Web 应用服务器自适应负载平衡服务*

范国闯+, 朱 寰, 黄 涛, 冯玉琳

(中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)

Towards Adaptive Load Balancing Services for Web Application Servers

FAN Guo-Chuang+, ZHU Huan, 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, Fax: 86-10-62562538, E-mail: fanguo@otcaix.iscas.ac.cn
http://otcaix.iscas.ac.cn
Received 2002-10-25; Accepted 2003-01-28

Fan GC, Zhu H, Huang T, Feng YL. Towards adaptive load balancing services for Web application servers. *Journal of Software*, 2003,14(6):1134~1141.
http://www.jos.org.cn/1000-9825/14/1134.htm

Abstract: WAS (Web application server) is a distributed system which provides runtime integrated services for transactional Web applications. To improve the dependability and scalability of WASs, one effective way is to balance loads via adaptive load balancing service based on the middleware. The existing adaptive load balancing services, however, have inadequacies such as lack of server transparency and extensible load balancing strategy etc., and provide insufficient functionality to satisfy WASs. In this paper, the key requirements of adaptive load balancing services for WASs are addressed first, and then the design of a novel adaptive load balancing service is presented. The key design challenges including hot plug-in, customizable load balancing strategy, load feedback and adaptive control, state migration and fault tolerance etc. are described, and the technical solutions are outlined. Lastly, a comparative study with other related works is given The adaptive load balancing service is developed and implemented in WebFrame2.0 application server by using Java RMI.
Key words: Web application server; adaptive load balancing service; middleware; dependability; scalability

摘 要: Web 应用服务器是为事务性 Web 应用提供一系列运行时服务的分布式系统.基于中间件的自适应负载平衡服务是为 Web 应用服务器提供高可信赖性和高伸缩性的一种有效方法,但目前还存在许多不足,如缺乏服务端透明性、负载策略不可替换等,不能满足 Web 应用服务器特有的需求.分析了 Web 应用服务器负载平衡服务的关键需要,设计了一种自适应负载平衡服务,阐述了在 J2EE 应用服务器 WebFrame2.0 上实现该服务的若干关键技术及其解决办法,包括可热插拔、负载策略可替换、负载反馈与自适应控制、状态迁移以及容错技术等,最后是相关工作介绍及其比较.该负载平衡服务已在 Web 应用服务器 WebFrame2.0 中得以实现.
关键词: Web 应用服务器;自适应负载平衡服务;中间件;可信赖性;可伸缩性
中图法分类号: TP393 **文献标识码**: A

* 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.

Web 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)的分布式系统[1].为了满足 Web 计算环境下大规模用户的并发访问,Web 应用服务器需要提供可信赖性(dependability)、高伸缩性(scalability)等 QoS 特性[2],提高系统整体性能和吞吐量.使用基于中间件(如 Java RMI[3],COBRA[4],COM[5])的负载平衡服务[6],一方面,可将客户请求均衡地转发到后端应用服务器进行处理,提高系统的可伸缩性;另一方面,它能动态地对硬件或软件故障引起的变化进行调整,增强系统的可信赖性.负载平衡中间件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.如在文献[7]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,不能应用到具有更复杂负载平衡需求的分布式系统.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[6].

自适应负载平衡服务目前还存在许多不足之处,如缺乏服务端透明性、仅支持无状态复件(stateless replica)的负载平衡、不支持容错、固化的负载策略和负载度量等等[8~10],不能满足 Web 应用服务器如下一些关键需求:(1) 不仅需要支持基于进程、TCP/IP 地址、系统服务等粗粒度的负载平衡,而且需要支持组件负载平衡模型[10],该模型不仅支持无状态的组件,而且支持有状态的实体组件,能够协调多个组件的交互;(2) 客户不需要修改代码就能将客户请求透明地分发到后端应用服务器处理.同样,在系统运行期间,可以适时地添加或删除服务端组件,不影响对客户的服务,不更改服务端应用代码就可以提供负载平衡服务;(3) 尽可能均衡后端应用服务器的负载,最大化系统可伸缩性,提高系统资源的利用率;(4) 虽然负载平衡服务不需要提供完全的容错能力,但能有效地处理应用服务器出现失败、崩溃的情况,提高系统可信赖性;此外,负载平衡服务自身必须容错,不会出现单点失败;(5) 不同的应用对负载有不同的要求,对负载度量和负载平衡策略有不同的语义,负载平衡服务需要支持客户定制的负载度量和负载平衡策略,在运行期间可以替换负载度量和负载平衡算法.

