ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.9, September 2007, pp.2117−2129
DOI: 10.1360/jos182117

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

一种基于性能模型的中间件自配置框架*

胡剑军[1,2,3+], 官荷卿[1,2,3], 魏 峻[1,2], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术研究中心,北京 100080)

[2](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100080)

[3](中国科学院 研究生院,北京 100049)

A Performance Model-Based Self-Configuration Framework for Middleware

HU Jian-Jun[1,2,3+], GUAN He-Qing[1,2,3], WEI Jun[1,2], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

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

+ Corresponding author: Phn: +86-10-62661581, E-mail: hujj@otcaix.iscas.ac.cn

Hu JJ, Guan HQ, Wei J, Huang T. A performance model-based self-configuration framework for middleware. *Journal of Software*, 2007,18(9):2117−2129. http://www.jos.org.cn/1000-9825/18/2117.htm

Abstract: High dynamic computing environment makes QoS (quality of service) guarantee more important for component-based distributed system. Software system should possess self-tuning capacity for reacting to external environment variation. This paper proposes an adaptive self-configuration framework, which can automatically tune configuration parameters to preserve QoS as workload changes. The key of this framework is a layered queuing network based performance model, and it guides the search for the best combination of configuration parameters to satisfy the QoS requirement. This self-configuration framework is prototyped on OnceAS application server, and is validated using StockOnline by comparing the performance requirement satisfaction with and without this framework. The results show that through the framework's regulation, system performs well on QoS goal.

Key words: EJB(enterprise Java Bean); QoS(quality of service); performance model; self-configuration; layered queuing network

摘 要: 高动态的计算环境使得 QoS(quality of service)保障对于基于组件的分布式系统越来越重要,软件系统需要具备自我调整的能力以适应外部环境的变化.给出一种自适应的中间件配置框架,能够动态感知负载变化,并自动调整系统参数配置以保持用户所要求的服务质量.该框架的核心是一个基于分层排队网络的性能预测模型,用于指导搜索最优的资源配置,使性能需求得到最大的满足.在 OnceAS 应用服务器上进行原型实现,并以 StockOnline 应用做实验,比较了在使用和不使用该框架时的性能需求的满足情况.结果显示,在负载增加时,通过自配置框架的调控,应用性能需求的保障程度得到了较大的提升.

关键词: EJB(enterprise Java Bean);服务质量;性能模型;自配置;分层排队网络

* Supported by the National Natural Science Foundation of China under Grant No.60573126 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312005 (国家重点基础研究发展计划(973))

Received 2006-06-12; Accepted 2006-08-22

中图法分类号: TP311 文献标识码: A

以Web应用服务器为代表的分布式组件中间件系统(如EJB,CORBA,.NET)已经发展为Web计算环境中的主要基础软件[1].中间件系统通过屏蔽底层平台的异构性提供大量应用所需要的服务(如事务、安全等),极大地简化了大规模复杂分布式系统的开发;另外,通过定义良好的组件模型,大量COTS组件能够部署到任何与标准兼容的中间件平台实现上,提高了软件复用的程度.

与Web计算环境的开放性、动态性、多变性等特征相适应,软件系统呈现出柔性、多目标、连续反应式等新的形态[2],这使得诸如性能、可靠性等非功能性需求受到重视,要求软件系统能够适应环境的变化,动态调整和演化,以提供满足客户需求的服务质量(quality of service,简称QoS).中间件在支持应用的功能性需求方面虽然取得了较好的效果,但在非功能性支持方面尚处于“尽力而为”阶段,缺乏相应的QoS保障机制,难以满足复杂多变的计算环境的要求.本文只关注性能特征方面.基于组件的应用,其性能不但决定于应用本身,而且决定于应用所部署的中间件系统,而中间件系统的性能在很大程度上依赖于正确的资源参数配置[3].目前,大部分中间件系统只支持静态的配置,在系统部署时,手工设置各种配置参数以获得满足需求的性能[4].这种手工配置方式通常需要进行或多或少的试运行以确定最优配置,耗费时间,效率低下,而且配置人员需要有较高的系统管理技能.另一方面,对于诸如e-commerce之类的计算环境,负载始终处于高动态变化之中,配置好的系统在某时刻能满足所要求的性能需求,但在另外一个时刻就不能再满足.因而,静态配置的系统难以适应负载的动态变化.为进行性能保障,中间件系统需要运行时自调整机制.目前,大多数研究使用基于控制论的方法,在每个控制回路中通过比较QoS实际值与期望值的差异,根据控制规则进行相应的配置调整.在这种方法中,系统是一个“黑盒”,控制器不了解系统的行为,无法评价所选取配置的好坏,缺乏足够的准确性,导致控制中的“摆动”[5].

为解决上述问题,本文以 EJB 中间件为目标平台,提出了一种基于性能模型的动态配置框架,在负载发生变化的情况下,自适应地调整中间件配置参数,使得系统的性能需求得到保证.该框架的核心是一个基于分层排队网络的性能模型,它能够预测在给定中间件配置和负载下的性能度量.在配置决定过程中,性能模型用来评估不同候选配置的好坏,指导搜索最优的配置,从而提高性能保障的准确性和有效性.本文的贡献主要有两点:一是使用“硬约束”和“软约束”概念来定义性能需求,将性能保障转化为一个多约束的优化问题;二是使用分层排队网络模型,有效地预测组件应用在不同配置下的性能度量,为配置决定提供依据.

本文第 1 节分析系统的性能行为和需求.第 2 节提出基于分层排队网络的 EJB 性能模型,并验证了该模型预测的有效性.第 3 节从架构上对性能自配置框架进行分析,着重介绍所采用的基于爬山优化的配置选择算法.第 4 节介绍实验设计及其结果.第 5 节比较相关工作.最后对本文进行总结,并对将来的工作进行展望.

1 动 机

QoS是用户需求、系统行为和该行为下的资源需求这 3 个要素间交互作用的结果,对任何一个要素的不了解将导致不可控制的QoS[6].为进行高效的性能保障,需要准确描述系统的性能行为,明确应用的性能需求目标.

