.合并过程会对对等节点的调用次数进行累加,用于计算平均请求概率,如 r'_{0_1} 即为各链中 r_{0_1} 的累加和.



(a) 服务执行轨迹 (b) 合并执行轨迹 (c) 服务的执行图

Fig.3

图 3

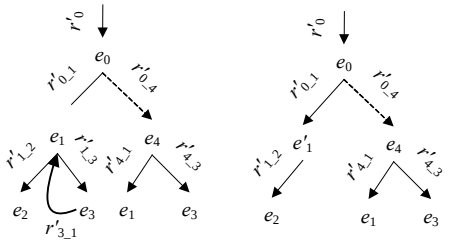
执行图是一个有向图,如果不考虑循环,则其宽度会受到服务总数的制约,不会超出程序中具有的服务总数.但是如果考虑到服务间的递归调用,则可能造成执行路径具有无限的深度.

为了发现循环,本文增加了判定循环的规则.如果节点α和β是对等的且在循环中,那么满足:

- 如果存在节点 v 是与α和β相同的服务,且 v 是α和β的祖先节点(节点可以是其自身的祖先).

该定义保证循环嵌套中的对等实例可以被识别出来.比如,如果存在一条执行轨迹 $e_0\rightarrow e_1\rightarrow e_3\rightarrow e_1\rightarrow e_3$,就会形成如图 4(a)所示的结构.但由于 LQN 是非循环的分层网状结构,所以需要消除循环嵌套.因此,本文将循环中的节点合并为一个节点以消除循环嵌套.

图 4(b)显示了合并后的结果.但消除循环会使多个服务的影响合并到一个服务中(如 e_1' 包含了 e_1 和 e_3),或者使一个服务的影响分散多个服务(如 e_1' 和 e_1).如果性能瓶颈发生在这一部分,则还需手工地在合并后的节点中分析原因.但在实际应用中,通常会采用层次化设计,即下层组件很少调用上层组件,所以循环嵌套并不常见.因此,为了降低循环对于性能影响因素的影响,本文引入了一个阈值θ(可以取大于 0 的任意值),只有当递归深度大于θ时才合并;否则,根据对等节点的定义,它们会被视为不对等节点,从而不影响对性能问题的分析.



(a) 循环调用 (b) 合并循环

Fig.4

图 4

执行图可以直接转换为 LQN 模型.节点对应 LQN 模型中“任务”的一个“入口(entry)”,入口请求的次数(权重)可以通过节点被调用的次数与其父节点调用的次数的比值来获得,比如 $R_{0\_1}=r'_{0\_1}/r'_0$.本文并不详细介绍转换算法,因为它比较直接.图 5 给出了图 3(c)和图 4(b)所对应的 LQN 模型,T_a 表示该服务的起始“任务”.图 5(a)中 e_1 被分成了两个入口(为了直观描述,图中这两个入口用了相同的标识),但因 LQN 的隐式队列位于任务中,所以不会改变入口的排队方式.另外,如果同一任务在不同服务的执行图中出现,那么将它们合并为一个任务,但并不合并入口,否则也会引起不一致问题.

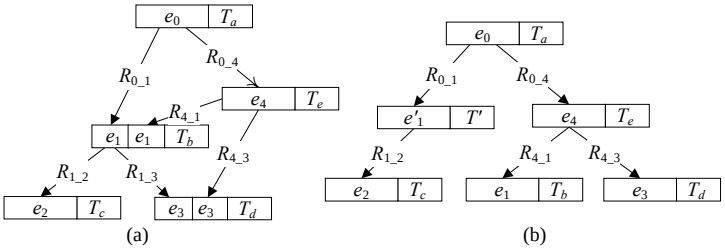


Fig.5 LQN model converted from service’s execution graph

图 5 服务的执行图转换后的 LQN 模型

2.3 服务时间估算

服务时间的获取是性能模型构造的一个难点,传统的基于回归分析的方法[27]又很难适用于这种具有变化性的场景.为了解决上述问题,本节提出了一种基于 Kalman 滤波的方法[28]以估算服务时间.Kalman 滤波最大的特点是可以用一种近似最优的方式基于可测量的值估算不可测量的值,并且可以随着新的测量值的到来更新

之前的估计值.在过去 40 多年中,Kalman 滤波在自动控制和辅助导航领域得到了广泛的使用和研究[29−31].

Kalman 滤波提供了一种在离散时间点估算不可观测状态 X 的通用方法.第 k 时刻,状态 X_k 可以定义为一个线性随机差分方程:

$$X_k=AX_{k-1}+Bu_{k-1}+w_k \tag{4.a}$$

第 k 时刻测量值 Z_k 定义为

$$Z_k=HX_k+v_k \tag{4.b}$$

其中,A 是从 k−1 时刻到 k 时刻状态转换矩阵;u_{k-1} 是可选的控制参数;B 是与控制相关的矩阵;w_{k-1} 为测量误差,其协方差矩阵为 Q_{k-1};H 是 X_k 到 Z_k 的转换矩阵;v_k 是测量误差,其协方差矩阵为 R_k.

本文将公式(4.a)和公式(4.b)作如下映射:

$$x_k=x_{k-1}+w_{k-1} \tag{5.a}$$

$$z_k=\sum_{i=1}^{n}t_i\cdot x_{i,k}+v_k \tag{5.b}$$

其中,$X_k=[x_{1,k},x_{2,k},\ldots,x_{n,k}]$,表示 k 时刻各服务的服务时间;z_k 为总 CPU 利用率;t 为各服务的吞吐率.根据 CPU 利用率法则,有公式(5.b),即总 CPU 利用率等于各服务吞吐率与服务时间乘积的累加和.所以 H 可以如下定义:

$$H_k=[t_1,t_2,\ldots,t_n] \tag{5.c}$$

Kalman 算法还需要一个初始值 $\hat{X}_0$ 和 P_0,其迭代过程如下所示:

1. 使用 w_{k-1}=0 更新 X 的状态

$$\hat{X}_k^-=\hat{X}_{k-1} \tag{6.a}$$

2. 更新协方差矩阵 P_k^-

$$P_k^-=P_{k-1}+Q_k \tag{6.b}$$

3. 计算 Kalman 增益

$$K_k=P_k^-H_k^T(H_kP_k^-H_k^T+R_k)^{-1} \tag{6.c}$$

4. 修正 X 的状态

$$\hat{X}_k=\hat{X}_k^-+K_k(z_k-H_k\hat{X}_k^-) \tag{6.d}$$

5. 修正协方差矩阵 P_k

$$P_k=(I-K_kH_k)P_k^- \tag{6.e}$$

迭代过程中,第 4 步修正 X 的状态是估算值更新的关键.该公式可以简化为 $X_{new}=X_{old}+K\cdot e$ 的形式,即 Kalman 增益 K 可以看作修正 X 的权重矩阵,利用误差 e 和相应的权重修正 X_{old} 的数据.