WebFrame2.0 是我们自行研制的 J2EE 应用服务器,提供组件生命周期管理、数据库连接管理、分布式事务、命名服务、消息服务等规范中规定的功能和服务.由于 J2EE1.3 规范并没有对负载平衡约定标准接口,也没有指定负载平衡机制,因此,需要扩展 J2EE1.3 规范的基本功能,实现负载平衡服务,满足 Web 应用服务器的上述需求.为此,本文设计一种自适应负载平衡服务,并分析了在 WebFrame2.0 上实现负载平衡服务的一些关键技术.

1 自适应负载平衡服务的设计

WebFrame2.0 支持纵向和横向集群拓扑结构,其集群逻辑拓扑结构如图 1 所示,一个 WebFrame 集群由多个 WebFrame 应用服务器组成,在每个 WebFrame 应用服务器上运行一个负载平衡服务,该负载平衡服务能够根据所有后端服务器运行状态动态地选择一个 EJB 复件(replica)响应客户的请求.EJB 复件是同一 EJB 组件在不同服务器上的实例.一个集群由多个子群(sub cluster)构成,在同一子群中的某个应用服务器在运行期间备份另外某个应用服务器 EJB 复件的状态.负载平衡服务将请求在集群中所有 EJB 复件进行分配.

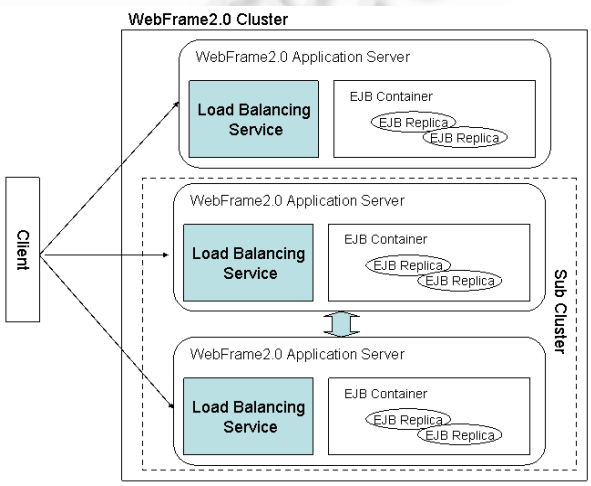

Fig.1 WebFrame cluster architecture

图 1 WebFrame 集群拓扑结构

WebFrame2.0 负载平衡服务组件结构如图 2 所示.根据 J2EE 规范,EJB 由容器(container)管理,客户不能直接访问 EJB 实例.为了实现负载平衡,需要定位 EJB 所在的容器,然后对客户请求进行分配,为此,设计了 Container Load Balancer 和 Request Load Balancer.Container Load Balancer 根据集群服务器的负载情况,选择一个负载轻的 EJB 容器响应客户创建 EJB

的请求,返回 EJB 在客户端的动态代理 Replica Proxy.Request Load Balancer 有负载和容错两大功能,它能根据 Replica 的负载情况或强制转发请求,选择 Replica 响应客户的方法调用;它还能够感知 Replica,Container 和服务器失败的情况,将客户的请求转发到备份服务器.Object Request Manager 从 Container Locator 获得 Container,然后将需要创建的 EJB 通过 JNDI 名字绑定在选定的 Container 上,返回 EJB 在客户端的动态代理 Replica Proxy,客户可以通过这个 Proxy 访问后端的 EJB 对象.Container Locator 根据 Container Load Analyzer 对 Container 负载分析的结果,定位一个负载轻的 Container.Replica Locator 与 Container Locator 类似,所不同的是,其功能是定位于一个负载轻的 Replica 上.Container Load Analyzer 应用负载平衡策略,选定一个轻负载的 Container 返回给 Container Locator,可以在运行期间指定负载平衡策略,满足系统负载动态变化的要求.Replica Load Analyzer 与 Container Load Analyzer 功能类似,其负载分析对象是 Replica.Container Load Monitor 主要监视 Container 的负载以及 Container 所在的应用服务器系统负载,如内存、CPU 等.Replica Load Monitor 有 3 大功能:(1) 监视 Replica 的负载;(2) 将负载监视结果发送给 Request Load Balancer;(3) 处理 Request Load Balancer 的请求转发消息,通知 Replica 转发或接受客户的请求.Replica Proxy 是 Replica 在客户端的代理,在运行期间动态产生,Request Load Balancer 通过这个 Replica Proxy 区分不同的 Replica,截取客户所有调用请求.Containerinvoker 是 Container 的入口,接收远程客户端调用,并将其传递给 Container.

