计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
44(8):1418~1428, 2007

一个基于自管理单元支持差分服务的 Web 容器

李 洋[1,2,3] 陈宁江[4] 金蓓弘[1] 左 林[1,2,3] 黄 涛[1,2]
[1](中国科学院软件研究所软件工程技术中心 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)
[4](广西大学计算机与电子信息学院 南宁 530004)
(fallingboat@otcaix.iscas.ac.cn)

A Self-Management Unit-Based and Differentiated Service-Enable Web Container

Li Yang[1,2,3], Chen Ningjiang[4], Jin Beihong[1], Zuo Lin[1,2,3], and Huang Tao[1,2]
[1](*Technology Center of Software Engineering*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100080)
[2](*State Key Laboratory of Computer Science*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100080)
[3](*Graduate University of Chinese Academy of Sciences*, *Beijing* 100049)
[4](*College of Computer and Electronic Information*, *Guangxi University*, *Nanning* 530004)

Abstract Web container conforming to J2EE specification, which provides runtime environment for servlet and JSP and adopts best-effort service mode, has become an effective platform to deploy enterprise Web applications on Internet. However, complex business Web applications require Web container to provide differentiated services that traditional Web container does not provide for requests from different clients according to role or payment *etc*. Some approaches such as admission control and priority scheduling have been applied to provide differentiated services for Web container, but they do not consider differentiated services in finer granularity so that they can only provide monotone and static strategy of differentiated services. To improve the deficiencies of traditional Web container, the requirement of differentiated services is analyzed and a differentiated service Web container (DSWC) based on self-management unit (SMU) is brought forward. In DSWC, process modules are encapsulated into SMUs, which are connected to compose a SMU chain. DSWC provides SMU-based and SMU chain-based differentiated services for requests according to SLA (service level agreement). At the same time, an adaptive strategy selection algorithm is also presented, which can adaptively select different strategies of differentiated services based on the dynamic environment. The experiments of prototype show DSWC can effectively provide differentiated services according to SLA for different type of requests.

Key words Web container; self-management unit; differentiated service; SLA; adaptive strategy selection algorithm

摘 要 随着 Internet 的飞速发展,符合 J2EE 规范的 Web 容器已经成为部署企业 Web 应用的主流平台. 同时,企业 Web 应用的多样化和复杂性的增长,迫切地要求 Web 容器提供差分服务的支持. 然而,传统的 Web 容器采用尽力而为的服务模型,无法提供差分服务支持. 已有的研究采用准入控制、优先级调度等方法来为 Web 容器提供差分服务支持,但是它们只能提供单调、静态的差分服务策略. 提出了一个基于自管理单元的 Web 容器 DSWC,它能够根据 SLA 定义提供两级差分服务支持. 此外,提出了一

收稿日期:2006-08-24;修回日期:2007-05-08
基金项目:国家自然科学基金项目(60573126);国家"九七三"重点基础研究发展规划基金项目(2002CB312005);国家"八六三"高技术研究发展计划基金项目(2006AA01Z19B,2006AA01Z180);国家科技支撑基金项目(2006BAH02A01)

个自适应差分服务策略选择算法,它能够根据动态变化的运行环境自适应地选择差分服务策略.原型系统的实验结果显示,DSWC 能够有效地为请求提供符合 SLA 定义的细粒度的差分服务支持.

关键词　Web 容器;自管理单元;差分服务;服务水平协议;自适应策略选择算法

中图法分类号　TP311.52

Internet 的发展使得符合 J2EE 规范的 Web 容器成为部署企业 Web 应用的主流平台,它为 Servlet 和 JSP 组件提供了运行时环境[1].开放、大规模的企业 Web 应用要求 Web 容器能够依据用户角色、业务逻辑以及服务付费等分类规则提供有差别的服务[2].但是,J2EE 规范仅定义了 Web 容器的基本功能需求,缺乏对服务质量差分的支持.