1.1 系统性能行为

描述系统的性能行为的关键是模型化因并发访问引起的资源竞争对请求处理的性能影响.对于基于容器架构的组件技术,每个组件都运行在其对应的组件容器中,应用服务器使用组件容器来管理组件的执行,所有对组件的调用都由容器进行处理,容器代表组件管理各种资源以及组件和外部系统的交互.图 1 给出了 EJB 容器架构.并发竞争主要体现在 3 个方面:线程、组件实例和数据库连接.当客户调用服务器端组件时,首先容器负责分配一个工作线程,然后容器负责获取所请求的组件实例,调用该组件实例的业务方法,如果组件需要访问数据库,容器则负责分配数据库连接以供使用.对于每一个请求,需要同时拥有这 3 个资源才能完成调用处理,在请求每个资源时,若没有空闲,则挂起请求进入排队等待.组件的性能行为依赖于其容器在这些资源上的配置.对于由多个组件装配而成的服务器端应用,其性能行为不仅与所包含的各个组件的性能行为相关,还依赖于组件之

间的交互关系,如组件之间方法调用的频度、并发度等.

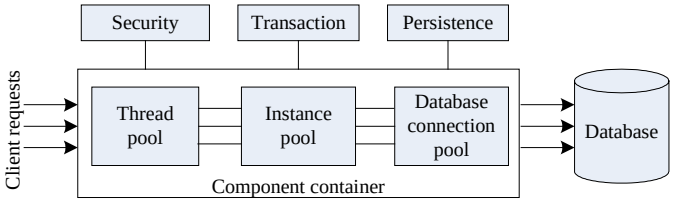


Fig.1 EJB container architecture

图 1 EJB 容器结构

排队模型在软件系统的性能分析上得到了广泛应用,其缺点是只能描述每次一种资源需求的软件组件,因而不适合用于EJB容器的性能建模.要描述多资源并发需求则需要更为复杂的模型,如扩展排队网络(extended queuing networks)、性能Petri网(performance Petri nets)、随机进程代数(stochastic process algebras).分层排队网络[7](layered queuing networks,简称LQN)是一种新型的扩展排队网络,它强调的是执行服务请求时服务实体间多层次的嵌套调用以及它所引起的并发资源竞争.因而,对于在组件交互中既提供服务又请求其他组件服务的组件,LQN能够很方便地描述其性能行为.另外,LQN能够避免其他模型的状态爆炸问题[8].基于这些原因,我们选择使用LQN来建模EJB应用的性能行为,第 2 节给出了EJB应用的详细性能建模方法.

1.2 性能需求

性能需求是进行性能保障的基础.性能需求通常表示为性能指标的约束.对于基于服务器端组件的分布式应用,性能需求有其特殊性.首先,对于应用提供的不同业务服务,客户有着不同的性能约束.以StockOnline为例,它模拟在线股票交易系统,通过 1 个会话Bean(*BrokerBean*)和 4 个实体Bean(*Account*,*StockHolding*,*StockItem*,*StockTx*)之间的交互协作,向客户提供 6 种业务服务,用户可以买卖股票、更新账户信息、查询股票价格、查询当前所持有股票的详情等.典型的性能约束可以是:查询股票价格服务的平均响应时间在 200ms以内,或者单位时间内购买股票的交易量需要达到每秒 50 次以上.因而,性能需求要能够区分不同服务的性能约束.其次,不同性能约束之间常常是相互影响的(比如服务吞吐量的提升常常会导致服务响应时间的下降,反之亦然[9]),或者应用对不同服务的性能约束有所偏重(比如相对于股票查询服务的吞吐量约束,应优先保证买卖股票服务的吞吐量以使系统能够处理更多的交易,从而获得更多的交易佣金),因而,性能需求要能表达不同性能约束之间的权衡.本文借鉴文献[10]将性能约束分为“硬约束”和“软约束”.“硬约束”强调性能指标必须达到的要求,代表了客户所能容忍的性能底线,而“软约束”表达的是所期望的性能目标,越接近,应用所能获得的效用越大,客户的满意度也越高.需要指出的是,这里的“硬约束”和实时系统里的硬实时约束有所不同,虽然两者反映的都是性能约束的严格性,但是后者要求对每个请求都必须满足约束,而在本文中,“硬约束”和“软约束”一样,针对的是一段时间内所有请求的平均性能.对于组件应用所提供的服务s,主要关注两个通用的性能指标:

- 服务的平均响应时间(R^s):对服务s的请求从到达服务器经过处理后再返回响应给客户端的平均时间花费.
- 服务的平均吞吐量(T^s):单位时间内处理完毕的对服务s的请求数.对于组件应用来说,提供给客户的每个服务即是一个业务(事务),可以使用每秒内完成的事务数(TPS)来度量.

对于这两个性能指标,客户所关心的是服务最大平均响应时间 $R^s_{\max}$ 和最小平均吞吐量 $T^s_{\min}$. $R^s_{\max}$ 指定了所能忍受的最大服务响应延迟, $T^s_{\min}$ 则指定所能接受的最低的服务执行能力.因此, $R^s_{\max}$ 和 $T^s_{\min}$ 描述了性能的“硬约束”,即:

$$R^s \le R^s_{\max} \quad \text{(constraint 1)}$$

$$T^s \ge T^s_{\min} \quad \text{(constraint 2)}$$

定义性能偏差为性能指标的实际值相对于“硬约束”值之间的偏离程度,那么响应时间偏差为

$$\Delta R_s=(R^s_{\max}-R^s)/R^s_{\max},$$

吞吐量偏差为

$$\Delta T_s=(T^s-T^s_{\min})/T^s_{\min}.$$

在性能偏差的基础上,定义效用函数 Utility 来描述“软约束”的性能效用:

$$Utility=\sum_{s\in S}w_s\times(w^R_s\times\Delta R_s+w^T_s\times\Delta T_s) \quad \text{(constraint 3)}$$

在(constraint 3)中,我们简单地使用性能偏差的线性关系来描述效用函数.可以看到,当响应时间越小时,吞吐量越大,则性能偏差越大,Utility 值也越大,因而 Utility 值直观地反映了“软约束”的满意程度.其中,S 为组件应用所提供的服务集合.w_s反映不同服务的权重,$\sum_{s\in S}w_s=1$.对于每个服务s, w^R_s 和 w^T_s 分别反映了服务s对响应时间和吞吐量的不同偏重, $w^R_s+w^T_s=1$.

