# OnceAS/Q:一个面向 QoS 的 Web 应用服务器*

黄 涛[1,2+], 陈宁江[1,2,3], 魏 峻[1], 张文博[1,2,3], 张 勇[1,2,3]

[1](中国科学院 软件研究所 软件工程技术研发中心,北京 100080)

[2](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[3](中国科学院 研究生院,北京 100039)

## OnceAS/Q: A QoS-Enabled Web Application Server

HUANG Tao[1,2+], CHEN Ning-Jiang[1,2,3], WEI Jun[1], ZHANG Wen-Bo[1,2,3], ZHANG Yong[1,2,3]

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

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

[3](Graduate School, The Chinese Academy of Sciences, Beijing 100039, China)

+ Corresponding author: Phn: +86-10-62553570, E-mail: tao@otcaix.iscas.ac.cn, http://www.iscas.ac.cn

Received 2004-07-06; Accepted 2004-09-06

**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.**

http://www.jos.org.cn/1000-9825/15/1787.htm

**Abstract**: With the increasing diversity and complexity of distributed systems in network environment, web application server (WAS) is required to transform its service model from "Best Effort" to "QoS (quality of service) Guarantee". However, most existing WASs are relatively weak in terms of QoS provision. In this paper, OnceAS/Q is presented to show how a QoS-enabled WAS can offer different QoS provisions for applications. OnceAS/Q provides a set of QoS related services and a framework to support QoS guarantee, in order to realize a QoS enabled WAS. This paper first describes the architecture and main components of OnceAS/Q. Then two key issues, related to QoS enabling and the corresponding solutions, are explored in detail: one is the definition and mapping of QoS specifications, and the other is the dynamic reconfiguration of QoS-aware service components and resources. Ecperf benchmark, a well-known performance benchmark for J2EE Application Server, is used to evaluate the effect of QoS enabling of OnceAS/Q. Experimental results show that OnceAS/Q can support better QoS for applications at a reasonable cost.

**Key words**: Web application server; QoS; middleware

**摘 要**: 网络分布计算环境下应用系统的需求多样化和复杂性的增长,要求位于中间件层次的 Web 应用服务器

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2004AA112010, 2003AA414310 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 (国家重点基础研究发展规划(973))

作者简介: 黄涛(1965－),男,江苏淮阴人,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;陈宁江(1975－),男,讲师,主要研究领域为软件工程,网络分布式计算;魏峻(1970－),男,博士,副研究员,主要研究领域为软件工程,网络分布式计算;张文博(1976－),男,博士生,主要研究领域为网络分布式计算;张勇(1975－),男,博士生,主要研究领域为网络分布式计算.

(web application server,简称 WAS)从原来的“尽力而为”服务模型转变为服务质量(quality of service,简称 QoS)保障模型,为具有不同需求的应用分别提供适宜的服务质量保障.目前的 WAS 系统在此方面仍然比较薄弱.OnceAS/Q 是一个面向 QoS 的 WAS 系统,它以 QoS 规约为基础,为不同应用提供不同的 QoS 保障能力.OnceAS/Q 实现了一个应用 QoS 保障框架,提供了一组 QoS 服务组件支持具有 QoS 需求的应用开发和运行.介绍了 OnceAS/Q 的体系结构和主要组件,详细阐述了两个关键问题,一是 QoS 规约的定义及其映射,另一个是面向 QoS 的服务组件和资源的动态重配.OnceAS/Q 原型在 Ecperf 测试基准下,对其 QoS 保障能力进行了实验.实验数据表明,在较大规模的应用环境下,OnceAS/Q 能够提供更好的服务质量,并且开销是可接受的.

**关键词**: Web 应用服务器;QoS;中间件

**中图法分类号**: TP311 **文献标识码**: A

Web 应用服务器(Web application server,简称 WAS)已经成为网络分布计算环境下的主流中间件[1,2].目前的 WAS 系统大多是遵循 J2EE(Java2 platform enterprise edition)[3]规范实现,它们从功能性方面为应用提供了基础运行支撑服务,但是对应用非功能性方面的支持比较薄弱,特别是在服务质量(quality of service,简称 QoS)保障方面.与应用服务器相关的 QoS 概念是从网络、多媒体等领域引申出来的.如同早期的网络/多媒体系统一样,目前的 WAS 基本采用“尽力而为(best effort)”服务策略为应用提供服务,即对所有应用请求一视同仁,尽最大努力地分配任务和资源.然而,Internet 环境的高度开放、动态演变的本质特征,以及应用需求的复杂多样和快速增长,使得 WAS 的“尽力而为”服务模型很难满足应用的高可靠、实时、安全等需求.因此,WAS 需要转换服务策略,随需而变,为有不同需求的应用提供合适的服务质量,如业务吞吐量、响应时间、可靠度等.

例如,在一个复杂的电子商务系统中,会包含不同规模的子系统,也可能并存多种类型的资源,视频/音频文件显然具有与文本文件不同的资源需求.当发生大规模的用户并发访问时,如果 WAS 采用尽力而为、先来先处理的策略,那么具有高负载性质的请求(如多媒体资源请求)可能会先被 WAS 尽力处理,造成系统在一段时间内过于繁忙而导致阻塞甚至崩溃;或者某些低级别的用户请求可能会占用过多的资源,从而会影响 WAS 对其他更重要的应用请求的响应.因此,WAS 如果具有对 QoS 特性的处理和优化能力,使应用系统的用户具有良好的使用感受,这对体现应用的价值有着重要意义.

面向 QoS 的 Web 应用服务器是指 WAS 能够根据应用对 QoS 的需求,以及运行环境情况,遵照 QoS 规约为不同应用提供不同的 QoS 保障,并且在运行时动态调整 WAS 的 QoS 保障行为.我们在自主研发的 J2EE 应用服务器——OnceAS[4]的基础上,研究了提供随“需(QoS)”而变能力的应用服务器体系结构、QoS 规约描述和不同层次的映射,以及根据 QoS 需求和运行时状态,动态重配 WAS 的服务组件和资源的机制.

本文第 1 节介绍 OnceAS/Q 的体系结构及主要组成部分.第 2 节阐述 OnceAS/Q 的 QoS 规约的定义和描述方法.第 3 节介绍 OnceAS/Q 的动态调整过程,包括服务组件和资源的动态适应机制.第 4 节给出了对 OnceAS/Q 的实验及其结果.第 5 节对相关的研究工作进行讨论.最后是对本文工作的总结.

## 1 面向 QoS 的 Web 应用服务器框架

### 1.1 OnceAS应用服务器对服务质量保障的不足

应用服务器 OnceAS 遵循 J2EE 1.4 规范,体系结构如图 1 所示,其中主要部件包括:EJB 容器和 Web 容器,它们为应用组件(EJB,Servlet 等)提供生命期管理和运行时服务;系列基础服务组件,包括命名、安全、事务等,为 WAS 和应用组件的运行提供支持;管理内核(management kernel)为基础服务组件提供了统一的管理和配置框架,进行服务组件的生命期管理和运行时控制;容器入口(invocation entry)响应客户的请求;拦截器(interceptor)[5](若干个拦截器组成拦截器链)在请求处理路径上拦截调用上下文(InvocationContext),OnceAS 根据上下文信息触发相应的服务组件处理特定的任务.