初始值 $\hat{X}_0$ 和 P_0 对 Kalman 滤波计算影响很小,可以设置为任何有意义的值.本文根据排队论的定义[32],将 $\hat{X}_0$ 设为 $x_{i,0}=rt_{i,0}(1-z_0)$.其中,$rt_{i,0}$ 为服务 i 的响应时间,即服务时间等于响应时间乘以 CPU 空闲率.另外,因为各入口的服务时间是独立的,所以 P_0 是对角矩阵,设为 $P_0=diag((x_{1,0})^2,(x_{2,0})^2,\ldots,(x_{n,0})^2)$,即取 X 初始值的平方.

每次迭代需要确定 H_k,Q_k 和 R_k 这 3 个矩阵.首先,H_k 可以通过监测技术获得,即各个服务的吞吐率;其次,Q_k 表示每一次迭代中 X 变化的协方差矩阵.对于在线的系统通常无法获得 Q_k,只能估计变化范围.如果 Q_k 太大,将导致估算结果抖动过大,太小又会使结果变化细微,体现不出服务时间的波动.应对策略是将 Q_k 设置为对角矩阵,且对角线元素为迭代过程中 X 变化的最大值的平方,即 $Q_k=diag(\xi_1,\xi_2,\ldots,\xi_n)$,其中,$\xi_i=\max_{1\leqslant j\leqslant k}((x_{i,j}-x_{i,j-1})^2)$;最后,$R_k$ 为测量值的误差,即总 CPU 利用率的测量误差.本文假定 CPU 总利用率的测量值误差很小,足以值得信赖,因此设 R_k=0.也可以通过先期实验统计 R 的值,将其设为一个常量.

本方法一次迭代的复杂度为 $O(m^3)$,m 为一台应用服务器上部署的服务总数.因为观测值 z 定义为 CPU 的总利用率,而 H 和 K 是向量,所以公式(6.c)中的求逆退化为对一个数求倒数.剩下复杂度最高的计算是公式(6.e)所对应的步骤,其复杂度为 $O(m^3)$.又因为部署在一台服务器上的服务量级不会特别大,所以本方法可以有效地用于计算运行态 Web 系统各服务的服务时间.

3 实现与验证

本节主要介绍原型工具的实现和在此基础上展开的实验验证.

3.1 工具实现

本方法的自动性体现在无需人工干预,可自动地从日志信息中构造出LQN模型.原型工具已在Trustie网站开源,可在“基于模型的Web应用性能剖析工具”项目(MBP)[33]中下载.图6给出了本工具的架构.

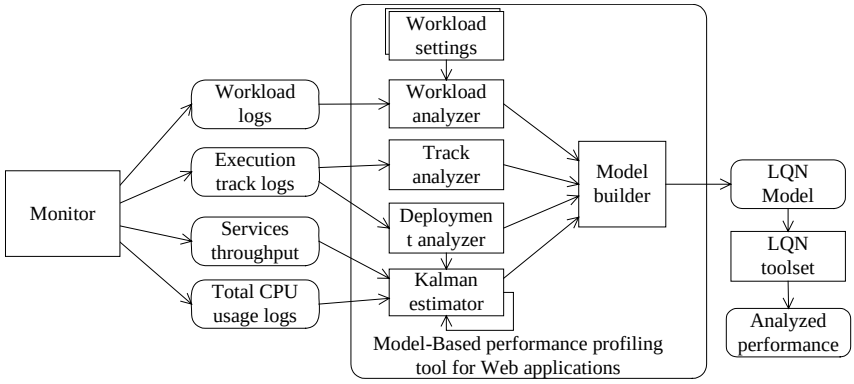


Fig.6 Architecture of model based performance profiling tools for Web applications

图6 基于模型的Web应用性能剖析工具架构

MBP的输入是监测工具收集的日志信息,包括负载、执行轨迹、服务吞吐率和总CPU利用率,可通过目前主流的监测框架获取,监测技术与日志收集并不在本工具实现范围之内.

MBP的核心模块包括负载构造器、轨迹分析器和Kalman估算器.它们分别实现了第2节模型构造中介绍的3种主要算法.其中,负载构造器与负载配置接口关联.负载配置接口的作用是为系统管理员提供一个人机接口,以便管理员指定负载并发规模和混合比例.Kalman 估算器上有一个自引用环,表示它会把上一次估算的结果作为下一次估算的输入.此外,Kalman 估算器的输入除了日志信息以外,还包括了部署分析工具的输出.部署分析工具的作用是根据日志来源服务器的不同,确定服务所在的宿主服务器,以便确定服务时间与CPU总利用率的关系.

上述模块处理的结果会统一输入到模型构造器这个模块,该模块再将负载、轨迹、部署位置和服务时间综合起来生产一个LQN模型.由于合成过程并没有复杂的转化,所以本文并未详细描述合成算法.生产的LQN模型经由LQN工具[24]求解,即可获得性能分析结果.

3.2 实验设置

本文选用TPC-W基准测试[17]作为实验基础.TPC-W是一个被广泛使用的模拟在线商城的Web应用标准测试规范.实验对象选用了中国科学院软件研究所软件工程技术研究开发中心研发的符合 TPC-W 规范的Bench4Q[18]基准测试套件,图7给出了该实验系统的拓扑结构,软、硬件环境则在表1中给出.