2 性能模型

本文采用基于模型分解/组合的方式构建组件应用的性能模型.首先,对于每个应用组件,使用分层排队网络方法来描述组件容器内的资源竞争,得到各组件的性能模型(称为组件性能子模型);然后,通过分析应用的 UML 设计(主要是序列图),从中抽取组件之间的调用关系,根据调用关系将各个组件性能子模型连接成应用的完整分层排队网络模型.

2.1 分层排队网络

LQN 使用服务实体(提供服务的软件组件或硬件)来定义软件系统.服务实体在 LQN 中表示为任务(task),分为 3 类:客户任务(client task,仅发送请求)、活动服务任务(active server task,能接受和发送请求)、纯服务任务(pure server task,仅接受请求).任务的入口(entry)表示任务所提供的服务,等同于软件对象对外暴露的接口方法.每个入口有其执行时间花费(service demand),入口在处理服务请求时可以向其他任务的入口请求服务,术语“分层”所表达的正是入口间嵌套调用的层次关系.每个任务有一个请求队列为其所有入口共用,每个服务请求都经过队列再分发到相应入口.每个任务有一个或多个工作线程用于各服务入口的执行,多线程任务可以同时处理多个服务请求.LQN 提供 3 种服务请求类型:异步、同步和转发.LQN 能预测的性能指标有响应时间、吞吐量、资源利用率和排队延迟.在模型的图形表示上,任务以包含多个分格的矩形来表示,最右边的分格是任务的名称和表示线程数量的参数,其中,*inf* 表示无限多个线程数.其他每个分格是任务的一个入口,每个入口有参数[*service demand*]代表该入口的执行时间花费.入口之间的同步服务请求用带实心箭头的请求弧表示,请求弧上的参数表示调用的次数.关于分层排队网络更多的知识,可以参考文献[7].

2.2 EJB组件性能模型

对于不同类型的组件,容器对调用请求的处理是不同的,本文分析并给出了无状态会话 Bean、有状态会话 Bean、实体 Bean 的容器性能模型.对于消息驱动 Bean,容器对其调用的处理也可以 LQN 方法进行建模,但是从 Bean 组件之间的交互来说,对消息驱动 Bean 的调用是异步的,不适合用 LQN 来描述.因而只给出了实体 Bean 和会话 Bean 的容器性能模型.

2.2.1 实体 Bean 组件性能模型

图 2 给出了实体 Bean 容器的 LQN 模型.ContainerFront-Entrance 任务接受调用请求,是容器的入口.在这里,特别地将 Home 接口中的 create 和 remove 方法单独列举出来,是因为这两种方法在实现上不属于组件所提供的服务,因而它们不需要组件实例,直接使用容器实现的回调方法(伪任务 CallBack)来完成实体 Bean 的创建和删除.Request Processor 任务是一个多线程任务,每个调用请求的处理由一个线程来完成.若并发请求数超过M时,则请求进入队列,处于等待状态.在每种业务方法处理之前,需要得到对应的组件实例,这由一个单线程的互斥任务 InstanceManager 实现.如果请求所对应的组件实例在实例池中,prepareInstance 入口直接返回池中的该实例,否则,调用容器的回调方法将实例池中的一个实例换回到数据库中,同时,将所请求实例从数据库中换入.

实例池由$I个同样的 Instance 任务组成,$I为实例池的大小.$p是所请求的组件实例在实例池中命中的概率,那么需要进行换入/换出操作的概率是 1–$p.获得请求对应的组件实例后,调用 Instance 任务的 busiMethod 入口,即组件实例的业务方法.Instance 任务是一个单线程任务,因为一个组件实例每个时刻只能服务一个请求.此外,数据库连接池虽然不包含在容器中,但是它影响实体 Bean 的请求处理,CallBack 的回调方法和组件实例的业务方法在进行数据库操作时,都需要从连接池中获取一个连接.Database Connection Pool 任务代表连接池组件,其参数$C为连接池的大小.