OnceAS 体系结构具有一定的可扩展性和可重配性[4,6]:(1) 通过 Interceptor 机制,WAS 在不改变内核的情况下能够灵活地增加、改变和撤销各种运行时服务;(2) 管理内核对各个服务组件实施了灵活的生命期管理,提供

了一个可扩展的服务管理和控制框架,能够动态地配置和替换服务组件.

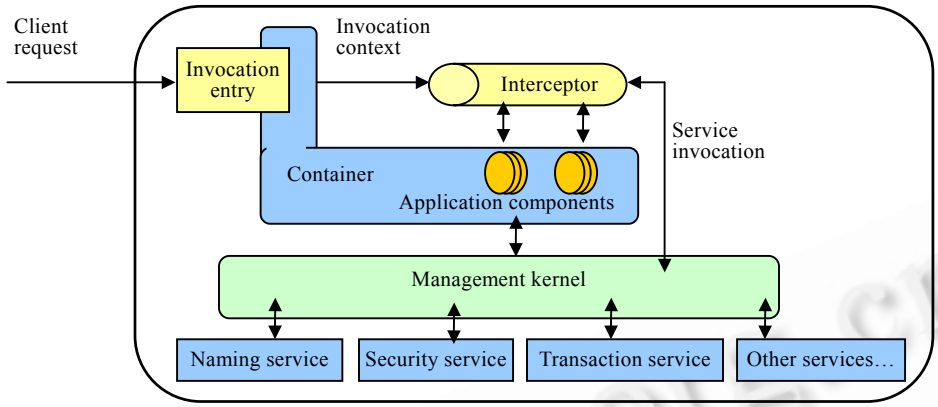



Fig.1 Architecture of OnceAS

图 1 OnceAS 的体系结构

但是,面对灵活的 QoS 保障需求,OnceAS 存在的不足主要表现在:(1) 应用开发方面缺乏 QoS 需求描述方法,OnceAS 也没有提供相应的 QoS 映射设施;(2) 由于 Java 虚拟机和 J2EE 的局限,OnceAS 还缺乏描述和管理系统资源的良好方法和设施;(3) 虽然 Interceptor 模式运用和 Java 简单反射机制[7]为实现组件运行时重配提供了一定的支持,但对于面向 QoS 的系统重配来说依然不够,需要有更高层次的动态适应机制.

因此,面向 WAS 的随"需(QoS)"而变的需求,围绕以上问题,我们对 OnceAS 进行了扩展,研究和设计了一个面向 QoS 的应用服务器——OnceAS/Q (QoS-Enabled OnceAS).

### 1.2 OnceAS/Q系统结构

#### 1.2.1 随需而变的服务模型

面向QoS的WAS是指在可用资源的约束下,能够提供满足应用特定需求的服务能力,即具有随需而变的服务模型,如图 2 所示.这可视作一个控制-反馈模型.

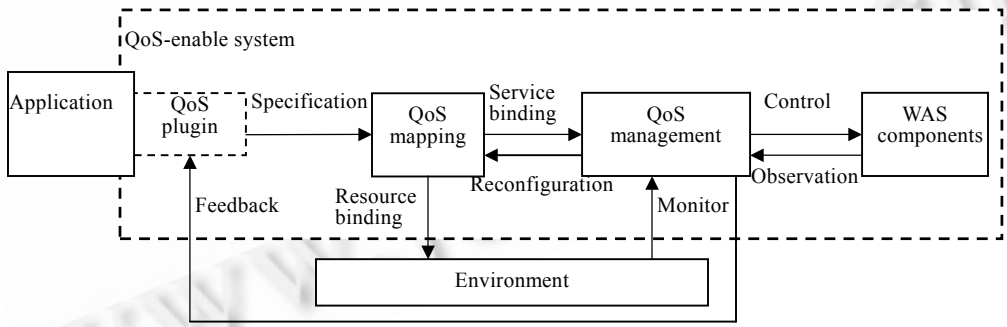



Fig.2 The QoS-enabled on demand service model

图 2 随需而变的服务模型

在这样的服务提供模型中,首先应用对服务质量的要求以WAS可理解和处理的方式表示出来,即QoS规约(表示应用与服务器之间的服务保障契约),WAS 的服务组件和资源的配置及调整都遵照该规约进行.这要求有完善的 QoS 规约描述方法,以适合 WAS 的特点和便于实现 QoS 管理.QoS 映射(QoS mapping)使应用与 WAS 之间建立起 QoS 关联,包括面向需求的服务和资源的配置和绑定.WAS 根据 QoS 映射驱动服务组件和服务器资源的重配.

其次,QoS 管理(QoS manangement)是在运行时进行 QoS 管理和控制.它监测环境(environment)的状态,向 WAS 组件(包括容器、基础服务组件、管理内核等)发出服务和资源重配置的命令.环境包含了 WAS 所使用的

底层资源.环境的变化会反馈给 QoS 管理过程,WAS 组件在运行时通过 QoS 管理按需改变配置和行为.QoS 管理是随需而变服务模型的核心,其主要任务有:QoS 协商控制——运行时动态修改 QoS 规约,驱动服务组件和资源的动态重配;QoS 准入控制——根据 QoS 请求和当前系统状态,决定是否允许当前应用进入执行状态;QoS 维持控制——确保应用组件所需的服务组件和资源,维持应用的 QoS 级别.同时处理用户违反 QoS 规定的行为,给出服务中止或者降级等反馈.

WAS 的 QoS 调整会对应用产生一定影响,可能要求应用也相应地调整.为了透明地在应用客户端动态加入 QoS 控制,使应用业务逻辑与 QoS 服务逻辑有效分离,在应用端引入 QoS 插件是一个好的方法,插件中包含需要在客户端处理的 QoS 相关逻辑.

我们在 OnceAS 体系结构的基础之上,以扩展的方式增加随需而变的服务模型的机制,同时也考虑充分利用 OnceAS 原有的可扩展机制等特点,保证了原有组件的复用性和应用的兼容性.

### 1.2.2 OnceAS/Q 的主要结构

按照上述 QoS 随需而变的服务模型,OnceAS/Q 提供了一组 QoS 服务组件(简称为 QoS 组件),支持具有 QoS 需求的应用开发和运行,这些 QoS 组件一方面接受管理内核的生命期管理和配置,另一方面通过相互协作和管理内核来配置其他基础服务组件的行为,为应用与 WAS 服务组件和资源之间建立有效的 QoS 映射,在运行时基于 QoS 需求适应性地调配服务组件和资源.OnceAS/Q 的系统结构如图 3 所示,按照功能不同,QoS 组件分为 QoS 映射、QoS 管理和客户端 QoS 支持等 3 类.