Table 1 Software and hardware environment

表1 软、硬件环境

Server	Processor	RAM (GB)	No.
Client (emulated-browsers)	Pentium IV/1.8GHz	2	3
Application server (*S*1)-Tomcat6.0	Pentium IV/2.0GHz	3	1
Application server (*S*2)-Jetty6.1	Pentium IV/2.8GHz	3	1
Database server-MySql-5.1	Pentium IV/3.9GHz	3	1

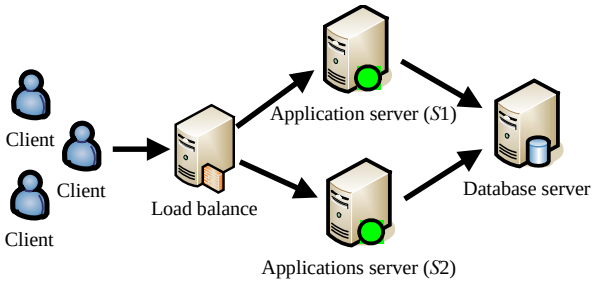


Fig.7 Architecture for the experiment

图 7 实验架构

TPC-W 的规范定义由模拟浏览器 EB 模拟用户会话.同时,TPC-W 定义了 14 种不同类型的事务(见表 2),分为浏览和订单两种类型.根据这两类服务比重的不同,TPC-W 定义了 3 种典型的混合方式.浏览型:95%的浏览,5%的订单;购物型:80%的浏览,20%的订单;订单型:50%的浏览,50%的订单.在本实验中,数据库采用默认设置,即 10 000 件商品和 1 440 000 个用户.

Table 2 Fourteen basic transactions in TPC-W

表 2 TPC-W 中 14 种基本事务

Browsing type	Ordering type
Home	Admin confirm
Best sellers	Admin request
New products	Buy confirm
Product detail	Buy request
Search request	Cart
Search results	Customer registration
	Order display
	Order inquiry

根据 TPC-W 规范的定义,EB 数(即并发用户数)保持恒定.但本文为了更好地模拟实际系统的负载变化,负载按图 8 给出的方式生成.整个实验历时 9 个小时(不包括预热和停止时间),负载的混合比例以小时为单位,周期性地轮换(B=浏览,S=购物,O=订单).此外,为了进一步验证 Kalman 滤波对于软件更新所带来影响的适应能力,在第 7 个小时,人为地给 Home 事务增加浮点计算,使应用服务器上计算开销增加,且在第 8 个小时恢复到原始状态.另外,由于平台(包括应用服务器或硬件平台,详见表 1)切换频率相对较慢,所以变化后的计算可视为一次全新的计算.因此,本实验并未设计运行时平台的切换,而是使用不同的平台配置验证 Kalman 滤波对不同平台的适应能力.

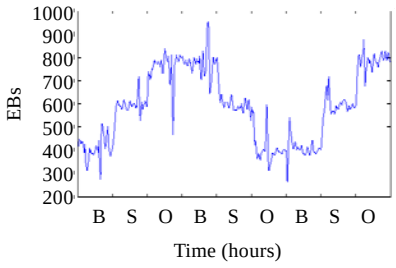


Fig.8 Workload generated for the experiment

图 8 负载生产情况

3.3 服务时间估算验证

服务时间是性能模型预测准确性的一个关键,本节就 Kalman 滤波估算的准确性进行讨论.首先,通过比较

估算值与实际服务时间的差异,讨论估算的准确性以及适应变化的能力;其次,与回归方法进行比较,讨论Kalman滤波对于估算具有噪声和时变性特点的服务时间所具有的优势.

3.3.1 服务时间比较

本节利用JProbe[34]收集服务时间,分析其准确性和适用变化的能力.考虑到JProbe会引入大量的额外开销,本文引入了剖析比例因子(profiling ratio factor,简称PRF)对收集到的数据进行处理.PRF基于一个假设,即额外开销会等概率地分布在整个测试之中,因而可以通过比较开启或关闭JProbe时的总CPU利用率来获得.

由于JProbe引入了大量的开销,如果收集服务时间时仍按图8所示的方式生产负载,则会使系统过于饱和,严重影响性能表现.本节通过4组对比实验,在低负载情况下收集平均服务时间,并换算出PRF.前两组实验启动JProbe,在并发量为20的情况下运行半小时,收集各个服务的服务时间,并记录总CPU利用率.其中,第2组实验增加Home事务浮点计算量.而后两组实验则关闭JProbe,其他设置与前两组相同.

通过比较,PRF确定为7.9.图9给出了Home事务的渲染服务在两台不同应用服务器上的实测值(JProbe收集的平均值)与估算值的比较.其他服务与之类似,限于篇幅,并未详细展示.从图9可以看出,预测结果能够持续地稳定在实测值附近,平均误差约为4.2%.且在算法起始阶段以及服务时间发生变化之后,服务时间经过2~3步迭代即可收敛于实测值.从图中还可以看出,虽然“应用服务器1”和“应用服务器2”采用了不同的平台配置(详见表1),但是对Klaman滤波的估算并未产生太大的影响.因为本文将服务视为平台相关的服务,即将平台支持服务所需的资源消耗与服务自身所需的资源消耗视为一个整体,所以平台的不同不会对估算算法造成影响.

此外,图中估算值始终有一定幅度的波动,除了噪声因素外,另一个主要原因是服务时间并非常数,而是具有一定的波动.这种变化与用户输入和系统状态等多种因素相关,因而Kalman滤波会根据日志信息不断地调整,以适应这种变化.