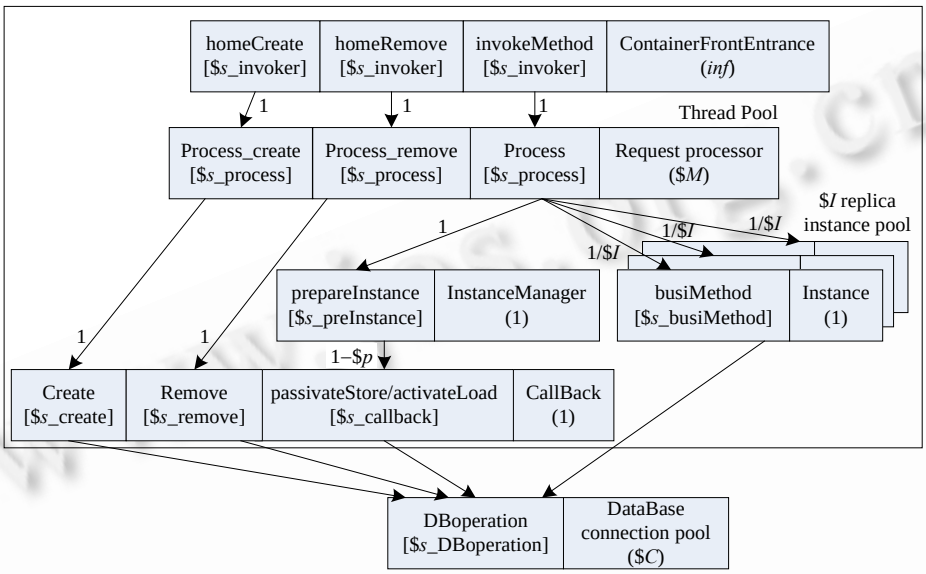


Fig.2 Layered queuing network model for entity Bean

图 2 实体 Bean 的分层排队网络模型

2.2.2 会话 Bean 组件性能模型

图 3 和图 4 分别给出了有状态会话 Bean 和无状态会话 Bean 的性能模型.有状态会话 Bean 容器的请求处理行为与实体 Bean 容器的很相似,主要差别在于有状态 Bean 容器的实例池的钝化/激活操作针对于本地存储,而不是数据库.对于无状态会话 Bean,容器无须为组件保持任何状态,并且组件的业务方法是可以任意多线程访问,因而无状态会话 Bean 无须实例池.同样,无状态会话 Bean 的创建和销毁操作(create/remove)基本没有性能开销,因而图 4 中没有给出相应的任务入口.

2.3 构造EJB应用性能模型

对于组件应用来说,性能行为由其所包含的组件所决定,其性能模型由各组件的性能子模型组合而成.构造 EJB 应用完整性能模型包括 3 个步骤:

(1) 获取组件交互关系.主要是EJB组件间的方法调用关系,如调用顺序、调用次数、调用并发度等信息.这可以通过分析应用的设计文档或源代码得到.目前有很多工具能够从UML图中抽取组件之间的交互关系[11,12].

(2) 模板实例化.第 2.2 节给出的性能模型是针对一类 Bean 的模板,没有涉及特定 Bean 的业务方法,入口的名称统一为 busiMethod 或 invokeMethod.因而对于具体的 EJB 组件,需要以各自的业务方法对模板进行实例化,得到该 EJB 组件的性能子模型.

(3) 组合子模型.根据第(1)步所得到的组件间的交互关系,将各组件子模型连接起来,组成整个应用的性能模型.

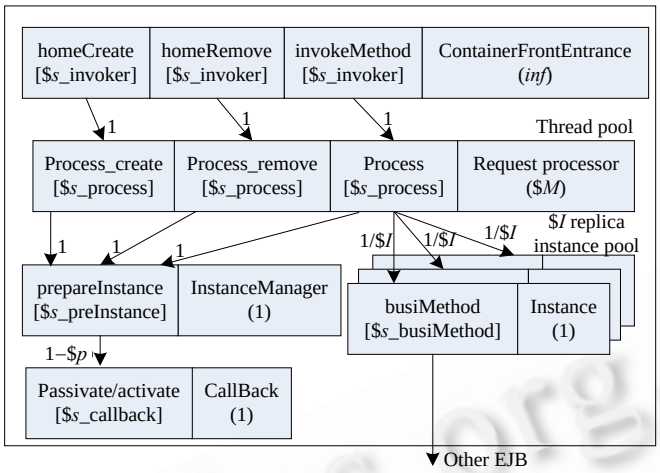


Fig.3 Layered queuing network model for statefull Bean

图 3 有状态会话 Bean 的分层排队网络模型

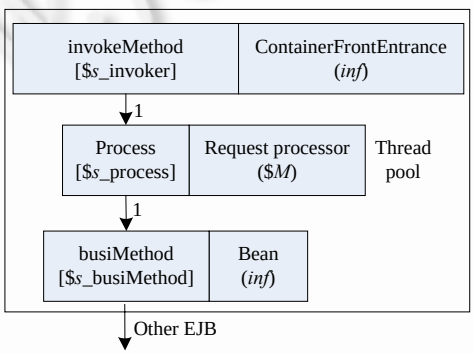


Fig.4 Layered queuing network model for stateless Bean

图 4 无状态会话 Bean 的分层排队网络模型

以 StockOnline 应用的一个场景来说明完整的性能模型的构造过程.图 5 给出了 updateAccount 服务的 UML 序列图.

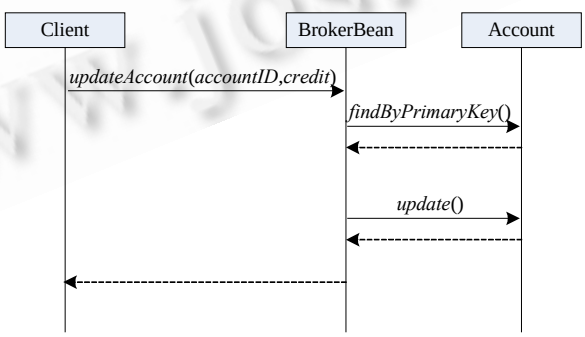


Fig.5 Sequence diagram for updateAccount service

图 5 updateAccount 服务的序列图

该服务涉及两个组件:BrokerBean 是一个无状态会话 Bean,是应用向客户提供的服务入口;Account 是 CMP 实体 Bean,维护账户状态.在 BrokerBean 的 updateAccount 服务执行过程中,首先调用 findByPrimaryKey 方法得

到对应的 Account 实例,然后调用 update 方法更新账户状态.对应的构造过程,首先对 BrokerBean 和 Account 分别套用对应的容器性能模板,得到其组件性能子模型(图的上半部和下半部),再根据调用关系将两个组件性能子模型连接起来.图 6 给出了连接后生成的性能模型.本示例只是针对 updateAccount 的一个服务场景,对于整个应用来说,需要综合分析所有场景得到组件间完整的调用关系.