互联网工程任务组(IETF)通过在 IP 包中使用差分服务字段为网络层通信选择不同的路由,在网络层首先提出了差分服务(differentiated service,DS)的概念[3].之后,一些研究者将差分服务的概念引入到中间件研究领域,向不同的用户提供不同质量的服务.Web 容器作为一种重要中间件,同样需要差分服务的支持.然而,Web 容器对用户请求的分类和处理都在容器内部进行,无法直接利用网络层的差分服务提供针对非功能属性(如响应时间和吞吐量)的差分服务.

在传统的 Web 容器体系结构中(如 Tomcat[4],OnceAS[5]),一个请求处理流程由不同的功能处理模块分阶段处理,这些处理模块通过传递请求对象相互连接,并且采用"每线程-每请求"的模式对请求进行处理.这样的体系结构没有区分用户请求,也没有提供差分服务支持.针对以上不足,本文根据对差分服务需求的分析,提出了一个基于自管理单元的、支持差分服务的 Web 容器 DSWC(differentiated service-enable Web container),它依据 SLA 定义为不同的用户请求提供基于自管理单元和基于自管理单元链的差分服务,同时它还可以根据其运行状态自适应地选择自管理单元的差分服务策略,为用户提供不同差分能力的差分服务.

1　传统 Web 容器体系结构

原有 OnceAS 的 Web 容器依据传统 Web 容器体系结构(如图 1 所示)实现.在这样的体系结构中,请求的处理过程按照其功能被划分为多个处理模块(如 request packer,servlet,logging 等),各处理模块之间通过传递请求对象相互连接组成一个请求处理流程(request process flow),同时与其他处理模块(如 session manager 等)合作完成对请求的处理.其运行原理如下:当客户请求到达一个 Web 容器监控的端口(port)之后,Web 容器首先从惟一的线程池中为其分配一个处理线程,将客户请求包装为请求对象.然后,处理线程通过请求处理流程完成对请求的处理.最后,处理线程将响应结果返回给客户后回到线程池中继续等待处理其他请求.

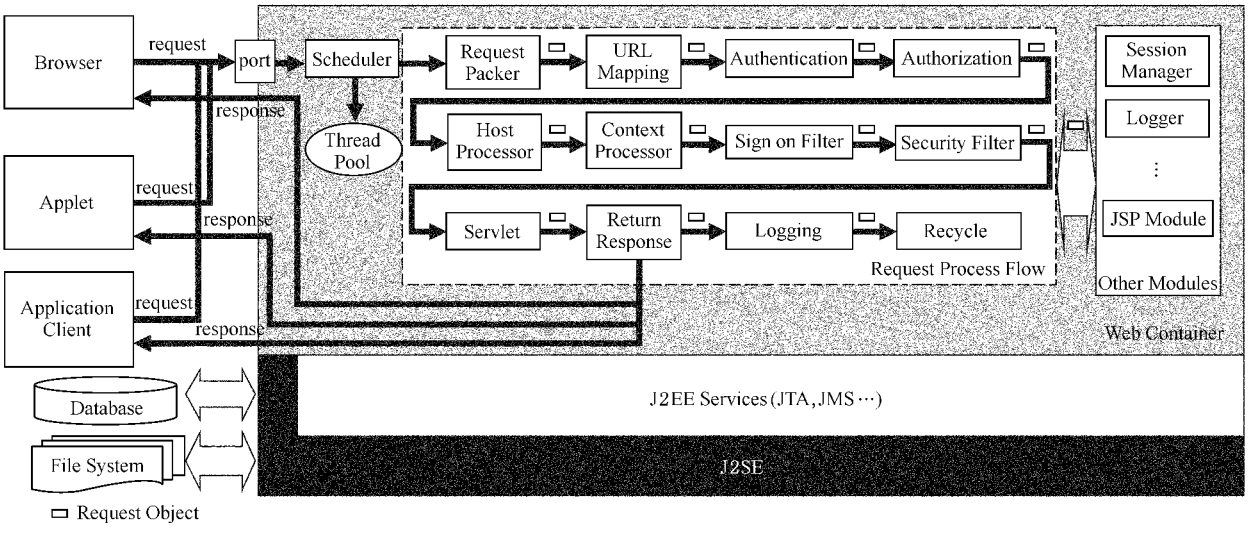


Fig. 1　The Architectuse of Traditional Web Container.

