Web 应用服务器研究综述*

范国闯+, 钟 华, 黄 涛, 冯玉琳

(中国科学院 软件研究所 软件工程技术中心,北京 100080)

A Survey on Web Application Servers

FAN Guo-Chuang+, ZHONG Hua, HUANG Tao, FENG Yu-Lin

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

+ Corresponding author: Phn: 86-10-62630989 ext 205, Fax: 86-10-62562538, E-mail: fanguo@otcaix.iscas.ac.cn

http://otcaix.iscas.ac.cn

Received 2003-03-03; Accepted 2003-06-10

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

Abstract: Web application server (WAS), which is considered to be one of the most exciting milestones of enterprise software technology since the relational database and has become very popular in the last few years. It solves the problems in applying traditional middlewares to web computing environment and provides web middleware platforms that support for the deployment, integration, and execution of transactional web applications. Attention to the new category of middleware has been drawn from the academia and software industry. The current state-of-the-art of Web application servers, including what is a web application server and its functionality, architectural model, component container, transaction processing, load balancing, mid-tier caching, web services and analysis criteria etc., are outlined. Using the Ecperf benchmark, a comparative study among some leading Web application servers is presented by evaluating their functions and performance. Lastly, some open issues about WASs are discussed, and the future direction of WASs is pointed out.

Key words: Web application server; middleware; transactional Web application; Web computing

摘 要: Web 应用服务器是 Web 计算环境下产生的新型中间件,为创建、部署、运行、集成和管理事务性 Web 应用提供一个跨平台的运行环境,被认为是自关系型数据库以来最令人激动的企业应用技术.诸多 IT 企业纷纷推出其各自的 Web 应用服务器产品和系统,学术界也对这种热门领域产生了浓厚的兴趣.在分析 Web 计算环境下传统中间件发展所遇到的问题的基础上,介绍 Web 应用服务器的起源和发展、运行模式以及评测基准,然后对 Web 应用服务器研究现状进行综述,主要包括 Web 应用服务器的定义、体系结构、组件容器、分布事务处理、负载平衡、高速缓存、Web Service 等研究热点和关键技术.根据评测基准,对若干主流 Web 应用服务器从

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2001AA113010, 2001AA414020, 2001AA414310 (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计划(973))

第一作者简介: 范国闯(1974－),男,湖南娄底人,博士生,助理研究员,主要研究领域为网络分布计算,软件工程技术.

功能和性能两个方面进行分析和比较.此外,还指出了 Web 应用服务器目前存在的不足以及未来的发展趋势.

关键词: Web 应用服务器;中间件;事务性 Web 应用;Web 计算

中图法分类号: TP393 **文献标识码**: A

1 Web 应用服务器的起源与发展

中间件位于应用与操作系统之间,屏蔽底层操作系统、网络以及数据库的异构性和复杂性,简化网络分布应用的开发、管理和维护,从 20 世纪 80 年代末到现在,取得了长足的发展,得到了工业界和学术界的青睐,已成为分布式系统的主流技术之一.传统中间件[1]大致可以分为远程过程调用中间件(RPC)、面向消息的中间件(MOM)、对象请求代理(ORB)、事务处理监控器(TPM)等.但随着因特网的飞速发展,企业应用环境逐渐向 Internet/Intranet/Extranet 环境转移,多层计算模式逐步取代两层客户/服务器计算模式,应用的需求和规模不断扩大,传统中间件所面临的网络分布计算环境发生了很大的变化[2],与中间件开始产生时的典型企业计算环境截然不同,而是面向 Internet 的 Web 计算.

在新的计算环境下,传统中间件遇到了很多挑战,如需支持多种客户类型、Web 服务器本身能力的不足、用户数量不可估计、大规模的并发用户访问、分布式长事务、与已有系统的无缝集成以及缺乏高可靠性、高伸缩性和高可用性等 QoS 保证.解决这些问题已成为 Web 计算和中间件领域的一些热点和方向[2,3].Web 应用大多数采用三层客户/服务器结构,如何实现位于客户和数据库之间的中间层是解决问题的关键.虽然事务处理监控、消息服务器等传统中间件能够实现中间层[4],但均受限于它们最初产生时的计算环境,不能封装、重用业务逻辑[5],只能为事务性 Web 应用提供部分功能.例如,消息服务器主要解决消息异步可靠传送问题,事务处理监控器主要用于大规模事务处理.1998 年,人们提出了一种实现中间层的新方式——Web 应用服务器.相对于传统中间件,可以认为 Web 应用服务器是在 Web 计算环境下产生的新型中间件.

Web 应用服务器自产生以来,得到了工业界和学术界的普遍关注,已成为近几年最热门的软件类型[6,7],E.D. Zander 认为,Web 应用服务器是自从关系型数据库以来最令人激动的企业应用技术[8].Web 应用服务器的应用前景非常看好,研究人员预期应用服务器软件平台市场将以 50%的年增长率发展[9].国际上许多著名的计算机公司和开放源码组织纷纷推出了各自的产品和系统,如 IBM Websphere, BEA Weblogic, Oracle9iAS, SUN iPlanet, 开放源码 JBOSS, ENHYDRA 等.国内如中国科学院软件研究所、国防科学技术大学、北京大学、西安交通大学等,也在 Web 应用服务器方面做了大量的研发工作.相对于 IT 企业完整的 Web 应用服务器产品和解决方案,学术界在 Web 应用服务器的研究方面虽然稍显落后,但部分研究人员对这种热门软件产生了浓厚兴趣,并开始重视 Web 应用服务器的研究.Mattern 在文献[3]中指出,应用服务器是分布式中间件的发展趋势;C. Mohan 在文献[10]中对应用服务器如何与后端系统进行交互等问题也进行了讨论;此外,在文献[10~16]中对 Web 应用服务器中的 Caching 技术进行了研究.

2 Web 应用服务器的运行模式

近年来,Web 应用服务器取得了长足进展,目前市场上国内外的产品超过 60 家.从技术实现和运行模式角度划分,这些 Web 应用服务器可以分为脚本模式、面向对象模式和对象模式 3 类[17].

脚本模式应用服务器通过脚本以超文本方式描述动态页面的内容和处理逻辑,当接受到客户请求时,应用服务器首先搜索相应的源文件,然后在服务器端解释该源文件中的脚本,最后将脚本解释器产生的结果汇编后返回给 Web 服务器.脚本模式 Web 应用服务器一般都通过扩展接口(如 NSAPI, ISAPI, CGI 等)或服务器端脚本语言(如 PHP, Livewire 等)动态地产生响应页面.但是,这种模式的应用服务器可扩展性、高可用性差,可重用性低.此外,该模式缺乏集成历史遗留系统以及事务处理的支持能力.

面向对象模式介于脚本模式和对象模式之间,其主要特点是使用面向对象语言编写脚本.例如,在 JSP 和 Servlets 中使用 Java 语言进行编码.相对于纯脚本模式,该模式的可重用性较好,但没有相应的规范,不提供统一

的接口规范,其使用范围和移植性受到了限制.