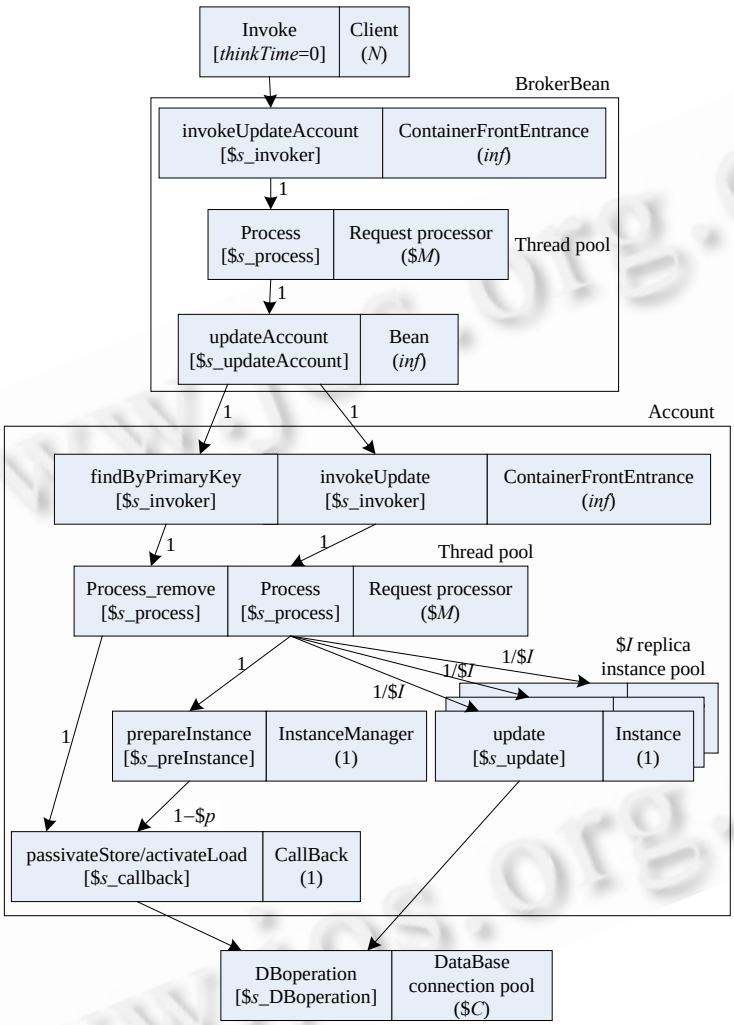


Fig.6 Layered queuing model for updateAccount service

图 6 updateAccount 服务的分层排队模型

2.4 模型验证

在框架中,性能模型用来计算给定配置的性能度量,以指导搜索最优的配置决定.因而性能预测模型应该具有足够的精确性,并能够反映性能随负载的变化趋势.为验证性能模型预测结果的有效性,我们在OnceAS[13]应用服务器平台上,以第 2.3 节中StockOnline应用的updateAccount服务场景作为实验,在给定服务器配置下,比较在不同负载时性能的预测值和实际测量值的差异.测试的软、硬件环境和第 4 节中的实验环境一样.为求解性能模型,需要确定性能模型中各个任务入口的CPU时间花费(即LQN中的service demand参数).在空负载情况下,让单个客户进行访问,使用性能剖析工具JProbe[14]来获得这些参数值.JProbe提供了一个Java代码的性能检测工

具,它能够给出线程执行路径中每种方法的执行时间,在此基础上,可以得到各任务入口的服务需求参数.测试得到的模型参数值如下:

BrokerBean: $s_invoker=0.823ms; $s_process=0.003ms; $s_updateAccount=0.020ms.

Account: $s_invoker=0.642ms; $s_preInstance=0.368ms; $s_callback=0.260ms; $s_update=0.220ms.

DataBase: $s_DBoperation=2.427ms.

为着重突出资源竞争对性能的影响,特意将服务器的配置参数设置为较小的值,并且关闭 EJB 容器的实例高速缓存机制,以保证实例池的 passivate/activate 操作.设定的 EJB 容器的配置为

BrokerBean: $M=6; Account: $M=6, $I=10; DataBase: $C=5.

我们分别测试了 1~15 个并发客户情况下的平均响应时间和吞吐量,每个客户线程连续以不同的 accountID 参数值调用 100 次 updateAccount 服务,因而对所有请求,Account 组件容器的实例池中的命中率$p=0.此外,每个客户相邻调用之间的间隔时间为 0(即 *ThinkTime*=0).图 7 和图 8 分别给出了模型预测值和实际测量值在平均响应时间和吞吐量上的比较.平均来说,模型预测值的响应时间偏差为 6%,吞吐量偏差为 8%.另外,从图 7 和图 8 中可以看出,模型预测值曲线和实际测量值曲线的形状变化很相近,说明模型能够很好地跟踪性能的变化,这对于自配置框架来说非常重要,因为自配置框架正是通过比较候选配置对应的模型预测值来选择最优配置的.

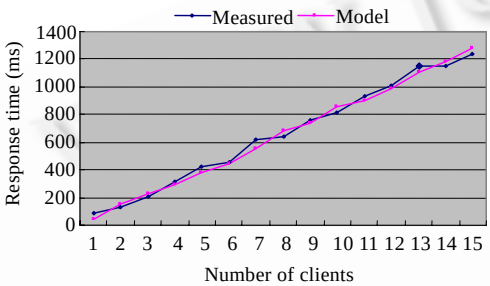


Fig.7 Response time comparison between measured and modeled

图 7 响应时间的测量值和模型预测值比较

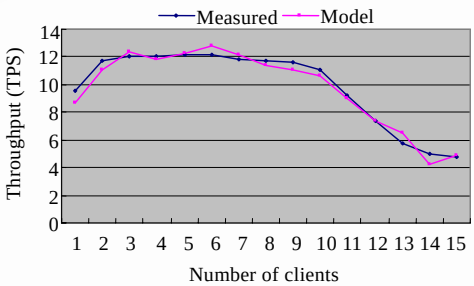


Fig.8 Throughput comparison between measured and modeled

图 8 吞吐量的测量值和模型预测值比较

3 自配置框架

3.1 框架结构

图 9 为自配置框架的体系结构,主要包括 6 个组成模块:负载监视器(workload monitor)、配置选择器(configuration selector)、性能剖析器(performance profiler)、模型构造器(model constructor)、性能模型求解器(performance model solver)和 QoS 监控器(QoS monitor).