图 1　传统 Web 容器体系结构

传统 Web 容器中的请求处理流程相对固定,并且往往是由一个处理线程负责整个请求处理流程,这种每请求-每线程的方式存在如下不足:1)任何一个处理模块的阻塞都会阻塞处理线程,降低了 Web 容器的并发能力;2)无法区分来自不同用户的请求;3)不能为不同类型的请求提供有区别的处理流程;4)无法在处理模块内部为请求提供有差别的服务.

2 DSWC 设计

支持差分服务的 Web 容器应当具备以下功能:1)提供一种方式定义并分类来自不同客户的请求;2)为请求提供不同的处理流程;3)在处理模块内部为请求提供差分服务. 针对以上需求,我们修改原有 OnceAS 的 Web 容器体系结构设计如下:

首先,使用服务水平协议(service level agreement,SLA)来定义差分服务信息,包括请求类型、请求优先级等信息. SLA 是服务提供商和用户之间签署的服务协议,它规定了服务水平、服务方式、用户需求分类等内容定义,同时规定了用户的分类准则和分类类别[6]. 以往一些差分服务的工作[7]通常采用业务无关的准则对请求(如客户 IP 地址,Servlet 类型等)进行分类. 在我们的设计中,除了业务无关的分类准则之外,还允许应用开发者根据其具体业务定义业务相关的分类准则(如根据登录用户属性区分的普通用户与 VIP 用户).

其次,为了在处理模块中为请求提供差分服务,我们扩展原有的处理模块为自管理单元(SMU). 每个自管理单元拥有并管理一个独立的线程池,其中的处理线程仅负责完成所属处理模块的功能,也就是说,一个请求处理流程被分解为多个线程对请求的依次处理. 这样的体系结构设计使得一个处理模块的繁忙不会引起所有处理线程的阻塞,大大提高了 Web 容器的并发能力. 同时,将线程管理分散到各个自管理单元之后,可以在自管理单元内部为请求提供更细粒度的差分服务. 此外,自管理单元还能够根据其内部状态动态地选择不同的服务策略为请求提供差分服务.

最后,由于处理流程被分割成独立管理线程资源的多个自管理单元,一个请求的处理流程将由多个异步处理的线程合作完成,所以我们需要以一种松耦合方式来传递请求对象并连接各个自管理单元.已有研究表明,基于事件的体系结构具有很高的松散耦合性,能够满足 Web 容器的异步的请求处理流程,同时 Welsh 等人的工作[8]也证实了基于事件的体系结构具有高并发能力. 因此,DSWC 采用事件包装请求对象,借以连接不同的自管理单元,构成面向不同服务的自管理单元链.

基于以上考虑,我们给出支持差分服务的 Web 容器 DSWC 的体系结构如图 2 所示(为简明起见,一些处理模块已经被略去,后面的实验及分析也基于同样考虑):自管理单元管理器(SMU manager)创建自管理单元,并包装原有的处理模块;每个自管理单元独立管理线程资源,不同的自管理单元之间通过传递事件相互连接(图中的虚线代表自管理单元之间通过事件系统间接连接,而不是直接传递请求对象);自管理单元管理器还根据 SLA 为请求定义不同的自管理单元链,以提供不同的处理流程. 比如 SMU2 完成本模块对请求事件的处理之后,就会根据事件类型决定将请求事件分发到 SMU3 或 SMU4,从而为不同的请求事件提供不同的处理流程. 在 DSWC 中,基本的请求处理流程如下:当一个客户请求到达 Web 容器后,首先由请求包装器(request packer;为自管理单元链的第一个单元)将客户请求包装成请求事件,同时根据 SLA 定义设置其类别;然后,请求事件通过一条自管理单元链被处理,生成最后的响应对象(response)返回给客户,完成一个请求的处理. 下面将详细介绍 DSWC 的组成.