对象模式应用服务器支持分布对象模型,能将应用划分为多层,易于维护,在开发和部署过程中支持组件重用,模块化程度高,业务逻辑的变化只需修改相关的组件即可.与面向对象模式相比,对象模式应用服务器遵循相应的标准和规范,其中较突出者主要有两大类:J2EE(Java 2 platform enterprise edition)类和微软.Net 类.J2EE 由 SUN 公司在 3 年前提出,目前至少有 40 多种实现 J2EE 规范的应用服务器.J2EE 为事务性 Web 应用的开发、部署、运行和管理提供一系列的规范和标准,主要包括 Java Servlets,JSP,EJB,JTA,JTS,JMS,JAXP,JMX,RMI-IIOP,JNDI,JCA,Java Mail 和 JAF 规范.这些 J2EE 规范为应用服务器的实现提供了一个完整的底层框架和一套标准的规范,在不同的 J2EE 应用服务器之上的应用组件也可以互操作,移植的风险和代价小.而微软则在其操作系统上附加一系列具备中间件功能的软件包来提供应用服务器的相应的功能.微软.Net 构建在 Windows DNA 技术(如 Microsoft Transaction Server, COM+, MSMQ, SQL Server 数据库等)基础上,在.NET 中提供了一系列企业级应用服务,为部署、管理和建立基于 XML 和 Web 的应用构筑了.NET 服务器结构,包括 Application Center, BizTalk Server, Commerce Server, Exchange Server 等,它们结合 Windows 平台上的一系列开发工具和技术(包括 Visual Studio.NET, ASP.NET 等),提供了强有力的应用服务器解决方案.虽然目前 J2EE 和.NET 势均力敌,但是 J2EE 作为一种规范,具有.Net 无法比拟的跨平台、企业应用集成能力以及可扩展性和开放性,得到了许多厂商的支持,已逐步被广大研发人员和企业所接受[18],有良好的前景,逐渐成为 Web 应用服务器研究和开发的一个方向.

3 Web 应用服务器的评测基准

鉴于微软.Net 服务器由微软公司自己开发,目前主要对不同厂商实现的 J2EE 应用服务器进行评测.对 J2EE 应用服务器的评测基准主要有两个:

(1) J2EE 兼容性测试 CTS(J2EE compatibility test suite)[19].CTS 是 SUN 公司提供的一个测试套件,包括 15 000 多个测试,其中包含了 J2EE 规范所要求的所有类和方法的测试,用以检查 J2EE 应用是否可以正确、一致地交互.它可以保障应用的可移植性,使 EJB 组件、JSP 和 Servlets 可以运行在不同的 J2EE 应用平台上,保障不同厂商之间的一致性.目前,主流 J2EE 应用服务器如 BEA Weblogic, IBM Websphere, Oracle9iAS 均已通过 J2EE 1.3 兼容性测试.

(2) ECperf.它是一种评测 J2EE 应用服务器可伸缩性和性能的测试基准(benchmark)[20],通过模仿现实世界中系统所要承受的工作量,具体包括制造、供应链管理和订单/库存等过程,重点测试 EJB 容器处理内存管理、连接池、去活/激活、缓存等方面的能力.目前,ECperf 规范有 1.0 和 1.1 两个版本,由测试应用程序和相关的驱动程序以及一套详细的测试规范和结果报告规范组成.ECperf 的主要度量值是性能度量值(BBops/min)和性能价格度量值($/Bbops).BBops/min 是每分钟基准业务操作数(BBops/min),表示在测量时间段中每分钟完成的成功基准业务操作的平均数量.性能价格度量值($/Bbops)是一个系统性能价格比,其值为受测系统(system under test,简称 SUT)的总价格除以性能度量值(BBops/min)所得到的值.ECperf 还可以通过业务事务(business transaction)响应时间的频率分布图和工作订单吞吐量图来评测 J2EE 的性能.

ECperf 由 JCP(Java community process)开发.2001 年,SUN 公司将它授权给 SPEC(standard performance evaluation corporation).SPEC 对 ECperf1.1 规范重新包装后,发布了 SPEC jAppServer2001.随后在 2002 年又发布了 SPEC jAppServer2002.SPEC jAppServer2001 与 SPEC jAppServer2002 基本相似[21],其区别在于所使用的 EJB 规范不同,前者为 EJB1.1,后者为 EJB2.0.jAppServer2002 取代 Ecperf 以后,逐步成为评测 J2EE 应用服务器性能的工业基准.

4 研究现状及目前存在的不足

Web 应用服务器覆盖了在 Web 计算环境下计算机软件技术的诸多领域,如软件工程、分布计算、面向对象、事务处理、数据管理、应用集成、网络安全等[7,10],其研究和发展均建立在这些技术领域发展的基础上.

限于篇幅,下面仅就 Web 应用服务器目前的研究热点和关键技术进行分析.

4.1 Web应用服务器的定义及功能

近年来,许多研究人员、组织机构对 Web 应用服务器分别提出了不同的定义,但到目前为止,Web 应用服务器仍然没有一个完全统一的定义.下面是两类有代表性的定义:

(1) C.Mohan, Dietzen 和 Stonebraker 的定义.C.Mohan 在文献[7,10]中认为,应用服务器是传统事务处理监控器(transaction processing monitor)在 Web 环境下的现代等同物,是事务性服务在线应用的执行平台,将现代的应用环境与历史遗留数据源集成是应用服务器最重要的特征之一.Dietzen 在文献[22]中认为,应用服务器是下一代事务处理监控器.Stonebraker 在文献[23]中认为,应用服务器是传统事务处理监控器的重命名,它主要提供应用的激活和其他相关的服务,如负载平衡、通过多线程和连接池技术提高处理效率等.

(2) Copeland, Marshall, Ricciuti, Dolgicer 和 Ritter 的定义.Copeland, Marshall, Ricciuti 和 Dolgicer 等人在文献[5,6,8,24,25]中认为,应用服务器是位于以浏览器为基础的前端和诸如历史遗留应用的后端之间的,基于组件的服务端软件,它高速存取后端系统的信息,处理业务逻辑,集成企业计算中的资源和应用,为安全、状态维护、数据访问、数据存取提供中间件服务.

第(1)种定义认为,应用服务器是面向 Web 应用的事务监控中间件,但应用服务器并不是传统事务监控器在 Web 上的简单延伸,它与传统的事务处理监控器的区别在于:(1) 支持更多的大规模用户;(2) 支持更标准、更开放的协议和 API;(3) 支持 Web 用户界面以及一系列 WWW 标准;(4) 侧重于有状态服务;(5) 支持 Web Services;(6) 传统事务处理监控器不具有执行和管理应用逻辑、定位和实例对象以及控制对象生命周期的能力[5].第(2)种定义则强调应用服务器是位于中间层的服务端软件以及对应用逻辑的处理能力.