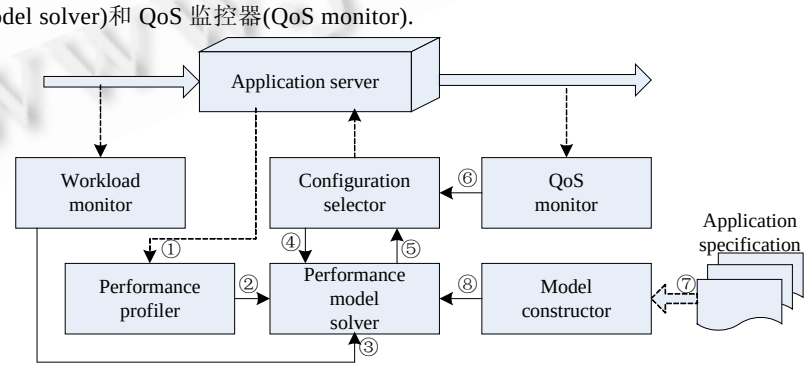


Fig.9 Architecture of self-configuration framework

图 9 性能自配置框架的体系结构

QoS监视器统计在每个调整时间间隔内各服务完成的请求数以及各请求的响应时间,计算在每个调整时间间隔内的平均响应时间和吞吐量,并检查是否违反QoS需求,决定是否需要进行重配.若需要,则向配置选择器发出重配命令⑥.配置选择器在接收到QoS监视器的重配命令后,在参数取值空间内搜索配置,对每个候选配置使用性能模型求解器得到在该配置时的性能预测值,以此选取最满足性能需求的配置.负载监视器统计在每个调整时间间隔内各个服务的请求数.模型构造器结合应用设计描述⑦,按照第 2.3 节中的方法为应用建立相应的性能模型⑧.性能剖析器在运行时度量各任务入口的CPU时间花费②,即性能模型的服务需求(service demand)参数.性能模型求解器根据服务需求②、服务器参数配置④(线程池大小、实例池大小等)及负载③作为输入,对模型进行求解,给出响应时间和吞吐量的预测值⑤.配置选择器根据预测值评估在此参数配置下的性能约束情况来衡量不同候选配置的"好坏",选取最优配置.在本框架中,性能模型求解器使用Carleton大学开发的分层排队网络求解工具LQNS[15].

3.2 配置选择算法

性能配置框架的目标是在负载变化的情况下,通过自适应地调整相关服务器参数配置,使得在满足各个服务的"硬约束"(constraint 1,constraint 2)的情况下,使"软约束"的效用值(constraint 3)达到最大,其中的关键是如何确定正确的配置参数.本框架的配置选择器采用爬山搜索算法,在服务器可配置参数的取值空间中寻找一个点,使得系统性能度量满足"硬约束"的要求,并且使QoS效用最大化.使用爬山算法是因为其实现相对简单,并且对于状态空间较小的情况(如本文目前只考虑 3 个参数),其收敛速度和其他智能搜索方法相比差别不大.

P维向量$C=(c_1,c_2,\ldots,c_P)$表示包含P个参数的配置,每个参数有一个给定的取值范围$c_i\in(c_i^{\min},c_i^{\max})$.$C$的相邻配置的集合为$M_C=\{C+l_i\}\cup\{C-l_i\}$,其中,$1\le i\le P$,$l_i$为$P$维向量$\{0,\ldots,1,\ldots,0\}$,第$i$个元素为 1,其余元素为 0.$C_0$为调整前的参数配置,此配置为搜索的初始中心点.每次迭代在中心点的相邻配置中选取最优配置作为下一次迭代的中心点.在搜索过程中,使用性能模型求解器(即predict函数)计算在每个候选配置下各服务的平均响应时间和平均吞吐量$\langle R,T\rangle$,其中,向量$R=(R_{s_1},R_{s_2},\ldots,R_{s_n})$为各服务$s_1,\ldots,s_n$平均响应时间的预测值,向量$T=(T_{s_1},T_{s_2},\ldots,T_{s_n})$为各服务$s_1,\ldots,s_n$平均吞吐量的预测值,以此可以通过Utility函数计算出在该配置下的效用值.HC为各服务"硬约束"的集合,satisfy 函数判断$\langle R,T\rangle$是否满足 HC 中的约束.

算法描述如下:

$Hops$=1;
C_{cur}=C_0;
C_{new}=C_0;
REPEAT
    $Improved$=FALSE;
    $\langle R,T\rangle$=$predict(C_{cur})$;
    Max$Utility$=$Utility(\langle R,T\rangle)$;
    FOR (each C in $M_{C_{cur}}$)
        $\langle R,T\rangle$=$predict(C)$;
        IF ($satisfy(\langle R,T\rangle,HC)$ AND $Utility(\langle R,T\rangle)$>Max$Utility$)
            Max$Utility$=$Utility(\langle R,T\rangle)$;
            C_{new}=C;
            $Improved$=true;
        ENDIF
    ENDFOR
    C_{cur}=C_{new};
    $Hops$=$Hops$+1;

UNTIL(*Improved*==FALSE OR *Hops*==Max*Hops*)

4 实 验

4.1 实验设计

我们在 OnceAS 应用服务器上对自配置框架进行了原型实现.为验证框架的性能保障效果,以 StockOnline 应用作为测试用例,比较了使用该框架和不使用该框架时,在负载变化情况下,系统的平均响应时间、吞吐量对于“硬约束”的满足情况以及“软约束”定义的效用值的变化情况.在实验中为简便起见,客户程序只访问 StockOnline 提供的两个服务:queryStockValueByID 和 updateAccount.queryStockValueByID 服务根据股票 ID 查询数据库,返回股票的价格.对这种查询性服务,系统应该具有快速的响应性.updateAccount 服务通过操作数据库来修改账户金额,属于应用的业务交易.相对而言,该服务吞吐量的要求更高,因为这意味着单位时间内能完成更多的交易量.同样原因,updateAccount 服务应该具有比 queryStockValueByID 更高的权重.相应的权重设置见表 1.