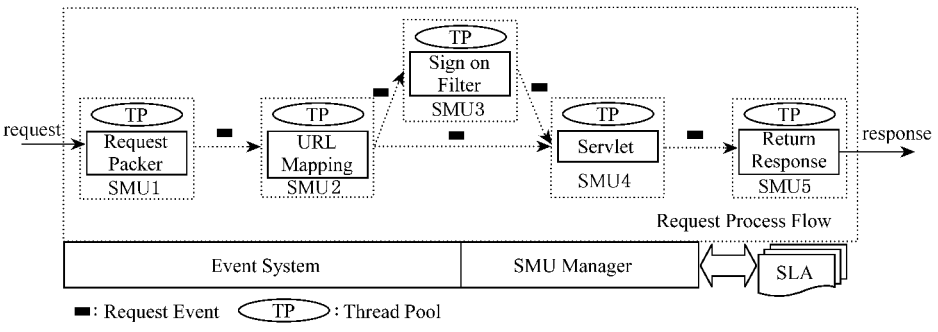


Fig. 2 The architecture of differentiated service Web container.

图 2 差分服务 Web 容器体系结构

2.1　DSWC 的组成

DSWC 主要包括以下基本组件:

1) 自管理单元(self-management unit)是 DSWC 的核心,它扩展了原有的处理模块,独立管理线程资源并处理请求事件. 同时,它还根据分发规则将处理后的请求事件分发到下一自管理单元以提供不同的请求处理流程. 最后,自管理单元还为不同请求提供了差分服务. 下一节将详细介绍其组成.

2) 请求事件(request event)包装了客户请求,它连接不同自管理单元形成请求处理流程. 一个请求事件可以定义为:*request-event* = 〈*request-type*, *request-object*〉,其中 *request-type* 与 *request-object* 分别代表请求类型和被包装的请求对象.

3) 事件系统(event system)负责请求事件传递的底层支持,它为每个自管理单元维护了一个输入事件队列,自管理单元之间通过事件系统传递请求事件相互连接.

4) SLA 定义了请求的类型以及不同请求所要求的服务水平,随 Web 应用一起被部署到 Web 容器中. 图 3 给出了 SLA 的 DTD 定义,其中 *classifier* 代表分类器. *service* 代表一类请求以及对应的服务需求:*request-type* 代表了请求类别,*priority* 代表请求的优先级,而 *smu-config* 代表了一个自管理单元的参数配置,*smu-name*, *target* 和 *property* 分别代表了自管理单元名称、请求分发目的自管理单元以及一组自管理单元属性取值(name-value 对).

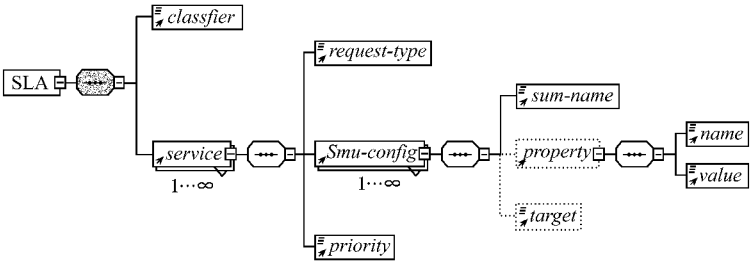


Fig. 3　DTD Definition of SLA.

图 3　SLA 的 DTD 定义

5) 自管理单元管理器(SMU manager). 自管理单元管理器管理 Web 容器中的自管理单元,它根据 SLA 定义创建、配置自管理单元.

2.2　自管理单元的结构

自管理单元是 DSWC 的核心,其结构如图 4 所示. 自管理单元在原有处理模块基础上增加了一些组件,以支持事件驱动的体系结构并提供差分服务能力,包括:

1) 事件管理器(event manager)为每类客户请求创建一个请求事件队列,我们称这些请求事件队列为"分类事件队列". 事件管理器将输入事件队列中的请求事件依照类型放入分类事件队列,这里的输入事件队列(in-event queue)对应事件系统中的一个具体队列.

2) 线程池(thread pool)存储了本单元的所有处理线程. 存在一个专门的调度线程,按照一定的周期检查线程池的状态,释放过多的线程或创建新线程以保证线程池拥有合理的空闲线程个数.