综合上述两种定义,我们定义应用服务器是为创建、部署、运行、集成和管理事务性 Web 应用提供一系列运行时服务(如消息、事务、安全、应用集成等)的可伸缩、高可用、高可靠、高效的中间件平台.应用服务器需满足 Web 计算的特定需求,所以,应用服务器通常同时具有 Web 服务器的功能,亦称为 Web 应用服务器.根据这个定义,应用服务器不仅具有传统事务监控器的高可伸缩性、高可用性、高可靠性、高效等高级特性,而且为事务性 Web 应用的创建、部署、运行、集成和维护提供所需的通用服务,更反映出应用服务器是在面向 Internet 的 Web 计算环境下产生的这一特点.

Web 应用服务器的主要功能有哪些?Mohan 和 Kassam 在文献[7,10,26]中对此进行了研究,但并不全面.根据前文对 Web 应用服务器定义,我们认为它应具有如下的功能:(1) 提供运行事务性 Web 应用的跨平台运行环境:支持分布计算模型(如 COBRA,EJB,COM 等)的组件容器;支持大规模 Internet 用户、多种设备的 Web 容器;提供进程管理、通信管理、线程池管理、连接池管理、内存管理、会话管理、状态管理、对象生命周期管理和对象池管理;(2) 提供一系列基本服务,包括事务服务、消息服务、安全服务、名字服务、邮件服务;数据访问服务,如关系型数据库、XML、文本数据等;企业应用集成服务:支持与其他外部资源的快速集成,如历史遗留系统资源、ERP 系统、关系型数据库或其他数据资源、COBRA 和 COM/ActiveX 组件;QoS 服务如集群、路由、失效恢复、状态复制、会话持久化、会话迁移、负载平衡、对象、数据高速缓存等;(3) 提供一系列工具,包括事务性 Web 应用开发工具;快速、简单方便的应用部署工具;可扩展、可集成的管理工具;支持高速缓存、负载平衡、失效恢复的第三方 Web 服务器插件.

4.2 Web应用服务器体系结构

如上所述,Web 应用服务器是为事务性 Web 应用提供一系列服务的中间件,屏蔽底层具体实现细节和运行环境的差异.绝大多数应用均受益于此,但为了满足 Web 应用和计算环境(如无线网络等)对系统不断变化的 QoS 需求,需开放底层实现,让应用动态地感知底层内部状态,甚至替换底层实现,提供一个开放、灵活的体系结构[27~31].许多传统中间件在开放性方面做了一些工作.如 OMG 为 CORBA 的一些支撑服务(如事务和安全性服务)提供了内部接口;而 Portable Object Adapter 也使 CORBA 应用系统的设计变得较为开放;RM-ODP (ISO/ITU-T reference model for open distributed processing)体系结构则通过区分对待计算和设计过程提高开放性.然而,或是由于最初设计思想的束缚,或是因为既成系统的调整存在很大的困难,这些传统的中间件只是解

决一些特定、局部的问题,不能满足体系结构开放性的要求.为此,许多研究人员通过应用反射机制实现运行期间可定制(customizable)、可重配置(reconfigurable)的体系结构提高中间件的开放性和灵活性[29~33],其中具有代表性的工作主要有 DynamicTao,OpenORB,FlexiNet,OpenCOBRA,QuarterWare 等.DynamicTao 扩展 TAO ORB,是一种反射的 ORB,使用一系列的配置器(configurator)管理组件之间的相互依赖关系,提供在运行期间监测、更改 ORB 配置状态和动态装载、卸载组件的元接口(meta-interface),使得用户可以飞速地(on-the-fly)重配置 ORB 及其应用.OpenORB 采用一种语言无关的反射体系结构,每个对象或接口均有一个元空间(meta-space),每个元空间被划分为 Interfaces,Architecture,Interception 和 Resource 多个元模型(meta-model),使用对象图描述复合组件之间的关系,支持结构性、行为性反射.此外,OpenORB 利用组件框架(component framework)支持元空间构造,用户可以在不同层次动态插入、替换、定制和扩展的组件框架.OpenCORBA 基于元类(meta-classes)的概念,通过替换 COBRA 服务的元类变更服务行为的方式提供反射功能.QuarterWare 通过一个反射接口将定制组件插入到组件框架中增强系统的开放性.上述大多数中间件使用面向对象的方法提供反射功能,从而提高系统的开放性,同时,也有一些研究人员使用 AOP(aspect-oriented programming)构建中间件元级体系结构(meta-level architecture).

目前,Bea Weblogic 和 IBM Websphere 等大多数主流 Web 应用服务器,都基于一体化体系结构(monolithic architecture)和黑盒抽象机制(black box philosophy)进行设计[5,9,10,29],存在开放性和灵活性低等不足.尽管传统中间件已在开放的体系结构方面做了许多改进工作,但均没有将系统服务从内核分离,其可定制、可重配等开放能力则有限,更不具备在中间件整个生命周期内提供多阶段可重配置的能力[32].如何应用反射机制,结合上述已开展的工作,研究 Web 应用服务器多阶可重配体系结构,将成为热点之一.

4.3 组件容器

Web 应用服务器一般都支持分布组件模型如 EJB,COM/DCOM 等.这些分布计算模型采用组件-容器-组件服务器这样的三级计算结构,该计算结构将组件的功能性方面(functional aspects)和非功能性方面(non-functional aspects)分离.目前对组件容器的研究主要在客户端透明性、容器的可重配、组件打包和部署技术这 3 个方面.

(1) 客户端访问透明性.实现客户端访问透明性的一种通用方法是为组件生成客户端的代理.该代理拦截客户的所有请求,将请求透明地转发到组件所在的容器.但这种方法存在一个不足之处,即它将所有的请求视为远程操作,而当客户和组件在同一主机或同一进程空间时,这种方法会增加不必要的通信开销.这样,通过定位组件所在的位置,自适应地选择最有效的通信机制(如函数直接调用、共享内存、TCP/IP、Message Queuing 等)激活操作,从而有可能提高响应速度.这个选择过程称为“分配优化”(collocation optimization).然而,在不影响后端组件容器服务质量的情况下,如何透明地在运行期间实现该“分配优化”选择过程,自适应地选择高效的通信机制,将成为一个值得研究的问题.