Table 1 Weights for performance requirement of the experiment

表 1 实验性能需求中的相关权重

Service	Service weight	Response time weight	Throughput weight
QueryStockValueByID (represented by q)	W_q=0.3	$w_q^R = 0.9$	$w_q^T = 0.1$
UpdateAccount (represeneted by u)	W_u=0.3	$w_u^R = 0.2$	$w_u^T = 0.8$

两个服务的最大响应时间和最小吞吐量为

$$R^q_{\max}=450\text{ms};\ T^q_{\min}=20\text{tps};\ R^u_{\max}=1500\text{ms};\ T^u_{\min}=50\text{tps}.$$

那么,性能“硬约束”为

$$R^q \le R^q_{\max} \tag{1}$$

$$T^q \ge T^q_{\min} \tag{2}$$

$$R^u \le R^u_{\max} \tag{3}$$

$$T^u \ge T^u_{\min} \tag{4}$$

应用的效用函数为

$$Utility = w_q \cdot \left(w_q^R \frac{R^q_{\max} - R^q}{R^q_{\max}} + w_q^T \frac{T^q - T^q_{\min}}{T^q_{\min}} \right) + w_u \cdot \left(w_u^R \frac{R^u_{\max} - R^u}{R^u_{\max}} + w_u^T \frac{T^u - T^u_{\min}}{T^u_{\min}} \right) \tag{5}$$

图 10 给出了设定的实验过程中各控制时间间隔单元内并发客户数的变化情况.

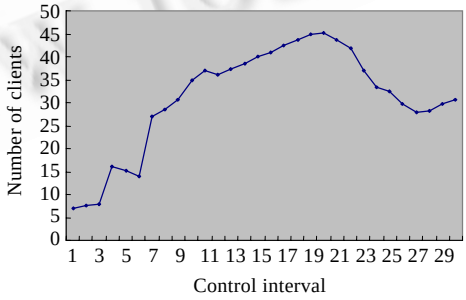


Fig.10 Variation of client number during the experiment

图 10 并发客户数变化

在两种测试情况下,服务器的初始配置都设置为一样,各个组件的容器参数配置都为$*M*=4,$*I*=6,数据库连接池的初始配置$*C*=5,并且,约束各参数的取值范围为 0≤$*M*≤50,0≤$*I*≤30,0≤$*C*≤50.

实验环境包括 3 台 PC 机.其中,一台 PC 机专门作为应用服务器,其配置为 Dell PowerEdge 2600,Dual CPU 3.04G,2G 内存;一台 PC 机为数据库服务器,其配置为 IBM Xseries 235,Single CPU 2.66G,2G 内存,Oracle8.1.6 数据库;另外一台 PC 机为用户测试机,其配置为 Dell OPTIPLEX GX260,CPU 2.66G,512M 内存.

4.2 实验结果

受篇幅所限,我们这里只给出 4 个“硬约束”中式(1)的比较图.图 11 给出了在两种测试情况下“硬约束”式(1)的满足情况.图 12 给出了 Utility 的变化情况.由图 11 和图 12 可以看出,在前 7 个时间间隔内,两种情况下的“硬约束”式(1)都得到了满足,而且 Utility 值差别不大.在第 9~第 20 时间间隔,当负载增加时,在没有使用自配置框架的情况下,响应时间延迟开始急剧增加,平均响应时间超过 450ms,“硬约束”式(1)不能保证,同时,Utility 值也开始下降.而对于使用该框架的情况下,通过调整参数配置,“硬约束”式(1)仍能得到保证,并且 Utility 值得到提高.

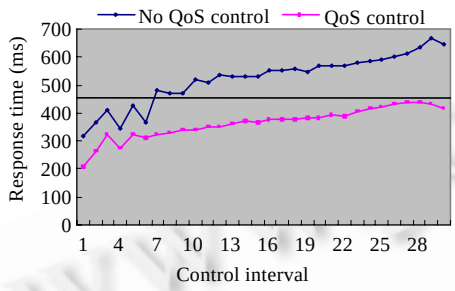


Fig.11 Response time constraint satisfaction comparison

图 11 响应时间约束满足情况比较

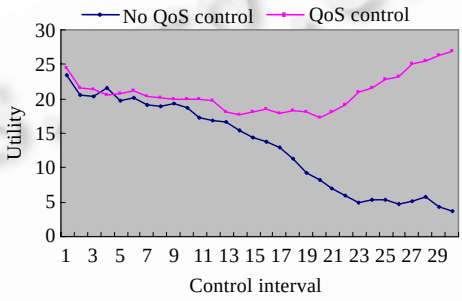


Fig.12 Utility comparison

图 12 Utility 值比较

5 相关工作

目前,在基于自配置的性能保障方面主要使用两种方法:一是基于控制论的方法,另外一种是基于性能模型的方法.在基于控制论的方法中大多使用反馈控制[5,16],通过比较性能指标的测量值和期望值之间的偏差给出相应的调控策略.在调控策略生成方式上有非线性控制、启发式控制、模糊控制等.对于基于控制论的方法,系统对控制器来说是一个“黑盒”,没有一个明确的机制来预先判断选取的调整策略的好坏,因而是一种“Try-Fail-Try”的调控模式,对于外界环境的变化需要反复多次的调整过程后才能满足要求,即所谓的“摆动”问题.基于性能模型的方法,使用各种性能建模方法对系统进行模型化,通过性能模型评估所选的调整策略,从而提高调整的准确性.如文献[9]使用排队网络方法对Web服务器进行性能建模,用于自适应地调整Web服务器的最大线程数.本文采用的是基于分层排队网络的性能建模方法,因为它能更好地描述多种资源并发竞争的情况.文献[9]和本文自配置框架的另一个区别在于,文献[9]的性能需求是针对单一的HTTP请求的性能约束,而本文的自配置框架则是在应用的业务服务级别上进行性能保障,性能模型更为复杂,所涉及的配置参数更多.