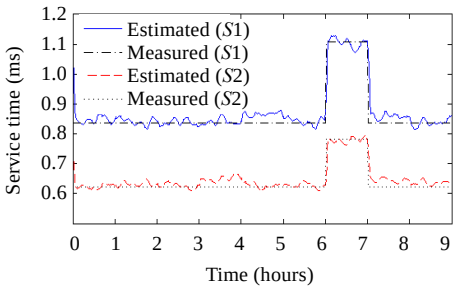


Fig.9 Comparison of estimated and measured service time

图9 服务时间估算与实测值比较

3.3.2 回归分析 vs.Kalman滤波

回归分析[27]是估算服务时间的另一种方法.但是,回归分析的方法对监测数据质量上的要求非常高,如果收集到的数据存在较大的偏差将会严重影响估算结果.如上一节所述,服务时间并不稳定,即便不考虑软件更新的影响,服务时间也会产生一定的波动,加之受监测数据存在的误差和噪音等因素的影响,致使应用回归分析方法估算服务时间的准确性难以得到保障.

本文采用非负多元线性回归[11,12,35]求解公式(5.b)对应的方程组(服务时间必定大于0),误差定义如下所示:

$$e=\sqrt{\sum(z'_k-z_k)^2}.$$

本实验中,回归分析采用的数据与Kalman滤波使用的数据相同.考虑到应用服务器上部署了14个服务,且采样周期为1分钟,所以本文以半小时为一个周期(即利用30组样本)进行回归计算.表3给出了“服务器1”上服务的估算结果.

从表中可以看出,在非负多元线性回归过程中,为了降低整体误差,大量的数据设置为0,而将所有的开销分配到少数几个服务之中,使得结果严重失真,无法准确获知各个服务的实际开销.与回归不同,Kalman滤波只需

要最新的一组数据即可进行计算.限于篇幅,图 9 给出了 Home 事务的服务时间,约为 0.83ms.其他服务与之类似,服务时间分布在 0.5ms~1.2ms 之间,且与实测值的误差小于 5%.因而我们认为,Kalman 滤波更适用于服务时间这类具有时变性参数的估计.

Table 3 Estimated results with regression method

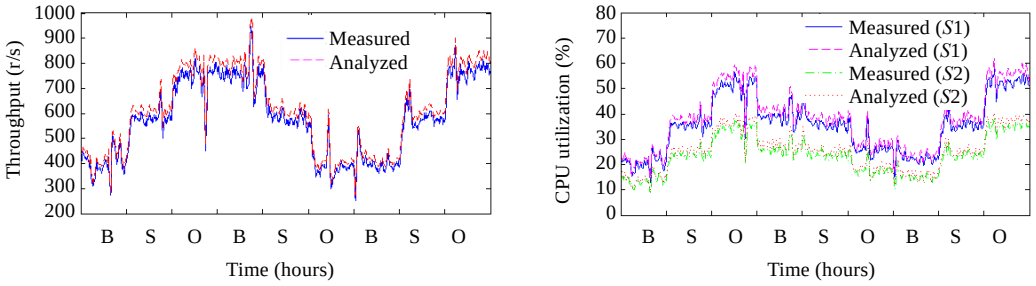
表 3 回归分析估算结果

Transaction \ Period	11	22	33	44	55	66	77	88	99	110	111	112	113	114	115	116	117	118
Home	5.1	4.8	3.3	2.8	0	0	5.0	4.5	3.1	2.7	0	0	5.8	5.9	3.5	3.3	0	0
Best sellers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
New products	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Product detail	0	0	2.1	2.3	0	0	0	0	2.3	2.8	0	0	0	0	2.0	1.9	0	0
Search request	0	0	0	0	2.6	2.9	0	0	0	0	2.2	2.8	0	0	0	0	2.5	2.7
Search results	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Admin confirm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Admin request	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Buy confirm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Buy request	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cart	0	0	0	0	2.7	2.2	0	0	0	0	3.1	2.7	0	0	0	0	3.0	2.9
Customer registration	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Order display	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Order inquiry	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.4 性能分析验证

本节将就分析出的吞吐率和 CPU 的总利用率与实测值进行比较,验证其与实测值的吻合程度.为了使模型能够更接近真实场景,下面我们来比较分析结果与实测结果的差异.

图 10 给出了图 8 所示负载变化情况下吞吐率与 CPU 利用率的实测值与分析值之间的比较.从图 10(a)可以看出,分析出的吞吐率很好地匹配了实测出的吞吐率,平均误差是 8.5%,90%以上的误差小于 12.2%.图 10(b)分别给出了应用服务器 1(*S*1)和服务器 2(*S*2)的 CPU 利用率.由于本实验中负载均衡器采用的是轮询策略,所以 *S*1 的利用率要高于 *S*2 的利用率.分析结果很好地反映了这种差别,两台服务器的平均误差分别是 7.7%和 7.3%.从图中还可以看出,第 7 个小时,虽然 Home 事务服务时间发生了改变,但是因为估算值即时做出响应,所以分析结果仍与实测值保持一致.



(a) 系统总吞吐率 (b) 应用服务器 CPU 利用率

Fig.10 Comparison of measured and analyzed results

图 10 实测值与分析值比较

4 基于模型的性能剖析

性能剖性工具由于能够剖析应用内部的执行细节,因而可为性能问题的发现提供不同粒度的性能数据.本节以两个典型应用为例,说明本方法所能提供的服务、请求和系统总体等几个粒度的性能数据对性能保障所起到的作用.

4.1 瓶颈定位

瓶颈定位是保障系统性能的有效手段,但本文并不讨论瓶颈定位的细节,而是在已有研究成果[36,37]的基础上,以服务为最小粒度报告在针对 TPC-W 基础测试所做实验中发现的潜在瓶颈.

图 11 给出了实验中数据库服务器的 CPU 利用率.对比图 11 和图 10(b)可以发现,在浏览模式下,数据库服务器的开销明显高于其他模式,而应用服务器此时开销并不大.但在其他模式下,应用服务器利用率反而会更高.本节将就造成数据库访问开销在浏览模式下激增的问题展开讨论.