(2) 组件容器的可重配置.由于组件容器所提供的非功能性服务的具体实现都被隐藏起来,配置这些服务受到了严格的限制,不具备在运行期间可重配置的能力,因此很难满足事务性 Web 应用不断变化的 QoS 需求.文献[33]将容器构建为带元接口(meta-interface)组件框架,通过元接口开放容器,组件框架负责处理容器的非功能性方面的服务,但该方法基于 COM 组件模型,受限于其运行的 Windows 环境,此外也没有详细地给出具体实现细节.文献[34]基于应用指定的配置策略,在运行期间响应每个激活请求时动态选择不同“组件类型”(component type)的实现满足 ORB 非功能性需求,但该方法可能会增大系统额外的开销,降低系统的处理能力;此外没有给出如何解决在运行期间动态替换“组件实例”(component instance)时组件状态一致性和组件类型匹配问题.文献[35]应用 AOP(aspect-oriented programming)编程技术,视组件容器为元对象(meta-object),使用可反射的“对象组合模型”(object composition model)分离、组合以及关联组件不同的非功能性服务,但该方法侧重于底层通信服务,实现较复杂,很难向其他非功能性服务属性扩展.上述工作大多数是对原有工作进行改进,只能实现组件容器部分非功能性服务的可重配,很难从根本上实现容器自身的可重配置和可定制.反射是一种实现中间件动态可重配置的有效方法之一.文献[36]虽然侧重于容器的入侵检测服务,但描述了设计可反射组件容

器时的若干初步想法,为设计一种在运行期间可检测(inspect)其自身的状态,并能根据指定的 QoS 需求自适应地调整(adaptation)其自身功能的可重配组件容器奠定了基础.

(3) 组件打包与部署技术.为了满足应用类型不断增加的需求,支持应用和组件的热部署,如何建立一种灵活、可扩展的热部署机制是 Web 应用服务器值得研究的一个问题.目前,主流 Web 应用服务器均提供部署工具,支持应用的热部署,但都只提供固定应用类型的部署,并已固化在应用服务器中,不具有扩展性,不能动态地支持不断增多的应用类型.文献[37]提出了很多提高灵活性和扩展性的通用设计模式(design pattern),如 Factory 等,应用这些设计模式将有助于建立一种灵活、可扩展的热部署机制.

4.4 Web应用服务器中的分布事务处理

分布事务处理是 Web 应用服务器重要特性之一,用来保证 Web 应用对分布资源访问的一致性和可靠性.目前 Web 应用服务器事务服务通常遵循国际/工业规范,如 J2EE JTS 或者 CORBA OTS 等.Web 应用服务器为电子商务、工作流、企业信息系统等不同类型的应用提供事务服务,相对于传统数据库事务,其事务更具有分布性和多样性的特点.此外,运行在 Web 应用服务器中的组件可能参与长时间分布式事务.然而,目前大多数事务服务都基于平面事务模型,采用两阶段提交协议,存在许多不足之处.比如在执行长事务时,平面事务模型降低系统的并发度和性能,增大事务间冲突概率,导致事务等待时间延长,并且会明显增加事务死锁.此外,平面事务的恢复处理不灵活.近十几年来,为了解决平面事务模型的不足,研究人员已在平面事务的基础上加入特定的应用语义,基于固定的控制和数据依赖,在不同方面放宽对 ACID 特性的要求,提出各种扩展事务模型,包括嵌套事务模型、多层事务模型、Sagas、分支汇合事务模型、柔性事务模型等.但由于不同应用领域需要不同的扩展事务模型,并且传统事务处理监控器采用封闭、一体化的结构[38],因此这些扩展事务模型大多数停留在理论研究上,实际可用的具体实现则很少.为此,研究人员提出一种可以支持多种扩展事务模型的框架,如 ACTA, ASSET,ECA,RTF 等,但这些框架只能在预先设定的若干扩展事务模型中静态地选择.为此,结合计算反射和开放实现技术,文献[39~41]设计了一种反射事务框架和反射事务服务,但实现一种能够根据环境变化自适应地调整不同的扩展事务模型的事务服务非常困难,也将成为 Web 应用服务器中的一个研究热点和难点.

在事务执行方面,目前 Web 应用服务器所支持的规范,如 JTS 或 OTS,均基于一对一的请求/回答方式进行同步交互.而 Web 应用运行在面向 Internet 的 Web 计算环境中,事务的执行可能是松耦合的,即在事务执行过程中,可能利用各种类型的消息,如同步消息、异步消息或异步通知(发布/订阅),发出事务请求,在事务提交之前,事务参与者响应该请求.文献[42]虽然指出了 EJB 规范缺乏对异步事务的支持,但没有给出解决办法.研究 Web 计算环境下的松散耦合型分布式事务,并在 Web 应用服务器中实现,也是 Web 应用服务器中的一个难点.

4.5 Web services

近年来,Web 服务(Web services)已成为一大热点,许多研究机构、组织和公司都纷纷投入到 Web Services 的研究和应用当中.Web 服务通过定义好的接口使用标准的 Internet 协议交互,并由一种标准的功能描述语言来描述,采用面向服务(service oriented architecture,简称 SOA)的体系结构,具有封装性好、耦合度低、互操作性好、可集成能力强等优良特性,被认为是一种推动分布计算向前迈进的革命性技术[43].从外部使用者的角度而言,Web Services 是一种面向 Internet 的应用服务.作为在 Web 计算环境下产生的中间件平台,Web 应用服务器自然是 Web Services 理想的执行环境.Web 应用服务器通过 Web Services 容器为 Web Service 应用提供一系列的基础服务,响应 SOAP 请求,发布服务接口.此外,Web 应用服务器还通过 SOAP 包装器(wrapper)把历史遗留系统或部署在后端应用服务器中的组件(如 EJB,COM 等)组合为 Web Service.目前,微软.Net 和 SUN 公司的 J2EE 两大类型的 Web 应用服务器均支持 Web Services[44].微软提出了一种全局 XML Web Services 体系结构 GXA(global XML Web services architecture),该结构包括一系列规范如 WS-Inspection,WS-license,WS-Referral, WS-Routing, WS-Security 等.为适应该体系结构,微软提供了一个创建、部署和运行 XML Web Services 应用平台,称为.Net 框架(.Net framework).此外,微软还提供了一个以用户为中心的 Web Service 体系结构和一系列 XML Web Services,统称为.Net My Services.SUN 公司的 J2EE 真正支持 Web Services 却相对较晚,在 J2EE 1.4 规范中将 Web Service 作为一种标准服务定义,并扩展了 EJB2.0 规范,提供 JAX-RPC,JAXB,JAXP,JAXR 等一系

列 Java API.许多 J2EE 应用服务器厂商对 Web Service 的支持非常活跃.如 IBM 开发了 Web Services 激活框架 WSIF(Web services invocation framework);Bea 发布了 Weblogic Workshop,使得开发人员非常轻松地使用可视化控件开发 Web Services(文献[44]详细地介绍了其他 J2EE 应用服务器对 Web service 的支持情况).

Web Services 是开发下一代分布式应用的重要技术之一.与处理其他应用类型一样,Web 应用服务器也要为 Web Services 应用提供可靠性、安全性、可伸缩性、高可用性、高效性等服务保证.而 Internet 的开放性和异构性使得大多数传统解决方案几乎不可重用,为此,Web Service 的可靠性、安全、事务、可伸缩性、高可用性等诸多问题需要进一步地加以研究[43,45].文献[46,47]分别对 Web Services 的可用性、互操作性进行了研究.IBM Almaden 的 CMohan 扩展了以前缓存的支持,在文献[12]中研究了 Web Services 的缓存技术.文献[48,49]将安全特性增加到 Web Service 中,2002 年 4 月,IBM、微软和 VeriSign 发布了 Web Service 安全规范.Web Services 的可靠性主要受到 HTTP 协议的影响,为此,设计了 HTTPR(reliable HTTP),BEEP(blocks extensible exchange protocol)和 DIME(direct Internet message encapsulation)协议[43],然而广泛地接受这些协议并得到各 Web 应用服务器厂商的支持还需要较长的时间.