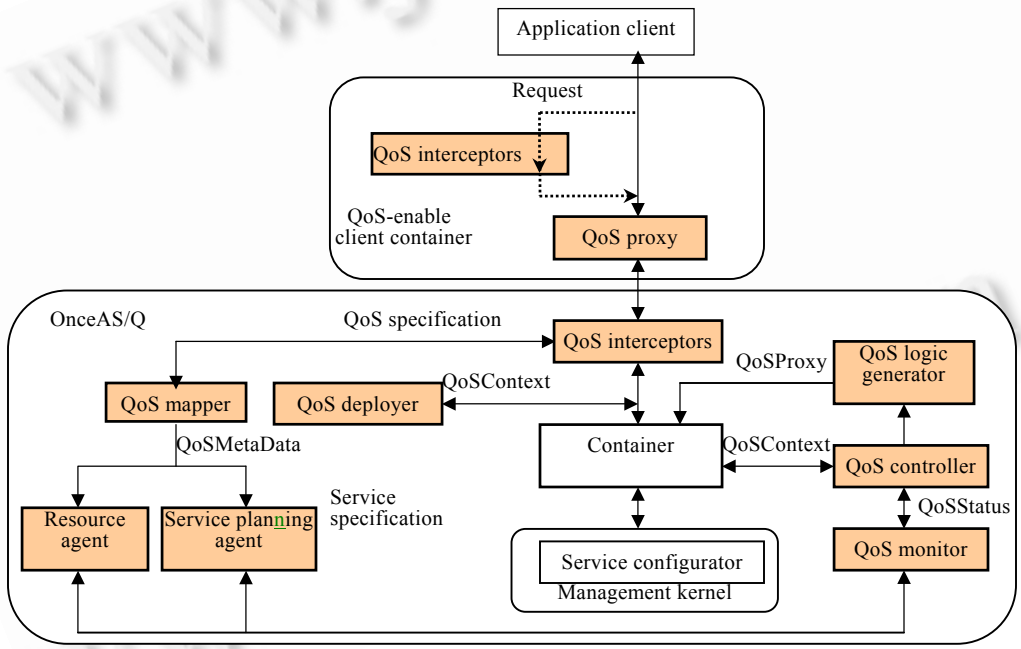



Fig.3 The architecture of OnceAS/Q

图 3 OnceAS/Q 系统结构

(1) QoS 映射组件

QoS 映射组件在应用部署时或运行时进行 QoS 规约的解析和映射.拦截器(QoS interceptor)拦截获得的 QoS 规约,生成 QoS 元数据(QoSMetadata),其中包含 QoS 请求级别、服务组件和资源配置等初始信息.QoS 映射器(QoS mapper)负责解析 QoS 规约.服务规划代理(service planning agent)与管理内核的服务配置器(service configurator)进行协调,根据 QoS 元数据来确定满足 QoS 请求的服务组件集合.资源代理(resource agent)对资源的状态进行监控,并根据 QoS 元数据形成满足当前 QoS 请求的资源配置,进行资源分配.

(2) QoS 管理组件

QoS 映射结果封装为 QoS 上下文(QoSContext),QoS 管理组件的操作将以 QoSContext 为依据.QoS 部署器

(QoS deployer)负责部署有 QoS 需求的应用组件,它根据 QoS 上下文,生成与应用相关联的服务组件和资源实例的集合.

QoS 控制器(QoS controller)是实施 QoS 管理的核心部件,主要处理 QoS 协商控制、QoS 准入控制和 QoS 维持控制等任务.QoS 监视器(QoS monitor)监视和感知 WAS 的状态,将整个系统的资源使用情况、运行状态等形成 QoS 报告(QoSStatus),定期反馈给 QoS 控制器.QoS 逻辑生成器(QoS logic generator)动态生成 QoS 相关的处理逻辑,将它和相关配置信息封装进 QoS 代理(QoS proxy)中;在容器返回应用请求处理结果时,QoS 代理对象被反馈到客户端,与 QoS 客户端容器动态组装起来,负责处理需在客户端进行的 QoS 服务逻辑.

(3) QoS 客户端容器(QoS-enable client container)

OnceAS/Q 在客户端部署 QoS 客户端容器.它作为应用与 WAS 的中介,充当 QoS 插件的角色.客户请求先经过 QoS 客户容器进行 QoS 相关处理,然后再传给服务器.QoS 客户容器主要包含两类 QoS 功能组件:QoS 代理(QoS proxy)和客户端 QoS 拦截器.

## 2 OnceAS/Q 的 QoS 规约

### 2.1 Web应用服务器的QoS规约

由于 QoS 规约所描述的对象和内容的多样性,关于 WAS 的 QoS 规约可从多个视图来描述,如图 4 所示.从物理层次视角看,包括应用层、容器层、WAS 层、节点层和网络层.各个层次都可以定义面向该层次实体的 QoS 规约,不仅定义的内容有不同要求,而且每层次之间存在一定的映射关系.从内容视角看,QoS 规约包括 3 个部分:(1) QoS 需求规约(QoS requirement specification,简称 QRSpec),从应用视角描述期望的服务质量性质,如期望的服务质量级别,可靠性、可伸缩性、性能等可度量的 QoS 参数及其值域等;(2) QoS 服务规约(QoS service specification,简称 QSSpec),规定 WAS 为满足应用 QoS 需求所需的服务组件集合及其可配置属性、服务行为和服务依赖关系等;(3) QoS 资源规约(QoS reSource specification,简称 QRSSpec),它描述满足 QoS 需求和服务规约所需的资源类型、资源配置参数及其值域、资源分配策略、协商策略等.

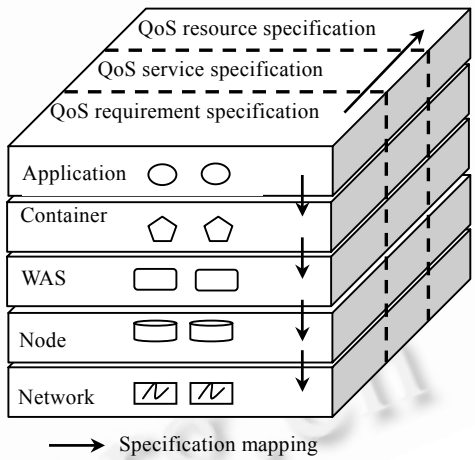



Fig.4 Multiple views of QoS specification
图 4 多视图的 QoS 规约

QoS 规约可以在应用设计阶段被定义,也可以在运行时动态生成或者修改,但是不同 QoS 规约在应用生命周期的重要性及其对 QoS 保障能力的影响是不一样的.例如,在应用设计阶段主要定义 QoS 需求规约,而服务规约和资源规约主要面向运行阶段.虽然也可以在设计时定义资源规约,但资源规约的准确性和完整性对用户不做较高的要求,而且 WAS 在进行 QoS 协商时,运行时生成的资源规约比设计时的处理优先级要高.

### 2.2 OnceAS/Q的QoS规约描述

WAS 关注的 QoS 性质有很多,可扩展的 QoS 规约描述方法是非常重要的.J2EE 平台以使用部署描述符文件的方式来定义应用和 WAS 之间的安全、应用环境变量、组件关系等信息.为了与 J2EE 平台兼容,以及提供简洁而可扩展的描述手段,OnceAS/Q 基于 XML 语言描述 QoS 规约,定义了 QoS 描述符(QoS descriptor,简称 QD).QD 将 QRSpec,QSSpec,QRSSpec 等部分集成起来,为各个层次的 QoS 规约提供了统一表示,图 5 给出了 QD 的 xml 模式结构.

在 QD.xml 中,*Tier* 标签定义规约的层次,即应用层、WAS 层、容器层等.*RequirementSpec* 标签定义需求规约部分,它的结构遵循以下 QoS 性质描述框架:类别(category)给出高层次的服务质量的类型描述;每个类别由若干属性(property)来描述该类别的各个方面;每个属性由若干量度(metric)给出定性或定量的指标;每个度量又由

若干参数(parameter)及其取值来表示.*ServiceSpec* 标签定义服务规约部分,包括服务名称、服务依赖关系定义(dependency)、实现类、参数配置集等.*ResourceSpec* 标签定义资源规约部分,包括资源类型、该类 QoS 监控器实现、资源配置参数等.为了使各个部分建立有机的联系,OnceAS/Q 定义了 QoS 角色(QoSRole)的概念.*QoSRole* 是一个逻辑概念,它标识了一组具有相同 QoS 需求的用户或者应用.QoS 需求规约、服务规约和资源规约以 *QoSRole* 为线索组织和关联起来.