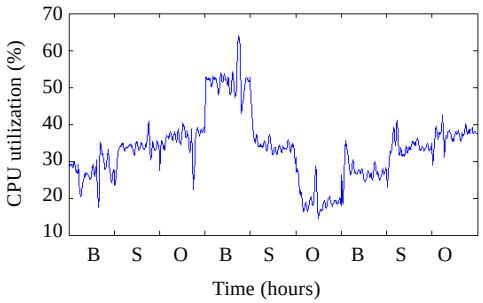


Fig.11 CPU utilization of database server

图 11 数据库服务器的 CPU 利用率

图 12 给出了实验中在第 7 个小时的分析结果.图左侧给出了“服务器 1”上访问最多的 6 个事务(占 95%的访问比例),右侧则是与之相应的数据库访问服务.圆圈中的数据表示该服务所占用的 CPU 利用率,请求起始点处的数据表示该事务所占的访问比例.

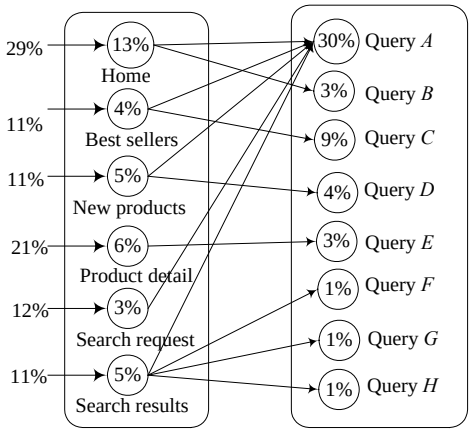


Fig.12 Analyzed results under browsing mix

图 12 浏览模式下的性能分析结果

可以看出,Home 事务占有近 1/3 的访问比例,它所调用的数据库服务也占用了大量的 CPU 时间.但问题是,这一数据库服务并非仅由 Home 事务单独使用,其他 4 个事务也会频繁调用该服务,它们共同造成了数据库访问开销的激增.因而,对该服务的优化会对多个事务的性能提升起到很大作用.进一步分析后发现,它是一个查询推荐商品的服务,通常用户可以容忍这类查询结果的更新具有一定时间的延迟,所以可以通过缓存的技术降低数据库开销.

事务的访问率与该事务所占开销的比例并没有直接关联.比如 Produce Detail 事务约占有 21%的访问比例,但是显然,它对数据库访问的贡献很小,不会造成数据库瓶颈.与之相反,Best Sellers 事务虽然访问比例较低,但

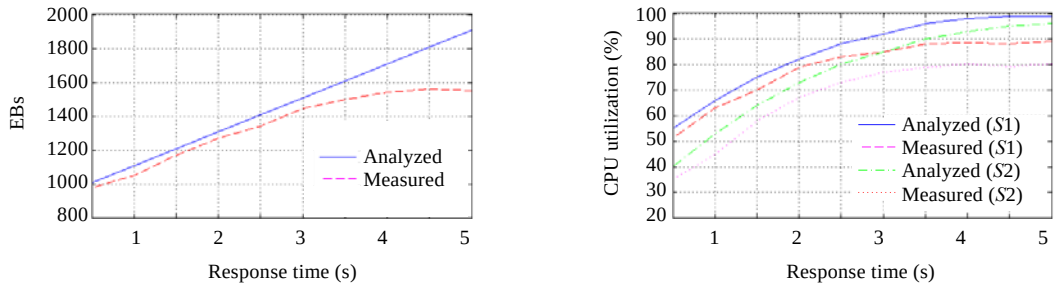
它除了调用“推荐商品”服务以外,还调用了另外一个查询“热门商品”的数据库服务,该服务所占有的 CPU 时间显然比其他服务要高出很多.因此,该服务也是一个潜在瓶颈[36,37].

4.2 容量规划

提前评估和规划系统容量是电子商务应用成功的关键.本节以容量规划为例阐述本方法对于评估系统未来性能的作用,并在此基础上进一步分析其预测能力的准确性.本实验用响应时间作为衡量用户使用质量的标准,分析在订单型负载模式下,保证请求成功率在 95%以上,并且“购买确认(buy confirm)”事务的响应时间分别为 2s,3s 和 4s 时的最大并发用户数.即在考虑请求这个粒度的情况下,评估系统总体性能表现.

图 13(a)给出了分析出的结果与实际结果的比较.分析结果一直很乐观,随着响应时间的增加,预测结果中的并发用户数也一直在增加,而实测的用户数不会超过 1 600.原因在于,性能模型并未考虑到队列长度的限制,而实际系统会因为超时和链接受限等因素造成错误率增加.当系统达到极限后,处理能力不仅不再增长,极端情况下反而会有所下降.所以,单纯地分析响应时间并不能给出系统处理能力的可靠数据.

图 13(b)描述 CPU 利用率.可以发现,分析结果的 CPU 利用率会一直增加,直到 100%,但实际系统却会因自身错误率的增加而很难到达这一指标.不过,值得注意的是,*S*1 的利用率会高于 *S*2 的利用率.因为为了保证系统有一个良好的处理能力,当错误率达到 95%时,不会继续增加并发用户数.所以,当 *S*1 发生明显错误时,*S*2 仍有额外的处理能力.但若分析 *S*1 的利用率会发现,当利用率接近 90%时,系统错误率明显增加.因此对于本系统,有理由怀疑利用率达到 90%以后的预测结果,因而当响应时间在 3s 以后时,预测结果的可信度会有所降低.利用 3s 作为上限重新观察图 13(a),可以推测响应时间为 2s 时大概支持 1 300 个并发用户,3s 时大概支持 1 500 个并发用户.至于 CPU 的利用率在什么范围内可信,则可以根据以往服务器维护的经验来估计.



(a) 并发用户数 (b) CPU 利用率

Fig.13 Analyze the performance of the system in future

图 13 系统处理能力预测

本文并不详细讨论优化策略,但可以明显地看出,服务器 *S*2 并没有被充分利用.因此,可以调整负载均衡的策略,以提高系统总吞吐率.通过调整性能模型的负载均衡策略可以发现,如果响应时间设为 4s,则系统现有资源能够满足需求,但已到了需要扩充资源的临界点.可以看出,本方法可以很好地利用监测数据,正确预测系统将来的处理能力,并可获得事务和系统总体这两个剖面的性能数据.