目前,Web 应用服务器无法保障 Web Services 应用的事务特性.Web Services 的事务模型和对应的 Web 事务基础支撑还没有建立.不同于传统事务处理,不同的事务模型和中间件技术在同一 Web 事务中出现,事务上下文表示、服务协调和管理必须以松散耦合的方式进行,从而使得 Web 事务处理更复杂[50].目前主要通过扩展两阶段提交协议或使用一种“事务补偿”的方法解决 Web Services 的原子性和一致性问题[42].2001 年,OASIS 组织提出一种管理 B2B 事务的 BTP 协议(business transaction protocol).该协议通过两种事务扩展两阶段提交协议,一种是通常所说的原子性事务;另外一种是所谓的“内聚事务”,该事务有多个原子事务复合,支持跨多个企业的长事务.文献[51]也使用一种类似的方法将对象事务组合为 Web Service 事务.“补偿事务”方法则不同于两阶段提交协议,事务通常可以允许提交,但该事务执行的行为和结果还可以在提交之后取消.但“补偿事务”的方法并不完全满足 ACID 特性的要求.此外,传统的两阶段提交协议还必须重新设计和扩展,不能简单地应用到 Web Services 事务中,这也说明,在 Web 应用服务器中实现 Web Services 事务还有很多工作要做.

随着 Web Services 相关规范的逐步完善以及各 Web 应用服务器厂商 Web Services 开发工具和运行环境的推出,Web Services 作为一种新的 Web 应用模式正在迅速步入实用阶段,基于 Web Services 的复杂应用也会逐渐增多.如何为这些 Web Services 提供 QoS 服务基础设施,有效地管理这些 Web Services 应用,复合已有的 Web Services,构筑新的 Web Services,也成为 Web 应用服务器需要进一步解决的问题.

4.6 Web应用服务器中的负载平衡

负载平衡主要在算法和服务实现两个方面进行研究.主流 Web 应用服务器,如 BEA Weblogic 等仅支持一些常见的负载平衡算法,如 Round-Robin、Weight-based、随机等,不能感知后端服务器的状态,从而不能根据系统负载信息进行决策,也不能进行负载反馈,不具有自适应控制能力.文献[52]等在动态负载平衡算法进行了全面、系统而深入的研究,这些算法都可以应用到 Web 应用服务器,但这些算法大多基于进程和 TCP/IP 地址,是一种粗粒度的负载平衡,而 Web 应用服务器需要支持组件负载平衡模型,其负载共享和任务迁移不是在进程之间,而是部署在不同进程中组件之间.在组件之间进行负载平衡,负载检测和任务迁移的开销会增大.如何尽可能地降低额外的开销、增大负载收益,是研究 Web 应用服务器负载平衡算法时一个突出的问题.当进行任务迁移时,需要在运行期间进行组件迁移,如何维持组件(而不是进程)的状态和通信是研究 Web 应用服务器负载平衡的一个难题.

在负载平衡服务方面,主要包括网络级、操作系统级和基于中间件这 3 种负载平衡服务.基于中间件的负载平衡服务位于应用和操作系统之间,屏蔽底层通信复杂性和操作系统的差异,客户可以根据需要在运行期间灵活地定制负载度量和负载平衡策略,克服了网络级和操作系统级负载平衡服务的不足.基于中间件的负载平衡服务有非自适应性和自适应性两种.非自适应性负载平衡服务只能为简单分布式应用提供负载平衡支持.比如,在文献[53]中通过集成名字服务为无状态分布应用提供负载平衡服务,但这种负载分配是静态的、非自适应性的,不能根据系统变化调节自身行为进行负载分配.自适应负载平衡服务则能根据系统运行状态信息进行负

载分配的决策,对系统负载的变化具有自适应的调整能力.文献[54~56]讨论实现 COBRA 负载平衡服务所需解决的问题,并应用多种模式设计了一种自适应的负载平衡服务.但该服务依赖于 COBRA 规范的 Request-Forwarding 机制,不能直接应用到 J2EE 应用服务器,其负载粒度是对象级的.如何在 J2EE 应用服务器中实现多粒度的自适应负载平衡服务,对多种客户请求进行自适应调度,是一个值得研究的问题.

4.7 高速缓存技术

高速缓存技术是提高 Web 应用服务器吞吐量、加快请求响应速度的一种有效办法.从缓存的对象来看,主要有两个方面.(1) Web 页面缓存方面.早期主要研究静态页面以及经常访问的多媒体等大数据静态对象,随着近年来个性化要求的不断提高和动态页面的出现,如 ASP,JSP 等,人们开始对动态页面缓存进行研究.与静态页面缓存不同,动态页面缓存需要解决一些新问题,如 Disambiguation 问题、会话管理、个性化以及持久管理,其中 Disambiguation 问题比较复杂.文献[57~60]试图通过建立后端数据和缓存内容的精确映射关系来计算和监视这种依赖关系,触发缓存内容的刷新,然而其监视、触发刷新以及分析依赖关系的开销可能很大,Oracle Web cache[13]则采用一种粗粒度、保守的方法,根本不维护数据映射关系,而是利用一种灵活的文档标识符维护依赖关系,优化了处理性能.(2) 中间层数据缓存方面.缓存中间层数据对象降低 Web 应用服务器查询数据库的开销,减少后台数据库的竞争,加快请求响应速度.结果集缓存(result set caching)是较早使用的一种数据缓存方法,但存在许多不足,如缺乏再查询处理能力,其缓存内容不可共享和更新.为此,文献[11,12,61]对数据库中的基本数据缓存进行研究,以克服部分不足.Oracle 9i AS 则能缓存整个表,依赖复制工具异步传递从后端数据库的更新.IBM 的 DBCache 结合 DB2 的特性,缓存 DB2 表一级的数据,具有分布查询处理能力,客户应用透明性好.但这两种方法都不能在缓存中更新表.TimesTen 是一种轻量级的中间层数据管理器,占用系统资源小,通过扩展 SQL 语法可以缓存整个表或子集,在缓存中允许更新内容,能与后端数据保持一致.与 Oracle9i AS 和 IBM DBCache 不同,TimesTen 基于 In-Memory 数据库技术,减少了基于磁盘的数据库系统读写磁盘的负载,性能较佳.随着 Web Services 应用的出现,如何缓存 Web Service 动态执行结果将成为 Web 应用服务器高速缓存的新问题.