QD 描述符作为标准 J2EE 部署描述符的补充,其在应用部署过程中先后由 QoS 映射器和 QoS 部署器解析和处理,生成 QoS 元数据.

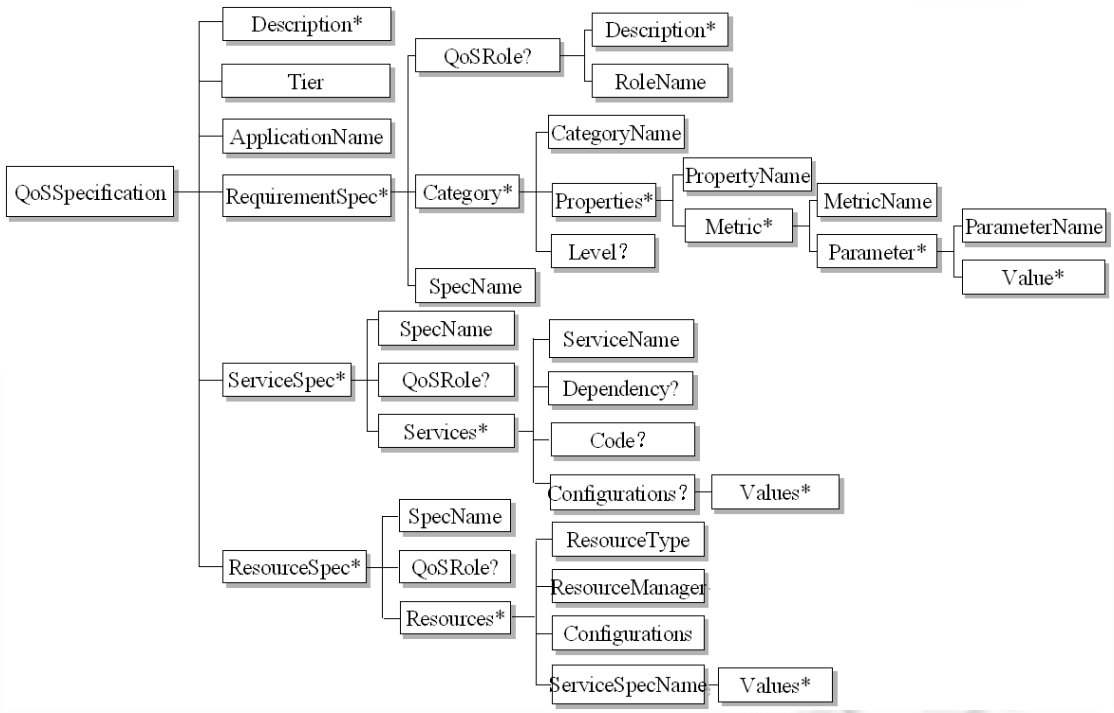



Fig.5 The structure of QoS descriptor

图 5 QoS 描述符文件的结构

## 3 OnceAS/Q 运行时的 QoS 管理与控制

OnceAS/Q 根据运行环境和应用的 QoS 需求动态地调整服务质量,这包括两个方面的问题:一是服务组件的动态配置,二是资源的动态分配.

### 3.1 服务组件的动态配置

OnceAS/Q 使用管理内核作为 QoS 服务组件的管理框架.QoS 服务组件与 WAS 的基础服务组件一样,都通过统一接口向管理内核注册.管理内核中的服务配置器(service configurator)负责生成和维护服务组件之间的依赖关系.各个服务组件之间的交互通过管理内核传递和响应“通知(notification)”来进行.

面向 QoS 服务重配的关键是如何区分不同应用的 QoS 重配请求,使不同应用所使用的服务集互相不干扰,即保证服务重配的一致性.在 OnceAS/Q 中,采取了应用服务空间区分的方法.

**定义 1**. 应用 QoS 环境(application QoS environment,简称 AQE). AQE 是与特定 QoS 角色(QoSRole)绑定的服务和资源配置.AQE=(*QoSRoleID*,*Qservices*,*Qresources*,*S2R*,*SDT*),其中,

- *QoSRoleID* 标识与具体应用绑定的 QoSRole;
- *Qservices* 是与 QoS 角色绑定的服务集合,QoS 映射器在解析 QoS 规约时生成;

• *Qresources* 是与 QoS 角色绑定的资源集合,QoS 映射器在解析 QoS 规约时生成;

• *S2R*: *Qservices*×*Qresources*,定义了服务与资源的映射关系;

• *SDT*: *Qservices*×*Qservices*,定义了服务依赖关系,是非自反的.*SDT* 实际给出了与 QoS 角色绑定的服务依赖树.

WAS 管理内核依据 AQE 为不同的 QoSRole 创建服务组件实例.不同 QoSRole 的 AQE 是相互分离的,因此,不同 QoSRole 拥有不同的服务组件实例.

在 WAS 中,存在着全局唯一的服务组件,即在运行时只能有一个服务组件实例存在,因此,nceAS/Q 设置一个单独的全局 QoS 环境(global QoS environment,简称 GQE)对全局服务组件进行管理.

**定义 2**. 全局 QoS 环境.GQE=(*Gservices*,*ServiceLinks*),其中,

• *Gservices* 是全局唯一的服务组件集合;

• *ServiceLinks*: AQE.*Qservices*×*Gservices*,是应用 QoS 环境中的服务与全局服务关联的关系,用以标识全局服务组件的不同配置实例.

AQE 与 GQE 将具有不同 QoS 需求的应用所使用的服务组件和资源实例区分开来,保证了为不同应用提供不同的 QoS 配置,从而避免了可能的服务配置冲突.AQE 被封装在 QoSContext 中,在服务重配过程中由管理内核、QoS 控制器等操纵.图 6 是一个 AQE 与 GQE 的示例.

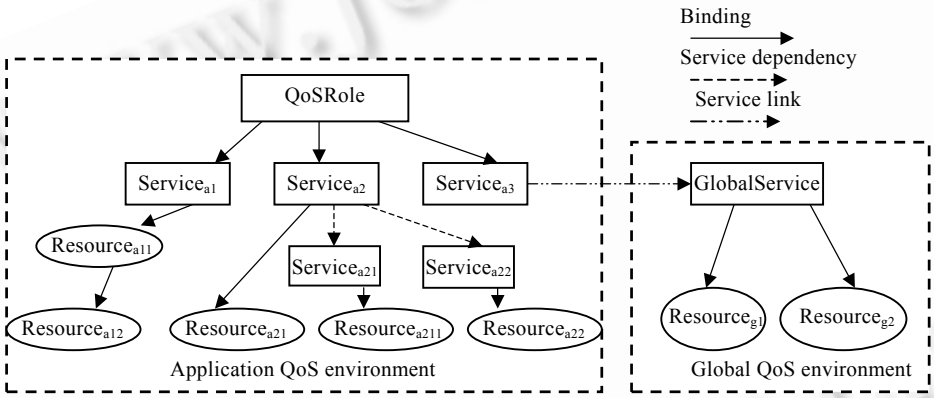



Fig.6 AQE and GQE

图 6 AQE 与 GQE 示例图

OnceAS/Q 的服务重配或者由 QoS 拦截器拦截应用请求而触发,或者由 QoS 控制器根据运行时状态触发,服务重配过程如下:

(1) QoS 拦截器拦截应用请求中的 QoS 需求(或者 QoS 控制器根据 QoS 监控器的报告生成新的 QoS 需求),将其封装到当前 QoSContext;

(2) QoS 控制器向管理内核发出“QoS 服务重配”通知,并将当前 QoSContext 传递给管理内核;

(3) 服务配置器从 QoSContext 中获得新的服务组件配置和调整策略;

(4) 对服务组件配置中的每个服务组件逐个处理:如果服务存在服务依赖关系,则构造出其服务依赖树,从树根依次对各个服务进行处理;

(5) 根据服务调整策略的不同类型,对服务分别做如下处理:

(5.1) “重置(reset)”策略:首先阻塞服务组件的执行;通过序列化、持久化或者日志(logging)等机制暂存当前服务的状态;重新设置服务组件的参数;将服务组件从暂存状态恢复到执行状态;

(5.2) “停止(stop)”策略:将服务状态写入日志;通知管理内核,停止该服务组件的运行;释放本组件占用的资源;依据服务依赖关系,管理内核通知其相关的服务组件除去与该服务的关联;

(5.3) “新增(append)”策略:通知管理内核装载所需服务组件,并生成组件实例;从 QoSContext 中获取 QoSRole 和应用 QoS 环境(AQE),装载新生成的服务组件实例;根据 QoS 规约配置新服务组件的参数;

(5.4) “替换(replace)”策略:首先执行(5.2)停止该服务组件;然后执行(5.3)增加新服务组件;

(6) 服务配置器生成新的服务依赖树;

(7) QoS 控制器发出调整系统资源的通知;

(8) 生成新的应用 QoS 环境(AQE),返回新的 QoSContext.

### 3.2 资源的动态分配

运行时的服务组件重配与系统资源的动态重分配紧密相关.一个运行时系统所使用的资源归根到底是内存、CPU、网络带宽、磁盘等类型.但是,一般用户和应用系统对于这些底层资源的配置以及需求是不敏感的,难以进行资源的按需管理.为此,WAS 应该为应用提供一定的资源管理透明性.OnceAS/Q 增强了关于系统资源监控、资源管理和分配策略等方面的支持机制.在 OnceAS/Q 中,建立了可控资源的统一视图,实现了一个扩展的资源管理框架,支持资源抽象、监控和管理.同时,为优化资源的分配设计了一个动态资源配置算法.它基于 QoS 资源规约和运行时资源状况,在各种资源配置之间进行权衡,寻找优化的资源配置方案.

#### 3.2.1 OnceAS/Q 的二级资源管理框架

OnceAS/Q 的系统资源抽象为两类:逻辑资源(logical resource)是指 WAS 能直接感知和控制的资源类型,如组件池/缓存、消息队列、对象锁等;物理资源(physical resource)是指内存、CPU、线程等底层资源. OnceAS/Q 更关注和更能有效控制的资源是“逻辑资源”,这也是应用所更能理解的资源类型.

OnceAS/Q 的资源管理建立在面向逻辑/物理资源的两级资源抽象模型之上.在低层资源的基础上构建高层的资源视图,即物理资源是对底层实际资源的抽象,而逻辑资源与物理资源之间存在一定的映射.该抽象模型便于对不同类型的资源进行统一管理,以及不同资源之间进行映射.

为了在系统运行时能够满足不同服务质量的需求,各个应用和服务组件所使用的资源要避免冲突.OnceAS/Q 通过 QoS 角色与应用 QoS 环境(AQE)的绑定,将应用与其所使用的运行时资源对象关联起来.不同 QoS 角色所使用的逻辑资源实例不同;每个逻辑资源映射和分配到若干物理资源对象.应用对逻辑资源进行直接的配置和操作,而逻辑资源又映射到物理资源,从而实现与底层环境资源的交互.

OnceAS/Q 实现了这样的资源管理框架,称之为面向 QoS 的抽象资源管理框架(Qos-Enable abstract resource management framework,简称 QARMF),如图 7 所示.QARMF 包括两个主要部分:(1)统一的资源表示(resource representation),提供了对资源的表示和访问接口;(2)资源管理(resource management)类组件.

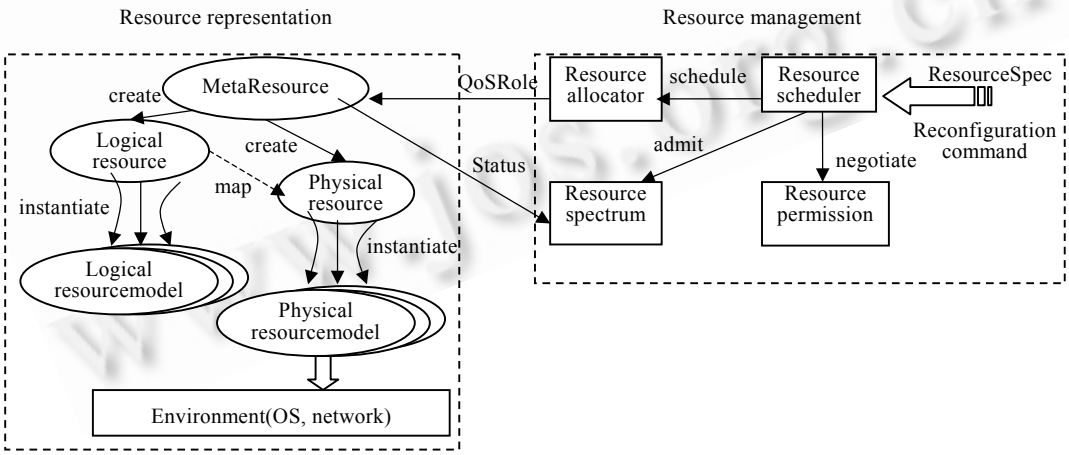



Fig.7 Qos-Enabled abstract resource management framework

图 7 面向 QoS 的抽象资源管理框架

资源表示中的 MetaResource 是抽象资源的元接口,定义了监视资源状态,获知 QoS 规约等操作.LogicalResource 和 PhysicalResource 分别定义逻辑资源和物理资源的描述和存取方法,逻辑资源根据资源规约映射成特定的物理资源配置.在 LogicalResource 和 PhysicalResource 的基础上,可以分别实现各种逻辑/物理

资源的具体类(称为资源模型类).例如,属于逻辑资源的组件池(ComponentPoolModel)、连接池ConnectionPoolModel)、锁(LockModel)等;属于物理资源的内存(MemoryModel)、文件资源(FileModel)等.各个资源模型类中分别实现监视和分配各自对应资源的方法.

资源管理组件包括资源调度器 ResourceScheduler、资源分配器 ResourceAllocator、资源许可ResourcePermission、资源配额 ResourceSpectrum 等,它们都是 QoS 监控器组件(如图 3 所示)的主要组成部分,根据传来的资源规约和资源重配命令,相互协作完成资源的协商、分配和回收等任务.

OnceAS/Q 在运行时资源重配的过程一般为:(1) QoS 监控器接收资源重配请求,获取 QoSContext;(2) QoS 监控器从 QoSContext 中取出当前的 QoS 角色及其资源规约,计算预期的资源分配表;(3) ResourceSpectrum 获得当前全局资源状态,计算当前可用资源;(4) QoS 监控器通过与 ResourcePermission,ResourceSpectrum 的协作,进行资源协商,生成在当前运行环境下能满足资源规约的最佳资源配置参数;(5) ResourceAllocator 与资源代理协作,通过 LogicResource 的具体实现模型,进行逻辑资源对象的分配,包括设置资源的参数值、申请新的资源、释放资源、缓存资源等;(6) 逻辑资源模型的配置变化驱动物理资源模型的重配,进而完成底层资源的重新分配.