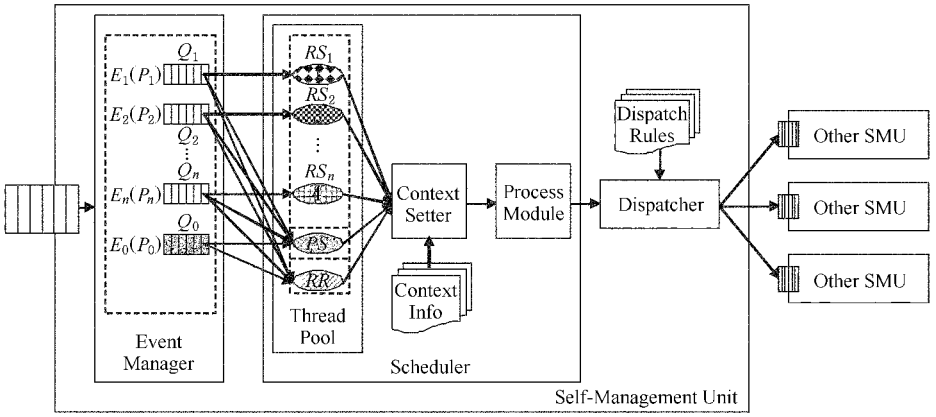


Fig. 4　The components of SMU.

图 4　自管理单元的结构

3) 分发器(dispatcher)根据分发规则(dispatch rules)与请求事件的类别将处理完的请求事件分发到处理流程的下一自管理单元. 默认的请求处理流程如图 1 所示. 分发规则的格式为:〈*request-type*,*target*〉,*request-type* 表示与 *request-event* 相对应的请求类型,*target* 代表请求处理流程中处理此类事件的下一个自管理单元.

4) 上下文设置器(context setter)按照请求类型为处理线程设置对应的上下文信息(context info),如日志级别、安全级别等属性值.

5) 调度器(scheduler)负责为分类事件队列中的事件分配处理线程. 它将线程池划分为 3 个逻辑上的子线程池:轮转调度线程池 RRTP、预留线程池 RSTP 和优先级调度线程池 PSTP. 每个子线程池采取不同的线程调度策略.

请求事件在自管理单元中的处理过程如下:1)一个请求事件 e 被放入自管理单元的输入事件队列;2)事件管理器将 e 放入分类事件队列,然后由调度器为其分配处理线程 t;3)上下文设置器设置 t 的线程上下文;4)执行处理模块,完成本模块的处理;5)分发器根据分发规则将 e 分发到下一个自管理单元,t 回到线程池继续等待处理下一个请求. 对于初始自管理单元(第 1 个自管理单元),它包含了一个监听系统特定端口来接收客户请求的监听器,接收到客户请求后将其包装成请求事件,并根据 SLA 定义设置其事件类别,然后将其分发到下一自管理单元.

自管理单元独立管理线程资源,一方面使得它能够在其内部通过预留线程、优先级调度等方式为请求提供差分服务. 另一方面,由分发器根据请求事件类型将请求事件分发到不同的自管理单元,这也使得 Web 容器能够为请求提供不同的处理流程. 最后,由于自管理单元之间通过请求事件异步连接,DSWC 容器能够在运行时动态重配一个自管理单元的差分服务配置或修改请求事件的处理流程(即 SMU 链)而不必重新启动 Web 容器. 下一节我们将详细介绍 DSWC 所提供的差分服务支持.

3 DSWC 提供的差分服务

DSWC 提供两个层次的差分服务:1)通过为请求部署不同的自管理单元链,提供基于自管理单元链的差分服务;2)由自管理单元根据其内部状态选择差分服务策略,提供基于自管理单元的差分服务.

3.1 基于自管理单元链的差分服务

自管理单元链(self-management unit chain)是一条逻辑处理链,由一组通过事件相互连接的自管理单元组成,它代表了请求的处理流程. 每条自管理单元链完成一类或几类请求的处理. 通过事件相互连接的各个自管理单元之间不再紧密耦合,可以在运行时修改自管理单元的分发规则为请求重新配置自管理单元链,实现差分服务的动态重配. 下面我们举例说明基于自管理单元链的差分服务及其动态重配.