5 主流 Web 应用服务器的分析与比较

下面我们从功能、性能两个方面对 BEA Weblogic,IBM Websphere 和 Oracle 9iAS 这 3 种主流应用服务器进行简单的分析和比较.在功能方面,Weblogic,Websphere 和 Oracle 9iAS 可以说相差无几,都获得了 SUN 公司的 J2EE CTS 认证,提供 J2EE 规范中所规定的基本功能和负载平衡、失效恢复、缓存等扩展功能.在 QoS 服务、企业应用集成以及配套工具等方面,Oracle9iAS,BEA Weblogic 落后于 IBM Websphere.BEA Weblogic 在高可用性、高可靠性、稳定性以及与其他中间件集成等方面具有一定的优势.Oracle 依靠其在数据库领域的领导地位,首先提出 ESI(edge side includes)的概念,在高速数据缓存技术方面占有一定的优势,可伸缩性相对较好.

在性能方面,根据 ECperf 发布的最新数据[62],使用 Ecperf 基准的性能度量值(Bbops/min)、业务事务响应时间的频率分布图和吞吐量图进行分析和比较.从表 1 可以看出,Oracle9iAS 每分钟处理的基准业务操作数(BBops/min)最大,其值为 60 000 多,优于 IBM Websphere 和 BEA Weblogic,是目前最快的应用服务器.Oracle9iAS 通过 Web Cache,Object Cache 以及 Cache Cluster 提供了系统的整个性能[13].

Table 1 Top five results by performance
表 1 ECperf 按性能排名前五名

Sponsor	J2EE System	BBops/ min@Std	Price/ BBops	J2EE server	DBMS	Date approved
Oracle, Sun	Sun Fire 3800 cluster	61 862.77	$28	Oracle9iAS Release 2 v9.0.2.1.0	Oracle9i (v9.2.0.1.0)	07/08/2002
Oracle, Sun	Sun Fire 3800 cluster	51 007.80	$37	Oracle9iAS Release 2 v9.0.2.1.0	Oracle 9i Release 2	6/24/2002
IBM	IBM eServer x330 cluster	44 294.97	$23	Websphere AE 4.0.3	DB2 7.2	07/08/2002
HP	HP rp8400	37 791.00	$36	BEA Weblogic Server 7.0	Oracle 9i	07/08/2002
Oracle, Sun	Sun Fire V480 cluster	36 122.60	$12	Oracle9iAS Release 2 v9.0.2.1.0	Oracle 9i (v9.2.0.1.0)	07/08/2002

ECperf 提供了生成新订单(new orders)、改变订单(change order)、客户状态(customer status)查询、订单状态查询(order status)等业务事务响应时间频率分布图.限于篇幅,我们选择“生成新订单”这一有代表性业务事务的频率分布图进行分析和比较.

从图 1~图 3 可以看出,三大主流应用服务器处理业务事务的整体响应时间均在 2 秒内,其中 WebLogic 相对较优,其响应时间控制在 1 秒内,而 Oracle9iAs、Websphere 部分业务事务的响应时间超过了 1 秒.但 Oracle9iAS 处理单个业务事务的响应速度最快.图 4~图 6 描述了在 1 800 秒内该三大 Web 应用服务器工作订单吞吐量的情况.图 1~图 7 的数据来源为 ECperf 官方公布的数据,2002-8-31,http://www.theserverside.com/ecperf.

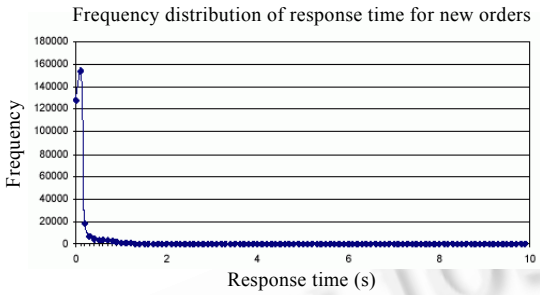

Fig.1 Weblogic’s frequency distribution
图 1 WebLogic 事务频率分布

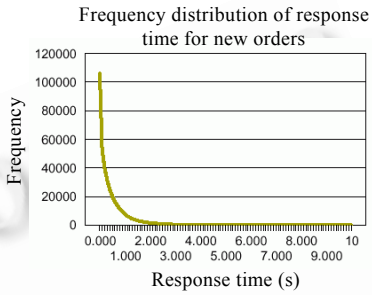

Fig.2 Websphere’s frequency distribution
图 2 Websphere 事务频率分布

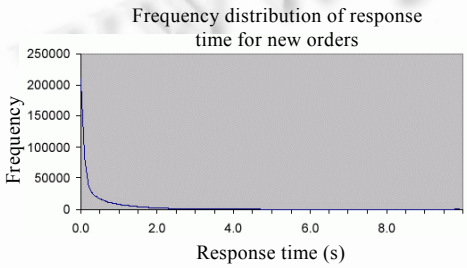

Fig.3 Oracle9iAs’s frequency distribution
图 3 Oracle9iAS 事务频率分布

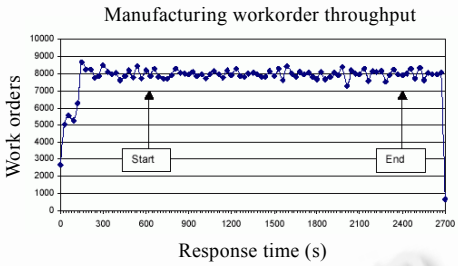

Fig.4 Weblogic’s workorder throughput
图 4 WebLogic 吞吐量图

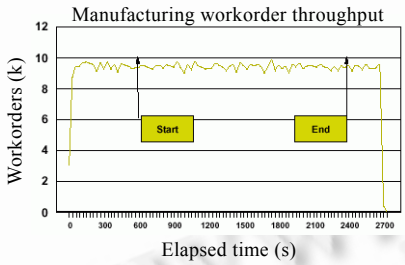

Fig.5 Websphere’s workorder throughput
图 5 Websphere 吞吐量图

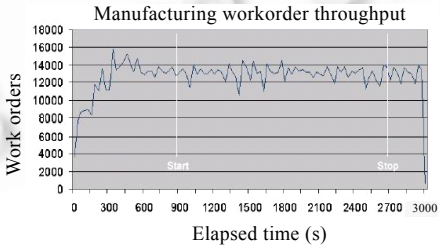

Fig.6 Oracle9iAs’s workorder throughput
图 6 Oracle9iAS 吞吐量图

Oracle9iAS 稳定以后,其吞吐量平均大约为 13 000,最大吞吐量为 14 300,远远超过了 WebLogic 和 Websphere.从图中吞吐量曲线来看,Oracle9iAS 的吞吐量受系统的影响比较大,最大和最小吞吐量相差约为 4 000,颠簸比较大,而 Weblogic 和 Websphere 则比较平稳.总之,从上述分析的结果来看,Oracle 后来居上,在性能上占有优势,不仅性能度量值(BBops/min)高于 Weblogic 和 Websphere,而且具有很高的事务吞吐量.Weblogic 和 Websphere 的吞吐量则比较平稳,亦具有一定的优势.