3.2.2 资源配置的动态计算

资源的重配归结到各种资源的具体参数配置.虽然应用在 QoS 资源规约中规定了所使用的资源的期望数值,但是服务器需要在考虑运行时的资源状态、其他应用对资源的需求的条件下,综合权衡,找到当前系统所能容许范围内的最佳资源配置参数集.

设 QoS 度量(metric)集合 $M=\{m_1,m_2,\ldots,m_k\}$,所有资源可配置参数为 $P=\{p_1,p_2,\ldots,p_n\}$.资源代理(如图 3 所示)负责进行各种 QoS 指标 $M$ 与资源配置参数 $P$ 之间的映射.一个 $m_i$ 可由若干个 $p_j$ 表示.对于度量 $m_i$,根据 QoS 规约得到的 $m_i$ 的最大值和最小值分别为 $\max(m_i)$和 $\min(m_i)$,将此二值中表示最低质量保障效果的值记为参考值 $ref(m_i)$,而另一值记为最佳值 $opt(m_i)$.例如,对于响应时间而言,参考值 *ref* 取 max 值,最佳值 *opt* 取 min 值.在此基础上,定义如下函数:

(1) 协商函数 $g_m(\delta)$: $g_m(\delta)=a\times\delta+b$.函数 $g_m(\delta)$表示计算 QoS 度量 $m$ 的函数,其中 $a,b$ 为正系数,$\delta$为搜索 $m$ 的最佳值时的步长,$\delta=\lambda\times|m_{probe}-ref(m)|/ref(m)$,$m_{probe}$ 为 QoS 监测器定期检测获得的 QoS 度量值,$\lambda$为步长调节系数.

(2) QoS 优化函数 $Q_{optimize}$:

$$Q_{optimize}=\sum_{i=1}^{k}w_i\times g_{m_i}(\delta).$$

其中 $w_i$ 是度量 $m_i$ 的权重,定义各种度量的重要性,$\sum_{i=1}^{k}w_i=1$.

OnceAS/Q 在运行时以探测的 $m_{probe}$ 计算 $Q_{optimize}$ 的最佳值,确定各项 QoS 度量 $m_i$ 的取值,然后通过资源代理在资源配置参数 $P$ 中进行映射,将 $P$ 反馈回 QoS 监控器以作为配置和调整资源的依据.

目前,OnceAS/Q 采用简单的爬山法(hill-climbling)[8]进行 $Q_{optimize}$ 最佳值求解,搜索边界为 $\left[\sum_{i=1}^{n}w_i*ref(g_i(\delta)),\sum_{i=1}^{n}w_i*opt(g_i(\delta))\right]$.爬山法虽然比较简单,但是容易实现,在实际中能较有效地找到解.

## 4 实验结果

OnceAS/Q 原型系统已经实现.为了检验 OnceAS/Q 的 QoS 提供能力,我们对该原型进行了一组实验.测试工具选用业界熟知的 J2EE 应用服务器性能测试基准 ECperf[9].ECperf 重点考察 EJB 容器以及基础服务组件的处理能力,它包括多种 QoS 度量指标,如请求平均响应时间、不同业务类型的比例、吞吐量等.ECPerf 所模拟的应用规模由事务注入率(TxRate)决定,并发客户数与 *TxRate* 成正比.ECPerf 测试的实验环境为:100M Ethernet 局域网,受测 OnceAS/Q 系统所在机器的配置为 DELL PowerEdge 2000 PC 服务器(Intel XEON CPU 3.04GHz×2,2GB 内存).

实验为 ECPerf 应用设定了 3 个服务级别:Level 0——不启动 QoS 管理机制;Level 1——期望的业务处理量

要求较低;Level 2——期望的业务处理量要求较高.根据服务级别,分别给出了不同的初始逻辑资源约束,见表 1.表中所列出的是对 Ecperf 应用具有重要影响的资源,其中 *Size*=[min*Size*,max*Size*],表示缓冲池容量的上下限,*QueueSize* 指定等待队列的长度,不能及时得到处理的请求将会被放入任务队列中等待处理,*RefleshCycle* 表示对缓冲池的刷新频率,刷新过程将会释放暂时没有被使用的资源.Level 0 级对这些资源没有限制;而 Level 1 级和 Level 2 级都对这些资源设定了不同的约束.

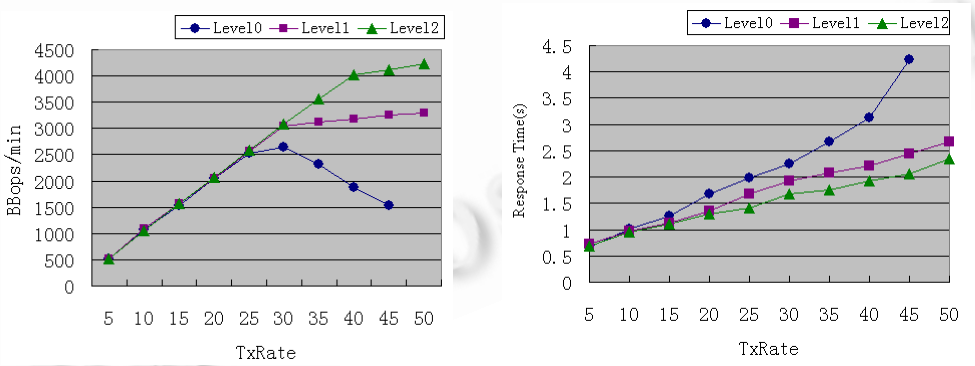
**Table 1** QoS parameters of OnceAS/Q in ECperf experiments

**表 1** OnceAS/Q 在 ECperf 测试中的主要 QoS 参数

| QoS level | Metrics | | | |
|---|---|---|---|---|
| | Server thread pool | DB connection pool | EJB object pool | EJB data cache |
| Level 0 | *Size*=Unlimited; *Queue*=Unlimited | *Size*=Unlimited; *RefleshCycle*=never | *Size*=Unlimited; *RefleshCycle*=never | *Size*=Unlimited; *RefleshCycle*=never |
| Level 1 | *Size*=[20,50]; *QueueSize*=20 | *Size*=[10,30]; *RefleshCycle*=300s | *Size*=[100,250]; *RefleshCycle*=60s | *Size*= [100,250]; *RefleshCycle*=60s |
| Level 2 | *Size*=[40,100]; *QueueSize*=50 | *Size*=[20,50]; *RefleshCycle*=900s | *Size*= [200,1000] *RefleshCycle*=300s | *Size*= [200,1000] *RefleshCycle*=300s |

图 8 显示了在不同的 QoS 约束下,Ecperf 应用的业务吞吐量(Bbops/min)和响应时间随 *TxRate* 值而变化的情况,图 8 (a)反映的是吞吐量,图 8 (b)反映的是响应时间.