假设 DSWC 中存在两类请求事件,它们的 *request-type* 分别为 A 和 B,其中 B 不需要检查其登录信息,而 A 则需要完整的处理流程. 因此,DSWC 为两类事件提供了图 2 所示的自管理单元链:B 的自管理单元链不包括 SMU3,而 A 的自管理单元链包含所有自管理单元. 因此,对于 SMU2,存在分发规则〈B,SMU4〉与〈A,SMU3〉,而对于 SMU3 则仅存在分发规则〈A,SMU4〉,这就为 A 和 B 提供了不同的请求处理流程. 另一方面,如果用户在 SLA 中重新定义了 B 的处理流程,要求 B 也需要登录信息检查. 那么自管理单元管理器将检测到 SLA 的变化,同时修改 SMU2 中关于事件 B 的分发规则为〈B,SMU3〉,同时给 SMU3 添加〈B,SMU4〉的分发规则,完成自管理单元链的动态重配.

3.2 基于自管理单元的自适应差分服务

自管理单元可以选择多种策略为请求提供不同质量的服务. 图 5 解释了 SLA 如何被映射为不同的差分服务策略,其中 *property* 标签被映射为上下文策略,*priority* 标签被映射为准入控制策略和线程调度策略(资源预留和优先级调度). 自管理单元将根据其内部状态(如请求事件队列长度等信息)动态地为请求选择差分服务策略. 下面将详细介绍上述

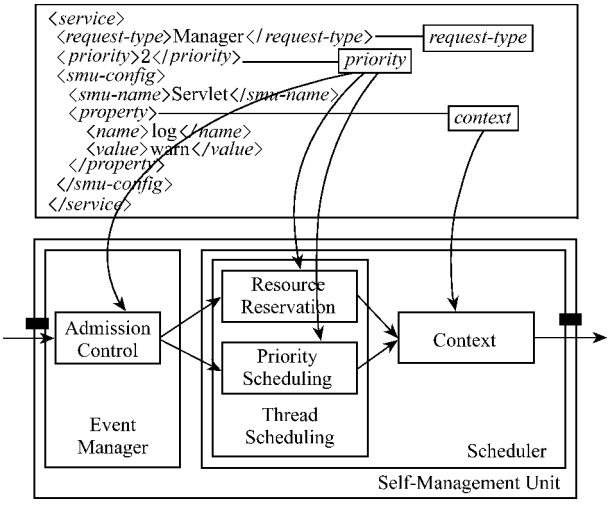


Fig. 5 Differentiated services mapping from SLA.

图 5 SLA 的差分服务映射

的差分服务策略.

为方便讨论,假设系统中存在 n 种客户的请求事件 $E_1,\cdots,E_n$,以 E_0 代表无类型请求事件,则自管理单元中包含了 $n+1$ 类事件 $E_0,E_1,\cdots,E_n$,其中事件的优先级 $E_k.priority \leqslant E_{k+1}.priority$,$E_k$ 对应分类事件队列 Q_k.

1) 上下文策略. 有些请求要求特殊的线程运行上下文,如日志级别、安全级别等. 因此,分配给请求事件 E_k 的处理线程由上下文设置器(context setter)为其设置其运行上下文 C_k,之后再完成对 E_k 的处理.

2) 准入控制策略. 事件管理器按照静态优先级 $prority$ 为事件 E_k 创建分类事件队列 Q_k. Q_k 的最大长度为 $L_{\max}(Q_k)=L_{\max}(Q_0)\times E_k.priority/E_0.priority$. 若 Q_k 中的请求事件个数达到最大长度,后续的 E_k 将被丢弃. 因此,$priority$ 值较大的请求将获得较长的分类事件队列,这使得当请求负载加大时,该类请求能承受更大负载.