5 讨 论

Web 系统通常会消耗多种类型的资源,比如网络和磁盘等,这些资源都可能潜在地成为瓶颈.但因为 HTTP 协议并没有给出区分页面请求与内嵌对象(如图片和脚本等)请求的方法,所以很难获知一个页面需要下载的内嵌对象.而内嵌对象消耗的主要是磁盘和网络资源,大多数情况下并不构成业务型 Web 系统的瓶颈.且很多时候,它们会部署在独立的 HTTP 服务器上以减轻服务器压力.此外,可以很直观地计算出内存需求,因为它与并发用户数线性相关[38].所以,本文仅关注于 CPU 资源.

本方法基于平均值分析算法(mean value analysis)[14−16]进行分析.平均值算法的优点是可以应对复杂的、大规模的模型计算,但是如果负载中存在一些异常,比如短时间内的极大峰值,分析结果将无法体现这种现象.目前,对于大型复杂的系统也没有方法能够完全解决这类问题.而本文则主要关注于如何通过收集系统在良好状态下的性能数据来构造具有剖析系统未来性能的模型,这也是目前常用来预测系统未来行为的有效途径[4].

此外,在有些情况下,特别是对于共享服务器的多承租模式情况下[39],逻辑资源对系统性能的影响将会非常显著.我们的前期工作[40,41]已经研究过 LQN 分析逻辑资源对性能影响的可行性问题,本文则主要关注自动构造性能模型需要解决的几个关键问题.引入逻辑资源影响的关键是针对不同平台进行适配,当获得所部署平台的逻辑资源(比如线程池、实力池和连接池等)配置信息后,逻辑资源可以很容易地织入本文所构造的模型之中[40,41].限于篇幅,本文则未就逻辑资源的问题展开讨论.

最后,本方法的另一个优势是支持多用户类型的建模.通常,不同类型的用户会具有不同的行为或权限.在本方法中,一类用户可以对应于一个 CBMG 的实例,用 EB 的形式刻画在模型当中.在这一特性的帮助下,不同类型的用户可独立地对其进行趋势分析,然后再统一地在一个模型中分析它们共同对系统造成的影响.比如,对于一个在线商城系统,买家和卖家的行为会存在很大的差异,因而有必要分开考虑他们各自的并发规模和混合比例的变化趋势,然后再分析在其共同影响下系统所具备的性能.

6 相关工作

传统的性能剖析方法通常基于测量的技术.一类方法[7,8]是以操作系统采样周期作为基本时钟长度,通过监测活跃采样周期数计算服务时间.但这类方法适用于大任务的 CPU 消耗监测,并不适用于 Web 应用这类服务时间远远小于采样周期的应用,除非对操作系统进行扩展或利用平台相关的属性(比如特殊的定时器),但这样又会极大地影响可以移植性.另一类方法[9]则是通过扩展虚拟机指令对执行的字节码进行追踪,然后再将其转化为 CPU 开销.这类方法与底层的虚拟机或操作系统无关,但是这种运行时转化会带来至少 30%的额外开销[10],因而也很难适用于在线系统.

Yoshihira 和 Jiang 提出了一种基于监测数据发现系统中稳定关联关系的方法[42−44].他们通过收集请求处理过程中组件对资源的消耗,分析其中稳定的关联,然后据此建立关联网络,达到评估系统可扩展性的目的.但该方法并未考虑性能属性,因而无法解释服务质量(SLA)和性能之间的联系.本文则关注于性能模型的构造,指导发现在服务质量约束下的性能问题.

Woodside[45]提供了一种在设计阶段通过植入监测代码自动构造 LQN 模型的方法.该方法以用户指定的抽象级别作为连接点自动织入监测代码,然后再通过特定的测试用例收集建模所需的参数.但该方法并没有考虑负载的变化,同时也很难保证从开发平台迁移到运行平台之后参数仍然有效.Fabian 也提出了一种测试阶段自动构造性能模型的方法[46].该方法以特定测试用例在低负载下的响应时间作为服务时间估算的依据,并不适用于在线系统.而本文则关注于在线系统性能模型的自动构造方法,且性能数据获取的基础是系统运行时可收集的日志信息,并不会给系统带来额外的性能开销.

Cherkasova[11,12,35]提出了一种自动构造在线系统性能模型的方法.该方法强调了用户行为对性能的影响,相比于传统的单用户类型的建模方法可以更好地适应负载的变化.但该方法并没有关注系统内部的执行状况,而是将事务分解成不同的层次,再在每个层次上根据请求类型混合比例进行加权平均,然后利用经典排队论方法求解封闭(closed)模式下系统的性能,其分析结果只能体现系统总体上的平均水平,并不能反映单个服务或者单个请求的性能表现.本文则进一步关注用户行为和服务行为之间的关联,构造具有剖析应用内部服务性能表现的模型.本文的另一个优势是利用 Kalman 滤波取代回归分析[27]估算服务时间.如第 3.3.2 节所述,回归分析依赖于高质量的监测数据,但是对于在线系统很难保证服务时间的稳定性,因而误差会非常大.Cherkasova 的方法通过样本选取和加权平均使误差得到一定的缓解,保证了在负载混合比例变化较小时预测误差在可接受的范围内.但就性能剖析技术而言,我们认为 Kalman 滤波更适用于这类具有时变性特点的参数估计,因为 Kalman 滤波本身就是为应对动态环境中具有时变性和噪声特点的参数估算而设计的.