6 总 结

Web 应用服务器解决了传统中间件在面向 Internet 的 Web 计算环境下遇到的问题,是 Web 计算环境下产生的新型中间件,不仅具有传统事务监控器的高可伸缩性、高可用性、高可靠性和高效等高级特性,而且为事

务性 Web 应用提供一个跨平台的运行环境、一系列通用服务以及相应的工具集.Web 应用服务器目前主要有脚本、面向对象和对象 3 种模式.J2EE 应用服务器是对象模式应用服务器中较突出的一种,得到广大研发人员和企业的青睐,具有很好的前景.

ECperf 是 J2EE 应用服务器的评测基准,通过性能度量值(BBops/min)、业务事务的响应时间频率分布图和吞吐量图对 Web 应用服务器的可伸缩性、性能进行全面的仿真评测.通过 ECperf 公布的最新数据,对 BEA Weblogic, IBM Websphere 和 Oracle9iAS 三大主流应用服务器进行分析和比较可知,Oracle9iAS 是目前性能最优、事务吞吐量最大的 Web 应用服务器.在 QoS 服务、企业应用集成以及配套工具等功能方面,虽然各自具有一定的优势,但相差无几.

Web 应用服务器覆盖了 Web 计算环境下计算机软件技术的诸多领域,如软件工程、分布计算、面向对象、事务处理、工作流管理、应用集成和网络安全等技术.虽然 Web 应用服务器已逐步成熟,但在满足不断变化的 QoS 需求方面,Web 应用服务器还存在着开放性和灵活性差、便利性和自适应能力低以及客户透明性不强等不足之处.目前,几个较为突出的研究方向包括开放的 Web 应用服务器体系结构研究、可动态重配的组件容器、自适应负载平衡、中间层高速缓存技术、Web 应用服务器高可靠性研究以及事务性 Web 应用开发和测试方法等等.此外,随着 Web Services 规范的不断完善以及 Web Services 复杂应用的逐步增多,如何快速地支持这些规范,有效地支持 Web Services 应用的开发、部署、执行、管理和复合,为 Web Services 提供 QoS 服务基础设施,也成为 Web 应用服务器需要进一步解决的问题和研究热点.Web 应用服务器将集成更多的中间件服务,逐步向完整的事务性 Web 应用基础平台发展,最终成为事务性 Web 应用的“操作系统”.

References:

[1] Bakken DE. Middleware, Encyclopedia of Distributed Computing. Kluwer Academic Press, 2001. http://www.eecs.wsu.edu/~bakken/middleware-article-bakken.pdf.

[2] Geibs K. Middleware challenges ahead. IEEE Computer, 2001,34(6):24~31.

[3] Mattern F, Sturm P. From distributed systems to ubiquitous computing—The state of the art, trends, and prospects of future networked systems. In: Proceedings of the Symposium ‘Trends in der Informations technologie am Beginn des 21. Jahrhunderts’. 2002. 109~134. http://www.inf.ethz.ch/vs/publ/papers/DisSysUbiComp.pdf.

[4] Sadoski GTE. Client/server software architecture—An overview. Technical Report, 3/2003, Software Engineering Institute (SEI), Carnegie Mellon University, 2000.

[5] Ritter D. The middleware muddle. ACM SIGMOD Record, 1998,27(4):86~93.

[6] Ricciuti M. Application server eludes definition. CNet News, 1998. http://news.com.com/2100-1001-214783.html?legacy=cnet.

[7] Mohan C. Tutorial: Application servers and associated technologies. In: SIGMOD, ed. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002. 636~636.

[8] Copeland R. Web application servers. Technical Report, 1999. http://www.informationweek.com/704/04iuapp.htm.

[9] McIsaac K. J2EE App server evolution: Survival of the Cheapeast? Technical Report, META Group Research, 2002. http://www.metagroup.com/cgi-bin/inetcgi/jsp/ displayArticle.do?oid=29616.

[10] Mohan C. Application servers: Born-Again TP monitors for the Web. In: SIGMOD, ed. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2001. 622~622.

[11] Altinel M, Luo Q, Krishnamurthy S, Mohan C. DBCache: Database caching for Web application servers. In: Franklin MJ, Moon B, eds. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002. 612.

[12] Mohan C. Tutorial: Caching technologies for Web applications. In: Proceedings of the 27th International Conference on Very Large Data Bases. Rome, 2001. http://www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf.

[13] Anton J, Jacobs L, Liu X. Web caching for database applications with Oracle Web cache. In: Franklin MJ, Moon B, eds. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002. 594~599.

[14] The TimesTen Team. Mid-Tier caching: The TimesTen approach. In: Franklin MJ, Moon B, eds. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002.

[15] Luo Q, Krishnamurthy S, Mohan C. Middle-Tier database caching for e-business. In: Franklin MJ, Moon B, eds. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002. 600~611.

[16] The TimesTen Team. Hight performace and scalability through application-tier, in-memory data management. In: Abbadi AE, Brodie ML, Chakravarthy S, eds. Proceedings of the 26th International Conference on Very Large Databases. San Fransisco: Morgan Kaufmann Publishers, 2000. 677~680.

[17] Owendo Technology. Application server panorama. 2002. http://e-serv.ebizq.net/aps/owendo_1a.html.

[18] Roman E, Oberg R. The technical benefits of EJB and J2EE technologies over COM+ and windows. Middleware Company, 1999. http://java.sun.com/products/ejb/pdf/j2ee_dnatwp.pdf.

[19] Sun Microsystems. J2EE compatibility. 2003. http://java.sun.com/j2ee/compatibility.html.

[20] Deshpande S. Eight reasons Ecperf is the right way to evaluate J2EE performance. 2002. http://ecperf.serverside.com.

[21] SPEC. SPECjAppServer2001 frequently asked questions. Technical Report, 2001. http://www.spec.org/jAppServer2001/docs/FAQ.html.

[22] Dietzen S. Web application server architecture. In: Proceedings of the 9th International Workshop on High Performance Transaction Systems (HPTS). 2001. http://www.research.microsoft.com/~jamesrh/hpts2001/presentations/WebAppServerArchitecture.ppt.

[23] Stonebraker M. Too much middleware. ACM SIGMOD Record, 2002,31(1):97~102.

[24] Java-application-servers.com. Application server definition. Technical Report, Barry & Associates, Inc., 2002.

[25] Dolgicer M. A detailed look at Java application servers. Technical Report, International System Group, 2001.

[26] Kassam R. Special topics: Application servers. Technical Report, Department of Computer Science University of Toronto, 2002.

[27] Bates J. The state of the art in distributed and dependable computing. A CaberNet Sponsord Report, 1998. http://citeseer.nj.nec.com/bates98state.html.

[28] Astley M, Sturman DC, Agha G. Customizable middleware for modular distributed software. Communications of the ACM, 2001,44(5):99~107.