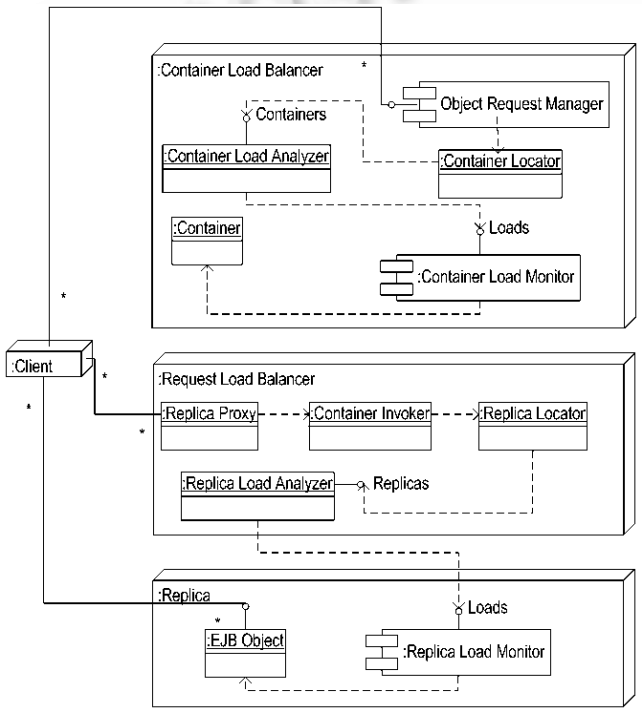

Fig.2 Components in WebFrame load balancing service

图 2 WebFrame 负载平衡服务组件结构图

从图 2 可以看出,应用服务器通过 Load Monitor 监视系统负载,Load Analyzer 根据负载情况在运行期间应用不同负载策略将负载在多个 Replica 平衡,解决了非自适应负载平衡服务不可感知的问题;Replica Load Monitor 向 Request Load Balancer 提供负载的反馈信息,Request Load Balancer 根据反馈结果向 Replica Load Monitor 发送请求转发的控制请求,使应用服务器在运行期间对系统负载的变化具有自适应的调整能力.负载平衡服务的交互如图 3 所示.当 Replica 接收到 Load Monitor 的“拒绝服务”的控制信息后,将 LOCATION FORWARD 消息发送给 Request Load Balancer,Request Load Balancer 接收到 LOCATION FORWARD 消息后,调用 Replica Proxy 的 Location_Forward 方法更改 Replica 所在的容器在客户端的存根,这样,客户可以继续使用原来的 Replica Proxy 向 Replica 发送调用请求.

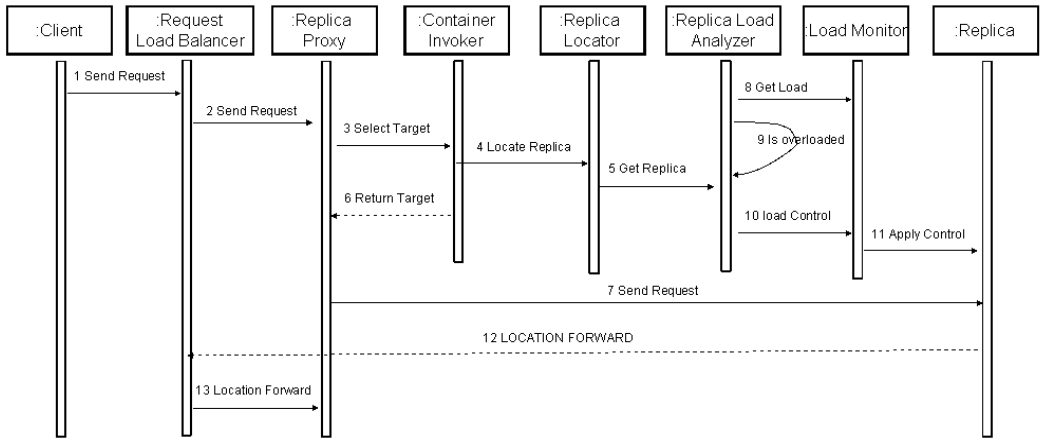

Fig.3 Interaction diagram for WebFrame load balancing service

图 3 WebFrame 负载平衡服务交互图

2 关键技术及其解决办法

2.1 负载平衡服务的高可用技术

自适应负载均衡服务一般采取集中的方式.集中方式虽然易于设计与实现,但是集中方式的负载平衡服务集中处理所有负载任务,分发所有客户请求,是系统的一个瓶颈.当负载平衡服务出现失败时,Web 应用服务器则不能接收客户的请求,出现单点失败,降低了系统的可靠性和可用性,不能满足 Web 应用服务器 24×7 的不间断运行的要求.为此,我们采取分散式负载平衡服务.分散式避免集中式遇到的上述问题,提高了系统的容错能力和可用性.如图 4 所示,每个应用服务器上都运行相同负载平衡器(load balancer),多个负载平衡器管理系统的负载,形成一个逻辑上的负载平衡服务.负载平衡器之间相互协调工作,将各自的负载信息发送给其他负载平衡器,每个负载平衡器根据各自收集到的负载信息通过 Load Analyzer 进行负载决策.