从图 8 中可以明显地看出,在没有 QoS 管理的 Level 0 级测试中,当 *TxRate* 为 25 时,服务器的服务能力接近峰值,此时它对资源的需求还能得到满足;但之后随着负载的进一步加大,服务器对资源不能进行有效的管理,从而导致服务能力出现一定程度的下降,此时关键业务的响应时间也相应地增加.当 *TxRate* 增加到 50 时,服务器因为服务能力无法满足如此规模的应用需求而崩溃,从而不能完成正常的测试.在 Level 1 级的资源约束下,OnceAS/Q 的服务能力在 *TxRate* 为 30 的时候接近峰值,之后由于受到所分配资源的限制,它的服务能力开始在峰值附近稳定,这也造成大批请求由于不能得到及时处理而使业务的响应时间增加.在 Level 2 的资源约束下,由于系统的资源供给情况有了明显的提高,所以 OnceAS/Q 在 *TxRate* 为 40 时才接近服务能力的峰值,其后与 Level 1 类似,其受到的资源约束也使得服务器的服务能力增长非常缓慢.因此,OnceAS/Q 为具有不同 QoS 约束的应用所提供的服务是不同的,特别是在应用规模较大时表现的差异较明显.




(a) Throughout (a) 吞吐量 (b) Response time (b) 响应时间

Fig.8 Experimental results of ECPerf benchmark

图 8 ECPerf 基准测试结果

为了更清楚地描绘不同服务级别下 OnceAS/Q 对 ECPerf 应用请求的处理情况,我们给出了在 3 种不同 QoS 级别中,OnceAS/Q 在 Ecperf 测试峰值时的关键业务响应时间频率分布,如图 9 所示.图中横轴表示响应时间,纵轴表示在该响应时间内的关键业务数量,*TxRate* 值分别取 25,30 和 40.以曲线 Level 0-25 为例,它表示在 Level 0 级下,事务注入率为 25 时关键业务响应时间的分布频率.从具有同一事务注入率的 3 条不同曲线上,可以观察不同 QoS 级别的资源约束对业务响应时间的影响.

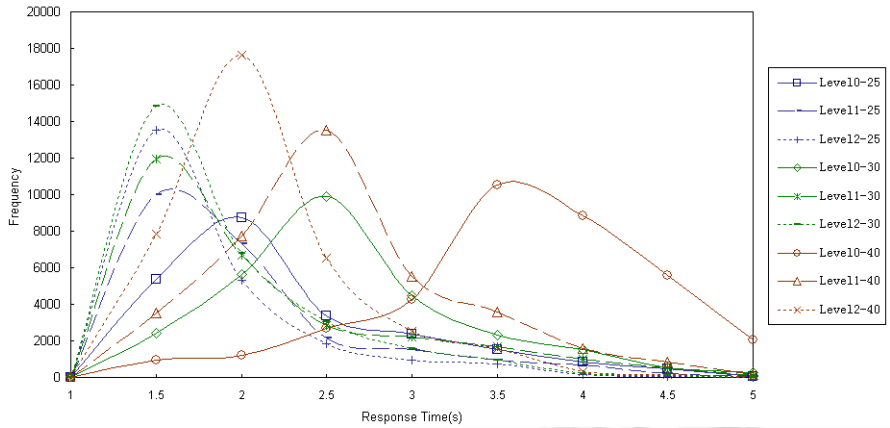



Fig.9 Frequency distribution of response times for key transactions

图 9 ECPerf 关键业务的响应时间频率分布图

从图 9 中可以观察到,在 *TxRate* 为 25 时,虽然 3 种级别的 OnceAS/Q 的吞吐量基本相同(如图 8(a)所示),但是,Level 0 级的 OnceAS/Q 服务能力已近峰值,而且响应时间介于 1.5s~2s 的业务最多;而在 Level 1 级和 Level 2 级的情况下,关键业务响应时间大多都分布在 1s~1.5s 之间,但后者比前者的分布频率更集中,这说明更高的 QoS 资源配给使得更多的关键业务能在较短的时间内完成,这在负载进一步加大后变得尤为明显.可见,虽然此时 OnceAS/Q 在不同的资源约束下所表现的吞吐量相差不大,但是遵循给定 QoS 规约的资源分配会改善 OnceAS/Q 内部的资源竞争,从而使系统可以提供更好的服务能力.在 *TxRate* 为 30 时,3 个级别的频率分布曲线与此类似,不过此时 Level 0 级的 OnceAS/Q 由于资源的过度争用,响应时间分布已经集中到 2s~2.5s 之间.当 *TxRate* 为 40 时,Level 0 级的 OnceAS/Q 服务能力已经开始明显下降,响应时间已经上升到 3s~3.5s 之间;Level 1 级的 OnceAS/Q 由于资源的配给也受到约束,响应时间上升到 2s~2.5s 之间;Level 2 级的 OnceAS/Q 在此时的服务能力接近峰值,其响应时间集中在 1.5s~2s 之间.

图 9 非常微观地描绘了不同的 QoS 约束对 OnceAS/Q 服务能力的影响.通过使用一定的 QoS 约束,可以避免资源争用过度而造成服务能力下降的情况.而不同的 QoS 约束对 OnceAS/Q 的服务能力具有不同的影响,OnceAS/Q 通过定义不同的 QoS 级别来满足不同的应用需求.

此外,我们对不同 QoS 资源约束条件下系统内存的使用情况进行了测量,如图 10 所示.

从图 10 中可以看到,在 *TxRate* 较低的时候,由于系统的负载较低,进行 QoS 资源约束比没有约束会消耗更多的内存.但是,随着负载的增加,由于无法对资源进行按需分配和管理,从而使无 QoS 保障的情况比有 QoS 保障的情况消耗更多的内存.而 Level 2 由于比 Level 1 享有更多的可支配资源,所以会消耗比 Level 1 更多的内存,但是二者的内存消耗在到达服务峰值后开始比较稳定的缓慢增长.

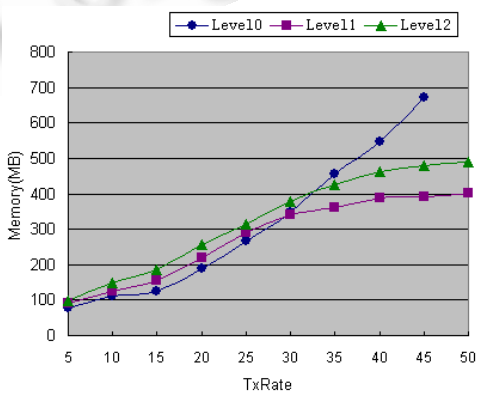



Fig.10 Usage of memory for different QoS levels

图 10 不同级别的内存使用

以上 ECPerf 实验结果表明,由于采取了有效的 QoS 控制和管理措施,OnceAS/Q 在给定 QoS 约束下,对资源的使用可以控制在一个可接受的范围内,会更好地提供满足应用需求的服务能力,在 Ecperf 应用中,这反映在业务吞吐量增大和响应时间较短等方面.在不同的 QoS 资源约束下,OnceAS/Q 也体现出与资源供给情况相符的服务能力,具有随需定制服务能力的特点,例如,不同服务级别下,Ecperf 应用的业务响应时间的频率分布件是不相同的.

## 5 相关工作

为中间件提供服务质量支持是中间件领域的一个重要发展趋势和研究热点[10,11],在此方面已经有不少研究工作,主要集中在多媒体、可靠性、实时系统等应用领域.