[29] Blair GS, Coulson G, Robin P, Papathomas M. An architecture for next generation middleware. In: Davies N, Raymond K, Seitz J, eds. Proceedings of the Middleware'98. Springer-Verlag, 1998. 191~206.

[30] Truyen E, Bo Nørregaard Jørgensen. Component architecture for dynamic reconfiguration of object request brokers. In: Proceedings of the Middleware 2000 Conference. New York: Springer-Verlag, 2000.

[31] Parlavantzas N, Coulson G, Blair GS. Applying component frameworks to develop flexible middleware. In: Cazzola W, eds. On-Line Proceedings of ECOOP 2000 Workshop on Reflection and Metalevel Architectures. 2000.

[32] Blair GS, Coulson G. The case for reflective middleware: Building middleware that is flexible, reconfigurable and yet simple to use. Communications of the ACM, 2002,45(6):33~38.

[33] Parlavantzas N, Coulson G, Clarke M, Blair G. Towards a reflective component-based middleware architecture. In: Cazzola W, eds. On-Line Proceedings of ECOOP 2000 Workshop on Reflection and Metalevel Architectures. 2000. http://citeseer.nj.nec.com/331827.html.

[34] Bo Nørregaard Jørgensen, Truyen E, Matthijs F, Joosen W. Customization of object request brokers by application specific policies. In: Proceedings of the Middleware 2000 Conference. New York: Springer-Verlag, 2000.

[35] Bruneton E, Riveill M. Javapod: An adaptable and extensible component platform. In: Proceedings of the Middleware 2000 Conference. New York: Springer-Verlag, 2000. http://www.comp.lancs.ac.uk/computing/ rm2000/paper-list.htm.

[36] Welch IS. The glasshouse—A reflective container for mobile code. In: Proceedings of the 7th ECOOP Workshop on Mobile Object Systems Development of Robust and High Confidence Agent Applications 2001. 2001. http://cui.unige.ch/~ecoopws/ws01/papers/glasshouse.pdf.

[37] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 2002. 223~325.

[38] Barga R, Pu C. Reflection on a legacy transaction processing monitor. In: Proceedings of the Reflection'96. San Francisco, 1996. http://citeseer.nj.nec.com/barga96reflection.html.

[39] Prochazka M. Advanced transactions in component-based software architectures [Ph.D. Thesis]. Charles University, 2002.

[40] Wu ZX. Reflective Java and a reflective component-based transaction architecture. 1998. http://www.csg.is.titech.ac.jp/~chiba/oopsla98/proc/wu.pdf.

[41] Yu WH. Reflective transaction processing—Motivations and issues. 2001. http://www.nik.no/2001/20-yu.pdf.

[42] Prochazka M. Extending transactions in enterprise JavaBeans. Technical Report, 3/2000, Prague: Department of Software Engineering, Charles University, 2000.

[43] Chaudhary AS, Saleem MA, Bukhari HZ. Web services in distributed applications—Advantages and problems. 2002. http://citeseer.nj.nec.com/548303.html.

[44] Myerson JM. Web service architectures. 2003. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf.

[45] Tartanoglu F, Issarny V, Levy N, Romanovsky A. Dependability in the Web service architecture. In: Proceedings of the WADS. 2002. http://citeseer.nj.nec.com/tartanoglu02dependability.html.

[46] Cotroneo D, Gargiulo M, Russo S, Ventre G. Improving the availability of Web services. In: Proceedings of the ICSE 2002 Workshop on Architecting dependable Systems. Orlando, 2002. http://www.cs.kent.ac.uk/events/conf/2002/wads/Proceedings/cotroneo.pdf.

[47] Koschel A, Klaus Hofmann zur Linden. Interoperability of standards: Web services & .NET, EJB and CORBA. Technical Report，IONA Technologies, Brunnenweg Weiterstadt, 2002.

[48] Snell J. Securing Web services. IBM, 2002. http://www.ibm.com/software/solutions/webservices/pdf/wp_securingws.pdf.

[49] Hondo M, Nagaratnam N, Nadalin, A. Securing Web services. IBM systems Journal, 2002,41(2):1~5.

[50] Feng YL, Huang T, Jin BH. Network Distributed Computing and Software Engineering. Beijing: Science Press, 2003. 206~210 (in Chinese).

[51] Mikalsen T, Rouvellou I, Tai S. Reliability of composed Web services—From object transactions to Web transactions. In: Proceedings of the OOPSLA Workshop on Object-Oriented Web Services. Tampa, 2001. http://www.research.ibm.com/AEM/pubs/web_services_oopsla2001.pdf.

[52] Jie Wu. Distributed System Design. Beijing: China Machine Press, 1999. 192~243 (in Chinese).

[53] IONA. Orbix 2000. http://www.iona.com/docs/orbix2000.html.

[54] Othman O, Schmidt DC. Issues in the design of adaptive middleware load balancing. In: ACM SIGPLAN Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems. New York: ACM Press, 2001. 205~213.

[55] Othman O, O'Ryan C, Schmidt DC. An efficient adaptive load balancing service for CORBA. IEEE Distributed Systems Online, 2001,2(3):1~8.

[56] Othman O, O'Ryan C, Schmidt DC. Strategies for CORBA middleware-based load balancing. IEEE Distributed Systems Online, 2001,2(3):10~12.

[57] Candan K, Li W, Luo Q, Hsiung W, Agrawal D. Enabling dynamic content caching for database-driven Web sites. In: Proceedings of the ACM SIGMOD 2001. Santa Barbara, 2001. http://www.cs.wisc.edu/~chenl/cache/ invalidation_sigmod01.pdf.

[58] Challenger J, Iyengar A, Dantzig P. A scalable system for consistently caching dynamic Web data. In: Proceedings of the IEEE INFOCOM'99. New York, 1999. http://www.csd.uch.gr/~hy558/papers/infocom2.pdf.

[59] Challenger J, Iyengar A, Witting K, Ferstat C, Reed P. A publishing system for efficiently creating dynamic Web content. In: Proceedings of the IEEE Infocom 2000 Conference. Tel-Aviv, 2000. http://citeseer.nj.nec.com/challenger00publishing.html.

[60] Degenaro L, Iyengar A, Lipkind I, Rouvellou I. A middleware system which intelligently caches query results. In: Proceedings of the ACM/IFIP Middleware 2000. Palisades, 2000. http://citeseer.nj.nec.com/degenaro00middleware.html.

[61] Oracle Corporation. Oracle Internet application server documentation library. Technical Report, Oracle, 2003. http://otn.oracle.com/products/ias/web_cache/content.html.

[62] Theserverside.com. Top ten results by performance. Technical Report, 2002. http://www.theserverside.com/ecperf/index.jsp.

附中文参考文献:

[50] 冯玉琳,黄涛,金蓓弘.网络分布计算和软件工程.北京:科学出版社,2003.206~210.

[52] 吴杰.分布式系统设计.北京:机器工业出版社,1999.192~243.