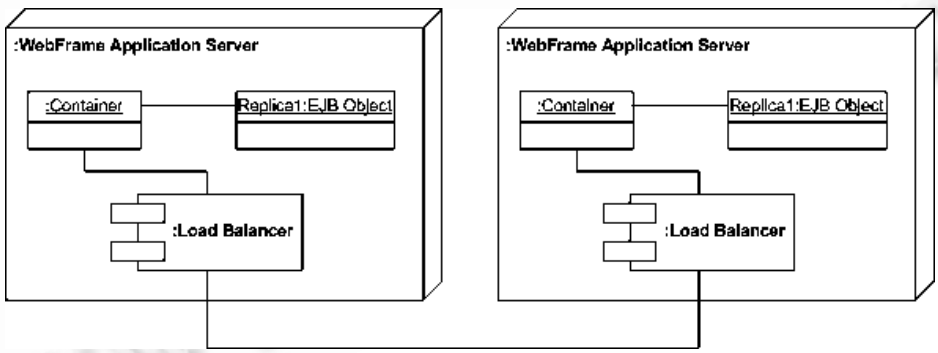

Fig.4 WebFrame decentralized load balancing service

图 4 WebFrame 分散式负载平衡服务

2.2 负载平衡服务的透明、可热插拔技术

为了实现客户和后端 EJB 组件的通信,Web 应用服务器需要接受客户直接请求或负载平衡服务的转发请求.虽然大多数 EJB 组件需要负载平衡,但是可能有一部分后端 EJB 组件不需要这种能力,为此,Web 应用服务器需要在运行期间根据不同要求决定是否提供负载平衡服务,不更改应用服务器代码实现负载平衡服务的热插拔(hot plug-in),从而实现负载平衡服务的透明性.采用 Interceptor 设计模式[11],如图 5 所示,Web 应用服务器在接收到客户请求这个事件的时候,Load Balancing Interceptors 透明地将请求转发给负载平衡服务,然后负载平衡服务将负载控制命令通过 Request Forward 例外的形式返回给 Load Balancing Interceptor,Replica Proxy 捕获这个例外后,将请求重新定向到指定的 EJB,最后调用 EJB 实例的方法.为了提供真正的服务端的透明性和服务可

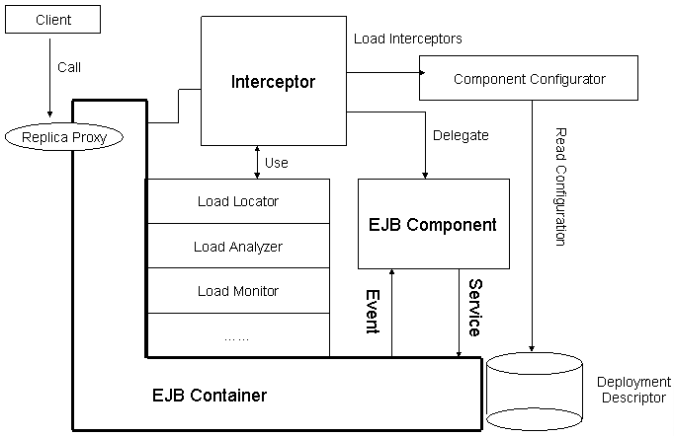

Fig.5 WebFrame transparent load balancing service
图 5 WebFrame 透明的负载平衡服务

转发,在运行期间动态地、透明地添加拦截请求的 Interceptor.