诸多具有 QoS 能力的中间件都提出了各自的 QoS 管理体系结构,以在 CORBA 平台上的研究为主,其中较具代表性的有:QuO[12]为有 QoS 需求的应用提供了一种通用的中间件支撑框架,它采用了一种与 AOP(aspect-oriented programming)编程范型类似的技术,在中间件的公共服务层提供了一种高级 QoS 抽象;TAO[13]支持实时 CORBA 规范,它的目标主要是为实时应用的 QoS 需求提供支持;DynamicTAO[14]、OpenORB[15,16]通过使用反射技术提供了对系统服务和资源进行动态重配的能力;CQoS[17]主要通过 Interceptor 机制提供了一个支持组合多种 QoS 性质的可定制框架.

Java 平台上的 QoS 中间件研究相对较少,主要借鉴了 CORBA 领域的研究成果.例如,2KQ+[18]框架提供统一的 QoS 管理框架,主要包括 QoS 编程环境、Q-Compiler 编译器和可重配置的中间件 2KQ,为开发具有 QoS 需求的应用提供了系统化的方法,以及在运行时提供 QoS 保障;Miguel 等人提出了 QoSBean 组件模型[19],通过扩展 EJB 组件模型来集成所需的实时 QoS 服务,但是缺乏对资源和服务重配的考虑; RAJE[20]对标准的 JVM 进行了扩展,在中间件的层次上提供监控资源存取和管理资源的手段.

目前大多数 WAS(如 WebLogic,WebSphere,Oracle9iAS 等)都隐藏接口之外的实现细节,在运行期间可定制和可重配置能力低,使得 QoS 的提供和保障能力较差,WAS 在运行期间较难根据运行环境和用户需求的变化进行适应性的变化.以 IBM 的“按需计算(on-demand computing)”[21]、JBossAOP[22]为代表,目前的 WAS 也在朝 QoS 保障的方向努力,但是技术尚未达到成熟的地步.

与相关研究工作相比,我们针对 J2EE 应用服务器的 QoS 提供与保障方面进行了较为全面的研究.首先提出了补充 J2EE 相关规范的 QoS 规约定义,给出了基于 QoS 规约,从应用层到资源层的 QoS 配置映射;建立了一个适应于 WAS 的 QoS 管理框架,并提供了服务组件和资源两个层次的 QoS 自适应机制,包括应用 QoS 环境(AQE)、抽象资源管理框架(QARMF)以及资源动态重配算法等.

## 6 结束语

在网络分布计算环境下,复杂的大规模应用系统对 QoS 具有较高的要求.作为应用的关键性基础设施,Web 应用服务器对于实现端到端的 QoS 保障有着重要的影响.本文对面向 QoS 的应用服务器的关键技术进行了研究.OnceAS/Q 提供了一个 WAS 的 QoS 管理和控制框架,本文阐述了该框架的结构和主要组件,对与 QoS 相关的两个主要问题进行了研究,一是 QoS 规约及其映射,另一是服务/资源动态适应机制.使用 J2EE 测试基准 ECPerf,对 OnceAS/Q 与 OnceAS 进行了对比测试.测试结果表明,OnceAS/Q 能够根据应用需求和运行环境,提供适宜的服务能力,而且 QoS 管理所带来的开销是处于可接受范围之内的.

下一步研究工作主要包括:如何在现有 QARMF 的工作基础上,加强资源的管理和自适应能力;WAS 的各种 QoS 性质(如可靠性、安全性、性能等)之间如何协调和权衡,以使 WAS 的 QoS 提供与保障更为准确和高效.

**References**:

[1] Mohan C. Tutorial: Application servers and associated technologies. In: SIGMOD, ed. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2002. 636.

[2] 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

[3] Sun Microsystems, Inc. Java2 platform enterprise edition specification (v1.4). 2003.

[4] Fan GC. Research on some key technologies for web application server [Ph.D. Thesis]. Beijing: Institute of Software, The Chinese Academy of Sciences, 2004 (in Chinese with English abstract).

[5] Schmidt DC, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects. Vol 2, New York: Wiley & Sons, 2000.

[6] Fan GC, Lin SB, Dong WC, Feng YL. WebFrame: An extensible multi-layer web application server. Chinese Journal of Computers, 2004,27(4):451~460 (in Chinese with English abstract).

[7] Sun Microsystems, Inc. Java dynamic proxy classes. URL: http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html

[8] Russell J, Norvig P. Artificial Intelligence: A Modern Approach. Englewood Cliffs: Prentice Hall, 1995.

[9] Sun Microsystems, Inc. ECperf specification (v1.0). 2001.

[10] Duran-Limon HA, Blair GS, Coulson G. Adaptive resource management in middleware: A survey. IEEE Distributed Systems Online, 2004,5(7).

[11] Schantz RE, Schmidt DC. Research advances in middleware for distributed systems. In: Chapin L, ed. Proc. of the IFIP 17th World Computer Congress. Deventer: Kluwer, 2002. 1~36.

[12] Vanegas R, Zinky J, Loyall J, Loyall J, Karr D, Schantz R, Bakken D. QuO's runtime support for quality of service in distributed objects. In: Davies N, Raymond K, Seitz J, eds. Proc. of the IFIP Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98). New York: Springer-Verlag, 1998. 207~223.

[13] Schmidt DC, Levine DL, Mungee S. The design of the TAO real-time object request broker. Computer Communication, 1998, 21(4):294~324.

[14] Kon F, Roman M, Liu P, Mao J, Yamane T, Magalhães LC, Campbell RH. Monitoring, security, and dynamic configuration with the dynamic TAO reflective ORB. In: Sventek JS, Coulson G, eds. Proc. of IFIP Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000). New York: Springer-Verlag, 2000. 121~143.

[15] Blair GS, Coulson G, Andersen A, Blair L, Clarke M, Costa F, Duran-Limon H, Fitzpatrick T, Johnston L, Moreira R, Parlavantzas N, Saikoski K. The design and implementation of open ORB version 2. IEEE Distributed Systems Online Journal, 2001,2(6). URL: http://csdl.computer.org/comp/mags/ds/2001/06/o6001abs.htm

[16] Coulson G, Blair GS, Clarke M, Parlavantzas N. The design of a highly configurable and reconfigurable middleware platform. ACM/Springer Distributed Computing Journal, 2002,15(2):109~126.

[17] He J, Hiltunen MA, Rajagopalan M, Schlichting RD. Providing QoS customization in distributed object systems. In: Guerraoui R, ed. Middleware 2001. New York: Springer-Verlag, 2001. 351~372.

[18] Wichadakul D, Nahrstedt K, Gu XH, Xu DY. $2K^{Q+}$: An integrated approach of QoS compilation and reconfigurable, component-based run-time middleware for the unified QoS management framework. In: Guerraoui R, ed. Middleware 2001. New York: Springer- Verlag, 2001. 373~394.

[19] Miguel MA. Integration of QoS facilities into component container architectures. In: Paul R, ed. Proc. of the 5th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing. Washington: IEEE Computer Society. 2002. 394~401.

[20] Sommer NL, Guidec F. A contract-based approach of resource-constrained software deployment. In: Bishop JM, ed. Proc. of the IFIP/ACM Working Conf. on Component Deployment. New York: Springer-Verlag, 2002. 15~30.

[21] Herness EN, High RH, McGee JR. WebSphere application server: A foundation for on demand computing. IBM Systems Journal, 2004,43(2):213~237.

[22] http://www.jboss.org

**附中文参考文献**:

[2] 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

[4] 范国闯.Web 应用服务器关键技术研究[博士学位论文].北京:中国科学院软件研究所,2004.

[6] 范国闯,林世彪,董伟川,冯玉琳.WebFrame:一种多层次可扩展的 Web 应用服务器.计算机学报,2004,27(4):451~460.