Zheng 提出了一种使用非线性 Kalman 滤波调整服务时间的方法[47],该方法的目标是为多种性能模型提供一种在线调整参数的方法.他们将性能模型视为黑盒,使用非线性的方法,通过近似地对性能模型求偏导数获得 H,即计算每一个组件服务时间的微小变化对结果的影响.因此,计算复杂度为组件个数与一次性能模型计算时间的乘积.性能模型的计算复杂度,根据模型复杂度的不同也有所不同,比如用 LQN 求解本文测试用例规模的模型,在 Pentium D 3.4G 的机器上,分析的方法需要 30s 以上,而仿真方法需要 30 分钟以上.而本系统中组件总数为 14×2+37=65 个.因此,使用分析算法一次迭代也需要 33 分钟左右的时间.随着系统的进一步复杂化,计算开销将更加庞大.所以,该方法只适用于服务器一级的性能预测,通过降低模型的复杂度降低计算开销.而本文提出了一种线性算法,不依赖于性能模型,复杂度降为 $O(m^3)$,可以即时算出结果.此外,本文的关注点主要是性能模型的构造,而不仅仅是对服务时间这个参数进行调整.

7 总 结

本文的主要贡献是,为在线的 Web 系统提出了一种基于性能模型的剖析方法.该方法使用监测数据动态构造出符合系统当前状态的性能模型,分析结果可以为性能剖析和评估提供多种粒度的性能数据.其中,主要技术贡献包括:1) 将负载特征转换为一种性能模型所能接受的保留了负载混合比例特征的模型;2) 给出了基于轨迹构造应用执行图的方法;3) 提出了基于 Kalman 滤波估算服务时间的算法,解决了服务时间适应系统软、硬件环境变化的问题.后续,我们准备一方面在 LQN 的诊断与分析技术基础上进一步研究解决自动瓶颈检测与优化的问题;另一方面,在应用的部署规划(placement)方面研究多应用的部署优化问题,以达到在保证用户使用质量的前提下降低能耗的目的.

References:

[1] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M. Above the clouds: A Berkeley view of cloud computing. Technical Report, No.UCB/EECS-2009-28, University of California at Berkley, 2009.

[2] Menasce DA. Capacity Planning for Web Performance: Metrics, Models, and Methods. Upper Saddle River, NJ: Prentice Hall, 1998.

[3] Yang FQ, Mei H. Internetware: New sofeware pattern in future. China Education Network, 2005,(7):52−54 (in Chinese with English abstract).

[4] Lü J, Ma XX, Tao XP, Xu F, Hu H. Research and development in internetware. Science in China (E: Information Sciences), 2006, 36(10):1037−1080 (in Chinese with English abstract).

[5] Litoiu M, Woodside M, Zheng T. Hierarchical model-based autonomic control of software systems. In: Proc. of the Workshop on Design and Evolution of Autonomic Application Software. St. Louis, 2005. 1−7. [doi: 10.1145/1083063.1083071]

[6] Abdelzaher TF, Shin KG, Bhatti N. Performance guarantees for Web server end-systems: A control-theoretical approach. IEEE Trans. on Parallel and Distributed Systems, 2002,13(1):80−96. [doi: 10.1109/71.980028]

[7] Sun Microsystems, Inc. JVM tool interface (JVMTI): URL. http://download.oracle.com/javase/1.5.0/docs/guide/jvmti/

[8] Jordan M, Czajkowski G, Kouklinski K, Skinner G. Extending a J2EE™ server with dynamic and flexible resource management. In: Proc. of the 5th ACM/IFIP/USENIX Int'l Conf. on Middleware. Toronto, 2004. 18−22. http://dl.ifip.org/index.php/lncs/article/view/26363

[9] Binder W, Hulaas J. A portable CPU-management framework for Java. IEEE Internet Computing, 2004,8(5):74−83. [doi: 10.1109/MIC.2004.28]

[10] Hulaas JG, Kalas D. Monitoring of resource consumption in Java-based application servers. In: Proc. of the 10th HP OpenView University Association Plenary Worksop. 2003. http://en.scientificcommons.org/22443790

[11] 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. In: Proc. of the Conf. on Middleware. 2007. http://dl.acm.org/citation.cfm?id=1516142

[12] Cherkasova L, Ozonat K , Symons J, Smirni E. Automated anomaly detection and performance modeling of enterprise applications. ACM Trans. on Computer Systems, 2009,27(3):6.1−6.32. [doi: 10.1145/1629087.1629089]

[13] Hrischuk CE, Murray WC, RoliaJerome JA, Rolia A. Trace-Based load characterization for generating performance software models. IEEE Trans. on Software Engineering, 1999,25(1):122−135. [doi: 10.1109/32.748921]

[14] Woodside M. Throughput calculation for Basic stochastic rendezvous networks. Performance Evaluation, 1998,9(2):143−160. [doi: 10.1016/0166-5316(89)90039-4]

[15] Woodside M, Neilson JE, Petriu DC, Majumdar S. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. IEEE Trans. on Computers, 1995,44(1):20−34. [doi: 10.1109/12.368012]

[16] Rolia JA, Sevcik KC. The method of layers. IEEE Trans. on Software Engineering, 1995,21(8):689−700. [doi: 10.1109/32.403785]

[17] TPC-W benchmark. http://www.tpc.org/tpcw/default.asp

[18] Bench4Q. http://forge.ow2.org/projects/jaspte/

[19] Menascé D, Almeida VAF, Fonseca R, Mendes MA. A methodology for workload characterization of e-commerce sites. In: Proc. of the ACM E-Commerce. 1999. 119−128. [doi: 10.1145/336992.337024]

[20] Cherkasova L, Phaal P. Session-Based admission control: A mechanism for peak load management of commercial Web sites. IEEE Trans. on Software Engineering, 2002,51(6):669−685. [doi: 10.1109/TC.2002.1009151]

[21] JavaEE. http://www.oracle.com/technetwork/java/javaee/overview/index.html

[22] Krishnamurthy D, Rolia JA, Majumdar S. A synthetic workload generation technique for stress testing session-based systems. IEEE Trans. on Software Engineering, 2006,32(11):868−882. [doi: 10.1109/TSE.2006.106]

[23] Schroeder B, Wierman A, Harchol-Balter M. Open versus closed: A cautionary tale. In: Proc. of the USENIX NSDI. 2006. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.4441