热插拔,需要在运行期间动态地、透明地为每个 EJB 加载 Load Balancing Interceptors 到应用服务器.Component Configurator 设计模式[11]可以实现在运行期间动态地将一个服务加载到应用,具体来说,Component Configurator 读取 EJB 的配置文件,然后将 Load Balancing Interceptors(如 Load Locator,Load Monitor 等)加载到 Interceptor 链.当接收到函数调用等事件时,自动地按照装载顺序触发相应的这些服务.这样,基于 Interceptor 和 Component Configurator 设计模式,在没有更改应用服务器代码的情况下,负载平衡服务能够透明地将客户请求

2.3 负载反馈与自适应控制

为了有效地在多个 Replica 上进行负载均衡,负载平衡服务在事先不知道负载度量类型的情况下,可计算出当前 Replica 的负载.当 Replica 负载重时,Replica 拒绝请求,将请求重定向给负载平衡器;当负载轻时,负载平衡器迫使 Replica 又能接收客户请求.为此,使用负载反馈和自适应控制机制.如图 6 所示,负载平衡器向 Load Monitor 询问 Replica 的负载或 Load Monitor 将负载主动发送给负载平衡器.负载平衡器接收到负载反馈后,根据系统整体负载情况分发请求,将负载控制消息发送给 Load Monitor,并设置当前 Replica 的负载状态.Load Monitor 基于 Strategy[12]设计模式,可对不同类型的资源进行监测,用户只需定义不同的对象实现即可.在对象实现里,用户可以采用不同的度量技术和策略获取给定资源的负载,

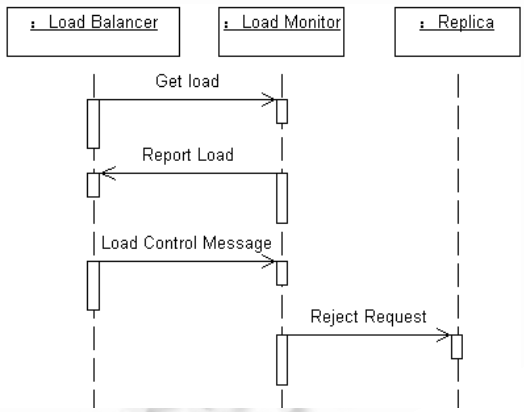

Fig.6 WebFrame load feedback and control
图 6 WebFrame 负载反馈与自适应控制

通过相同的接口向负载平衡器反馈负载.由于没有将负载度量绑定在负载平衡器中,负载平衡服务可以基于不同类型的负载度量进行均衡.负载平衡器和 Replica 并不直接通信,而是通过一个 Mediator[12]进行交互,这个 Mediator 就是 Load Monitor,Load Monitor 响应负载平衡器的请求,根据请求的类型,Replica 决定是否继续接收请求还是拒绝请求.

2.4 负载平衡算法的可替换技术

许多负载平衡服务只支持一些常见的负载平衡算法,如 Round Robin 等,但是这些负载平衡算法远远不够,可以根据不同的情况使用不同的负载分析技术,某些应用需要动态更改负载平衡算法(如在不同的访问时间段使用不同的算法),以便适应新的负载要求.此外,开发人员可能需要评估不同负载平衡算法在 Web 应用服务器中负载的效果和性能,也可能需要在测试、部署时定义一种应用相关或综合的负载平衡算法.任何负载平衡服务都不能实现所有用户所需要的负载平衡算法.为此,自适应负载平衡服务需要在运行期间动态配置负载平衡服务,如 Load Monitor 和 Load Analyzer,以替换负载平衡算法.

结合 Component Configurator 和 Strategy 设计模式,负载平衡算法可以替换模型,如图 7 所示.LBStrategy Configurator 使得应用在运行期间能动态装载或卸载负载策略实现,用户就可以根据不同要求和应用场景在不停止服务器的情况下动态替换负载平衡算法,具有很好的扩展性.Strategy 设计模式应用于负载平衡算法的实

现,每个负载平衡策略(strategy)有相同的接口,但却有不同的负载分析技术和算法.WebFrame 负载平衡服务支持 Round Robin,First Available,Weight Based,随机以及模糊的自适应策略等.模糊的自适应策略比 Round Robin 算法更复杂,允许 Replica 间负载有差别的情况,尽可能地将负载平均分配到各个 Replica 上.之所以称之为"模糊"是因为该策略以整体性能为考虑,不精确考虑到每个 Replica 的负载而进行任务迁移.模糊的自适应策略描述如下:(1) 每当负载平衡器完成负载决策后,计算出 Replica 的平均负载,表示为 AverageLoad;(2) 比较各个 Replica 的负载与平均负载的差,表示为 LoadDiff;(3) 如果 LoadDiff 的值大于系统设定的负载差异阈值 $LoadDiff_{Max}$ 的 N 分之一,负载平衡器则将负载迁移到低负载的 Replica,其中 N 是当前系统所有 Replica 的数目.模糊的自适应策略在某段时间内,在每个 Replica 的负载可能不完全相等,但是随着时间的延长和 Replica 数目的增加,每个 Replica 的负载趋向平衡.许多研究人员在负载平衡算法方面做了大量的工作[13],他们提出的大多数负载平衡算法、策略都可以集成到 WebFrame 自适应负载平衡.