在过去的一些性能模型研究中,已有许多性能建模方法应用于基于组件的软件系统,如文献[11,12,17],这些方法主要侧重于从组件的交互关系上建立性能模型,如文献[12,17]提供了从 UML 到 LQN 模型的转换工具.然而,这些方法没有显式地考虑组件基础结构及其性能属性,忽略或极大地简化底层组件基础结构对性能的影响,因而这些方法缺乏准确性.在最近的一些研究中,中间件的性能模型开始得到重视.文献[18]最早分析了 EJB 容器的请求处理过程,给出了基于排队网络(QNM)的性能模型,但他们的工作只是对实体 Bean 性能模型进行了分析,没有考虑其他 Bean 类型的性能模型.在文献[18]对实体 Bean 容器的请求处理过程分析的基础上,文献[8]使用 LQN 方法对实体 Bean 进行了性能建模,随后,文献[19]使用排队网络模型对基于异步调用的消息驱动 Bean 进行了建模.但文献[8,18,19]存在的一个共同的不足之处是,它们只是在容器层面上分析单个组件的性能模型,没有考虑包含多个组件的应用的整体性能模型,对于实际的 EJB 应用性能预测意义不大.本文的建模方法通过模型分解/组合的方式,按照组件间的交互关系将各组件的性能子模型连接起来建立组件应用的整体性能模型.

6 结束语

本文提出了一个以基于分层排队网络的性能模型为核心的中间件性能自配置框架,能够在运行时动态地调整系统的资源配置参数.在配置决策过程中,框架通过性能模型预测值来评估各候选配置的好坏,指导在参数取值空间中搜索最优的配置组合,从而提高调控的准确性.本文主要以 EJB 平台作为研究对象,通过捕获 EJB 组件容器中的资源竞争和 EJB 组件之间的交互关系来建立性能模型.实际上,该性能建模方法可以用于其他组件平台,如 NET,CCM 等,因而本文所提出的自调整方法对这些组件平台也适用.下一步的工作包括两个方面:一是扩展性能模型,使之能够描述更多的资源配置;另一方面,将该框架用于其他组件平台上.

References:

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

[2] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30 (12A):1901−1906 (in Chinese with English abstract).

[3] Brebner P. Evaluating J2EE Application Servers-Version 2.1. CSIRO Publishing and Cutter Consortium. 2002.

[4] Raghavachari M, Reimer D, Johnson RD. The Deployer's problem: Configuring application servers for performance and reliability. In: Clarke L, ed. Proc. of the 25th Int'l Conf. on Software Engineering. Portland: IEEE Computer Society Press, 2003. 484−489.

[5] Bergmans L, Halteren A, Ferreira L, Sinderen M, Aksit M. A QoS-control architecture for object middleware. In: Boavida F, Monteiro E, Orvalho J, eds. Proc. of the 7th Int'l Workshop Interactive Distributed Multimedia Systems. LNCS 1905, Netherlands: Springer-Verlag, 2000. 117−131.

[6] Woodside M, Daniel A. Application-Level QoS. IEEE Internet Computing, 2006,10(3):13−15.

[7] Woodside M. Tutorial introduction to layered modeling of software performance. Carleton University, 2002. http://www.sce.carleton.ca/rads/lqn/lqndocumentation/tutorialg.pdf

[8] Xu J, Oufimtsev A, Woodside M, Murphy L. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. In: Tracz W, ed. Proc. of the Workshop on Specification and Verification of Component-Based Systems. New York: ACM Press, 2005.

[9] 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.

[10] Barthwal N, Woodside M. Efficient evaluation of alternatives for assembly of services. In: Siegel H, Bader D, eds. Proc. of the 19th Int'l Parallel and Distributed Processing Symp. Denver: IEEE Computer Society, 2005. 275−282.

[11] Bernardi S, Donatelli S, Merseguer J. From UML sequence diagrams and statecharts to analysable Petri nets models. In: Balsamo S, Inverardi P, eds. Proc. of the 3rd Int'l Workshop Software and Performance. Rome: ACM Press, 2002. 35−45.

[12] Cortellessa V, Mirandola R. Deriving a queueing network based performance model from UML diagrams. In: Woodside M, ed. Proc. of the 2nd Int'l Workshop Software and Performance. Ottawa: ACM Press, 2000. 36−55.

[13] 2005. http://www.once.com.cn

[14] Quest Software, Inc. JProbe profiler: Developer's guide version 5.2.2. 2004. http://questsupportlink.quest.com/support_downloads/Resources/JProbe/docs/JProbeProfilerGuide.pdf

[15] Franks G, Hubbard A, Majumdar S, Petriu DC, Rolia J, Woodside CM. A toolset for performance engineering and software design of client-server systems. Performance Evaluation, 1995,24(1-2):117−135.

[16] Li B, Nahrstedt K. A control-based middleware framework for quality of service adaptations. IEEE Journal Selected Areas in Communication, 1999,17(9):1632−1650.

[17] Gu PG, Petriu DC. XSLT transformation from UML models to LQN performance models. In: Balsamo S, Inverardi P, eds. Proc. of the 3rd Int'l Workshop Software and Performance. Rome: ACM Press, 2002. 227−234.

[18] Llado CM, Harrison PG. Performance evaluation of an enterprise Java Bean server implementation. In: Woodside M, ed. Proc. of the 2nd Int'l Workshop Software and Performance. Ottawa: ACM Press, 2000. 180−188.

[19] Liu Y, Gorton I. Performance prediction of J2EE applications using messaging protocols. In: Heineman G, ed. Proc. of the 8th Int'l SIGSOFT Symp. on Component-Based Software Engineering. St. Louis: ACM Press, 2005. 1−16.

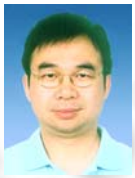
附中文参考文献:

[1] 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

[2] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12A):1901−1906.

胡剑军(1979－),男,湖北黄梅人,博士生,主要研究领域为网络分布式计算.

官荷卿(1980－),男,博士生,主要研究领域为网络分布式计算.



魏峻(1970－),男,博士,研究员,主要研究领域为软件工程,网络分布式计算.

黄涛(1965－),男,博士,研究员,博士生导师,主要研究领域为软件工程,网络分布式计算.