[24] Franks G, Hubbard A, Majumdar S, Neilson J, Petriu C, Rolia J, Woodside M. A toolset for performance engineering and software design of client-server systems. Performance Evaluation, 1995,24(1-2):117−136. [doi: 10.1016/0166-5316(95)96869-T]

[25] InfraRED. http://infrared.sourceforge.net/versions/latest/

[26] Israr T, Woodside M, Franks G. Interaction tree algorithms to extract effective architecture and layered performance models from traces. The Journal of Systems and Software, 2007,80(4):474−492. [doi: 10.1016/j.jss.2006.07.019]

[27] Rolia J, Vetland V. Correlating resource demand information with ARM data for application services. In: Proc. of the ACM Workshop on Software and Performance. 1998. [doi: 10.1145/287318.287366]

[28] Kalman RE. A new approach to linear filtering and prediction problems. Trans. of the ASME-Journal of Basic Engineering, 1960. [doi: 10.1115/1.3662552]

[29] Song WY, Zhang Y. Kalman Filter. Beijing: Science Press, 1991 (in Chinese).

[30] Catlin DE. Estimation, Control, and the Discrete Kalman Filter. New York: Springer-Verlag, 1989.

[31] Geir E. Data Assimilation: The Ensemble Kalman Filter. Berlin, New York: Springer-Verlag, 2007.

[32] 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 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3781.htm [doi: 10.3724/ SP.J. 1001.2010.03781]

[33] MBP. http://www.trustie.net/projects/wiki/view/MBP

[34] Jprobe. http://www.quest.com/jprobe/

[35] Zhang Q, Cherkasova L, Mi NF, Smirni E. A regression-based analytic model for capacity planning of multi-tier applications. Journal of Cluster Computing, 2008,11(3):197−211. [doi: 10.1007/s10586-008-0052-0]

[36] Xu J. Rule-Based automatic software performance diagnosis and improvement. In: Proc. of the Workshop on Software and Performance (WOSP). 2008. [doi: 10.1145/1383559.1383561]

[37] Li J, Chinneck J, Woodside M, Litoiu M, Iszlai G. Performance model driven QoS guarantees and optimization in clouds. In: Proc. of the ACM/IEEE ICSE Workshop on Cloud Computing. Vancouver, 2009. [doi: 10.1109/CLOUD.2009.5071528]

[38] Capacity Planning for WebLogic Portal. http://download.oracle.com/docs/cd/E13218_01/wlp/docs81/index.html

[39] Azeez A, Perera S, Gamage D, Linton R, Siriwardana P, Leelaratne D, Weerawarana S, Fremantle P. Multi-Tenant SOA middleware for cloud computing. In: Proc. of the 2010 IEEE 3rd Int'l Conf. on Cloud Computing (CLOUD). 2010. 458−465. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5557959 [doi: 10.1109/CLOUD.2010.50]

[40] Zhang Y, Huang T, Wei J, Chen NJ. Architectural level performance modeling of component system based on container middleware. Journal of Software, 2006,17(6):1328−1337 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/ 1328.htm [doi: 10.1360/jos171328]

[41] Huang X, Zhang WB, Zhang B, Wei J. Impacts separation framework for performance prediction of middleware-based systems. In: Proc. of the IEEE Int'l Computer Software and Applications Conf. (COMPSAC). 2009. 178−187. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5254130 [doi: 10.1109/COMPSAC.2009.131]

[42] Jiang GF, Chen HF, Yoshihira K. Discovering likely invariants of distributed transaction systems for autonomic system management. Cluster Computing, 2006,9(4):385−399. [doi: 10.1007/s10586-006-0008-1]

[43] Jiang GF, Chen HF, Yoshihira K. Efficient and scalable algorithms for inferring likely invariants in distributed systems. IEEE Trans. on Knowledge and Data Engineering, 2007,19(11):1508−1523. [doi: 10.1109/TKDE.2007.190648]

[44] Jiang GF, Chen HF, Yoshihira K. Profiling services for resource optimization and capacity planning in distributed systems. Cluster Computing, 2008,11(4):313−329. [doi: 10.1007/s10586-008-0063-x]

[45] Woodside M, Hrischuk C, Selic B, Brayarov S. Automated performance modeling of software generated by a design environment. Performance Evaluation, 2001,45(2-3):107−123. [doi: 10.1016/S0166-5316(01)00033-5]

[46] Brosig F, Kounev S, Krogmann K. Automated extraction of palladio component models from running enterprise Java applications. In: Proc. of the Int'l ICST Conf. on Performance Evaluation Methodologies and Tools. 2009. http://dl.acm.org/citation.cfm?id=1698835 [doi: 10.4108/ICST.VALUETOOLS2009.7981]

[47] Zheng T, Woodside CM, Litoiu M. Performance model estimation and tracking using optimal filters. IEEE Trans. on Software Engineering, 2008,34(3):391−406. [doi: 10.1109/TSE.2008.30]

附中文参考文献:

[3] 杨芙清,梅宏.网构软件:未来的新型软件形态.中国教育网络,2005,(7):52−54.

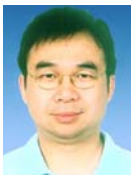
[4] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E 辑:信息科学),2006,36(10):1037−1080.

[29] 宋文尧,张牙.卡尔曼滤波.北京:科学出版社,1991.

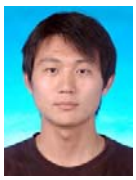
[32] 王伟,张文博,魏峻,钟华,黄涛.一种资源敏感的 Web 应用性能诊断方法.软件学报,2010,21(2):194−208. http://www.jos.org.cn/1000-9825/3781.htm [doi: 10.3724/SP.J.1001.2010.03781]

[40] 张勇,黄涛,魏峻,陈宁江.基于容器中间件的组件系统体系结构性能评价.软件学报,2006,17(6):1328−1337. http://www.jos.org.cn/1000-9825/17/1328.htm [doi: 10.1360/jos171328]

黄翔(1982－),男,湖南常德人,博士生,主要研究领域为网络分布计算,软件工程.



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.



王伟(1982－),男,博士,助理研究员,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,主要研究领域为网络分布计算,软件工程.

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