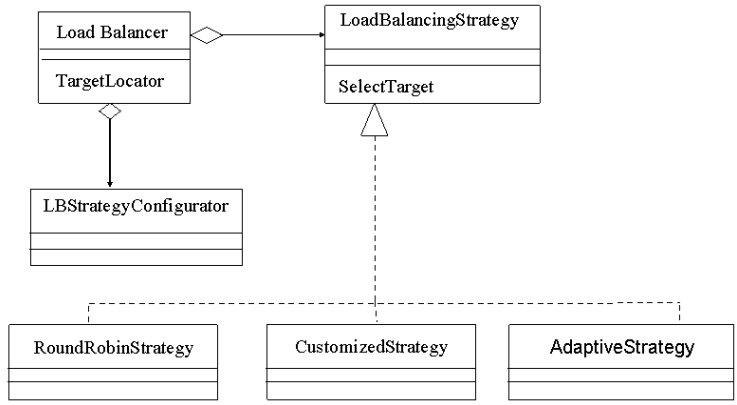

Fig.7 WebFrame replaceable load balancing strategy
图 7 WebFrame 负载平衡策略可替换模型

2.5 负载平衡服务的容错与状态迁移技术

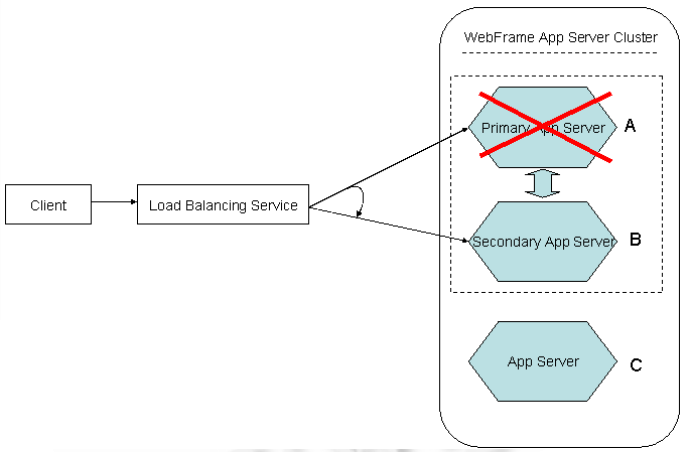

Fig.8 WebFrame fault tolerant load balancing
图 8 WebFrame 容错的负载平衡服务

如图 8 所示,当后端 Web 应用服务器崩溃的时候,客户就不能访问该服务器上的所有 Replica,自适应负载平衡服务不仅需要将客户的请求转发到备份的 Web 应用服务器,而且不影响转发后请求的执行结果,也就是说,自适应负载平衡服务需要具有较强的容错能力,能透明地处理应用服务器崩溃的情况,客户根本不会感知请求已转发,但请求会继续正常执行.为了实现自适应负载平衡服务能透明地处理系统失效,要求后端 Replica 的状态必须一致.Web 应用服务器中的 Replica 主要有无状态(stateless)、有状态(stateful)和实体(entity)3 种类型,它们具有各自不同的特点,负载平衡服务针对不同类型的 Replica 请求具有不同的容错处理策略.对于 Stateless Session Bean 来说,由于不保存任何状态信息,其请求转发策略很简单,负载平衡服务只需将客户任意调用路由到任何一台已部署该 Bean 的可用服务器即可;然而对于 Statefull Session Bean 的请求,由于同一 Replica 实例的不同请求之间存在着状态相关性,因此对它们进行请求转发前必须确保状态的一致性.为此,Web 应用服务器在客户创建 EJB 对象时,负载平衡服务选择一个备份服务器(secondary server)保存 Statefull Session Bean 状态的备份.请求执行后,Statefull Session Bean 状态的改变也同时复制到备份服务器中.当主服务器不可用时,自适应负载平衡服务自动地把客户的请求指向备份服务器(如图 8 所示),此时,备份服务器才在本地创建 EJB 实例,恢复 EJB 实例以前的状态,备份服务器继续响应客户的请求,从而完成客户请求的透明切换.此时的备份服务器已升级为主服务器,Web 应用服务器又为这个新的主服务器选择一个备份服务器,最后自适应平衡服务每次完成客户请求调用后,自动刷新 EJB 实例其备份服务器的信息.对于 Entity Bean,它的状态数据保存在数

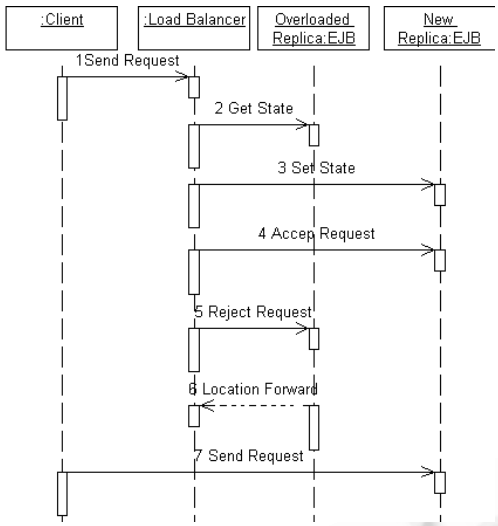

Fig.9 WebFrame load balancing stateful replicas
图 9 WebFrame 有状态复件负载平衡

据库中.当某个 Entity Bean 不可用时,负载平衡服务将客户调用转向备份的服务器,重新创建实例,并将 Entity Bean 的状态从数据库中重新装载到实例中,最后修改 Entity Bean 所在的主服务器、备份服务器的位置信息.

Stateful Session Bean 和 Entity Bean 都是有状态的,统称为 Stateful Replica,当负载平衡服务将请求从过载 Stateful Replica 转发到轻负载 Replica 时,必须进行状态迁移,实现状态的持久性、一致性.应用 Memento 设计模式[12],通过 Get_State 和 Set_State 方法实现状态迁移.Stateful Replica 的负载平衡序列图如图 9 所示,通过如下操作将请求转发到一个低负载的 Stateful Replica:(1) 负载平衡器透明地接收客户发出的请求;(2) 为了把负载迁移到一个新的 Stateful Replica,负载平衡器调用 Get_State 操作从文件、数据或内存中获得重负荷 Replica 的状态;(3) 负载平衡器调用 Set_State 操作将状态恢复、载入到轻负载的 Stateful Replica;(4) 当状态迁移成功完成后,负载平衡器通知新的 Stateful Replica 可以处理客户的请求;(5) 负载平衡器通知负载重的 Stateful Replica 将所有客户请求都转发给负载平衡器;(6) 负载重的 Stateful Replica 可以拒绝客户的请求;(7) 客户将请求发送给新的 Stateful Replica.

2.6 负载颠簸与负载成群问题

自适应负载平衡服务可以提高 Web 应用服务器的响应速度,但是可能由于负载快速变化而导致系统不稳定.由于 Web 应用服务器的负载粒度小,负载平衡服务就可能不断地在两个 Replica 之间快速地切换负载,这种现象称为负载颠簸.负载成群是指,当有一个轻负载 Replica 可用时,负载平衡服务在很短时间内将所有请求都转发给这个 Replica 的现象.这两个现象都是由于负载平衡服务对负载动态变化反应过度造成的.有两种策略可以避免上述两种现象对系统整体性能的影响.第 1 种是增大 Load Monitor 的负载取样时间间隔 T,但不超过系统容许的最大时间间隔 T_{max}.由于负载颠簸降低系统性能的原因在于负载平衡服务"频繁"地将系统负载在多个 Replica 间进行切换,增大取样时间间隔可以使"频繁度"降低,从而减少颠簸对系统性能的影响;第 2 种办法是采用第 2.4 节所描述的"模糊"的负载平衡策略,增大负载差异阈值 $Loaddiff_{max}$.基于该策略,负载平衡器并不会在出现负载不均衡的瞬间就进行负载切换.此外,当 Replica 的数目减少时,负载切换频率尽可能地降低;当 Replica 的数目增多时,负载平衡器能自适应地多进行负载切换,以尽可能地将负载均衡.

3 相关工作

负载平衡服务主要包括网络级、操作系统级和基于中间件这 3 种.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,提供一致对外接口,克服了网络级和操作系统级负载平衡服务的不足,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,根据请求内容进行决策,因此具有很高的灵活性.

BEA Weblogic 和 IBM Websphere 是目前两大主流的 J2EE 应用服务器,虽然它们都能对 3 种不同类型的 EJB 进行负载平衡,但是,与本文的负载平衡服务相比,它们存在如下差异:(1) 由于将负载服务硬编码到应用服务器中,从而不可热插拔,不具有便利性,只能通过一些管理工具静态地配置负载平衡策略;(2) 仅支持一些常见的负载平衡算法,如 Round Robin(缺省)、Weight-Based 和随机算法,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;(3) 由于将算法的实现硬编码在应用服务器中,负载平衡算法和负载度量都不能根据实际需要进行定制,不能替换;(4) 客户端透明性差.以 Weblogic 为例,为了实现动态的负载平衡,在处理请求时进行任务迁移,客户在调用每个对象方法之前需要显式调用 Callrouter 的方法获得对象所在的服务器名.而本文设计的负载平衡服务则不需要更改客户任何代码就可以实现负载平衡,具有很好的位置和访问透

明性.而微软通过 Application Center 为运行在 MTS 应用服务器上的 COM 组件提供负载平衡服务[14],其基本原理是由 Application Center 接受客户创建组件实例的请求,然后在事先配置好的服务器列表中根据负载平衡策略选择一个 COM 服务器响应客户的请求.客户需要更改客户端代码后负载平衡服务才可以截获请求.微软的组件负载平衡服务存在与 Bea Weblogic 和 IBM Websphere 相同的不足之处,其客户端透明性更不如 Bea Weblogic 和 IBM Websphere.

文献[6,8,9]中的负载平衡服务基于 COBRA 规范的 Request-Forwarding 机制对客户请求进行自适应负载分配,相对于上述主流 Web 应用服务器,具有较好的透明性和便利性.本文的工作与之相比,具有如下优点:(1) 灵活性好.由于在 J2EE 规范中没有提供相应的负载平衡机制和接口约定,其具体实现策略则可以非常灵活.本文通过自定义“请求转发”例外实现客户请求转发,无论是系统服务还是具体应用都可以根据需要抛出“请求转发”例外灵活地实现负载平衡;(2) 服务器透明性高.本文的负载平衡服务作为 Web 应用服务器的系统服务可以在系统运行期间实现热插拔,还可以根据需要在不更改系统源码的情况下,对负载平衡算法的具体实现进行动态替换,具有较高的服务端透明性;(3) 客户端透明性高.通过运行期间远程下载 Replica 在客户端的代理 Replica Proxy,然后在客户端根据负载平衡策略透明地进行负载分配;此外,本文的“请求转发”不是直接返回给客户,而是由 Load Balancer 透明地处理,极大地提高了客户端透明性.

4 总 结

负载平衡服务是一种提高分布式系统整体性能和吞吐量的有效方法,满足高可信赖性(dependability)、高可伸缩性(scalability)等需求.基于中间件的负载平衡有非自适应性和自适应性两种.我们针对 Web 应用服务器负载平衡服务的特殊需求,设计了一种自适应负载平衡服务,并已在 WebFrame 应用服务器中实现.该服务克服了非自适应负载平衡服务的诸多不足,如后端不可感知、静态负载分配等,通过 Load Balancer 和 Load Monitor 之间的负载反馈和自适应控制,系统在运行期间对负载变化具有较强的自适应调整能力.

今后需要进一步研究的问题是:首先,如何尽可能地降低负载平衡和容错带来的额外开销,增大负载收益;其次,对 Web 应用服务器负载效果(如响应时间、吞吐量、自适应调整等)进行度量需要建立一个合适的评测基准;再次,根据评测基准进行性能测试和模拟以后,进行相关工作的对比分析;最后,根据对比分析结果,进一步优化、完善 WebFrame2.0 的自适应负载平衡服务.

References:

[1] 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.

[2] 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.

[3] Object Management Group. The common object request broker: architecture and specification. 2.4 ed., 2000. http://www. omg.org/docs/ptc/96-03-04.pdf.

[4] Sun Microsystems Inc. Java remote method invocation specification (RMI). 1998. http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-title.doc.html.

[5] Kindel C, Booch G. Essential COM. 2nd ed., Addison-Wesley, 2001.

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

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

[8] Othman O, O'Ryan C, Schmidt DC. An efficient adaptive load balancing service for CORBA. IEEE Distributed Systems Online, 2001,2(3). http://www.computer.org/dsonline.

[9] Othman O, O'Ryan C, Schmidt DC. Strategies for CORBA middleware-based load balancing. IEEE Distributed Systems Online, 2001,2(3). http://www.computer.org/dsonline.

[10] Lindermeier M. Load management for distributed object-oriented environments. In: IEEE Computer Society, ed. Proceedings of the 2nd International Symposium on Distributed Objects and Applications (DOA 2000). Antwerp: OMG, 2000. 59~68.

[11] Schmidt DC, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects. 2nd ed., New York: John Wiley and Sons Inc., 2000. http://www.cs.wustl.edu/~schmidt/POSA/.

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

[13] Hui C-C, Chanson ST. Improved strategies for dynamic load balancing. IEEE Concurrency, 1999,7(3):58~67.

[14] Ewald T. Use application center or COM and MTS for load balancing your component servers. 2000. http://www.Microsoft.Com/Msj/0100/Loadbal/Loadbal.Asp.