3) 线程调度策略. 调度器将线程池分为 3 个子线程池:轮转调度线程池(RRTP)、优先级调度线程池(PSTP)和线程预留线程池(RSTP),其中 RSTP 的线程个数上界为 $RST_{\max}$,RRTP 和 PSTP 的线程个数下界分别为 $RRT_{\min}$ 和 $PST_{\min}$. 每个线程池中的线程采用不同的策略调度事件处理:

① 轮转调度线程. 统一对待所有分类事件,它依次处理所有分类事件队列中的分类事件.

② 优先级调度线程. 为事件处理器中的分类事件队列提供基于严格优先级的策略调度. 它每次处理具有最高优先级的分类事件队列队首的事件. 由于分类事件队列的长度 L 反映了自管理单元处理请求事件的压力,我们采用它作为优先级调度算法的优先级函数,$L(k,t)$ 表示在时刻 t 分类事件队列 k 的长度,优先级调度的主要步骤如算法 1 所示.

算法 1. 优先级调度算法.

① int $P_{\max}=0$; /* $P_{\max}$ is maximal priority value… */
② int $m=0$; /* m is queue number to be processed */
③ for (int $k=n$; $k>=0$; $k--$) {
④　if ($L(k,t)>0$) { /* $L(k,t)$ represents the length of Q_k in time t. */
⑤　int $P_{cur}=L(k,t)\times E_i.priority$; /* P_{cur} represents the value of priority function */
⑥　if ($P_{\max}<P_{cur}$) {
⑦　$P_{\max}=P_{cur}$;
⑧　$m=k$;
⑨　}
⑩ }
⑪ if ($m==0$) {
⑫　thread goes on waiting;
⑬ }
⑭ else {
⑮　thread processes the first E_m in Q_m.
⑯ }

③ 预留线程. 调度器通过为请求预留不同数量的处理线程来提供预留差分服务策略. 假设为 E_k 预留的线程队列为 TQ_k,则其最大线程数为 $TQ_{\max}(k)$,调度器根据 $E_k.priority$ 值按比例为 E_k 提供了预留线程资源,其最大值为 $RST_{\max}(k)=RST_{\max}E_k.priority/\sum_{i=1}^{n}Ei.priority$.

3.3　差分服务策略的自适应选择

比较上面几种差分服务策略,上下文策略由用户指定,无法动态修改. 准入控制策略是最有效的策略,它通过丢弃请求事件直接降低了自管理单元的负载. 对于线程资源调度的 3 类策略:统一对待所有请求事件的轮转调度没有提供差分服务;预留策略仅为一类事件 E_k 提供了线程预留,而优先级调度对 E_k 的服务介于两者之间. 因此,可以认为 4 种策略的差分服务能力关系为:轮转策略<优先级策略<预留策略<准入控制策略. 但是,准入控制策略丢弃事件会造成部分用户的请求不能得到响应,因此除非其他策略都无法满足差分服务要求时才考虑采用准入控制策略. 对于基于线程调度的 3 类策略,预留线程资源容易造成线程资源的浪费;优先级策略则每次都需要计算所有分类事件队列的优先级,占用了处理资源;而轮转调度仅需要找到一个非空队列即可. 因此,根据对自管理单元性能的影响不同,我们应该按照以下的优先级顺序选择差分服务策略:轮转策略>优先级策略>预留策略.

由上面的分析可知,在满足差分服务要求时,我们应该优先采用对性能影响较低的策略以提高性能,仅当不能满足差分服务要求时,才依次采用具有更高差分服务能力的策略.

为给出差分服务策略的选择算法,我们作出以下约定:在时刻 t,轮转调度线程数为 $RRT(t)$,优先级调度线程数为 $PST(t)$,E_k 的分类事件队列 Q_k 的长度为 $L(k,t)$,为请求事件 E_k 预留的线程数为 $RST(k,t)$. 所有处理线程完成对请求事件的处理之后都回到其所属的子线程池等待处理新的请求事件,若对应的子线程池中线程数已经到达上限值,则

将该线程并入轮转调度子线程池.这里依然使用分类事件队列 Q_k 的长度 $L(k,t)$ 作为改变差分服务策略的依据.根据以上分析,我们给出自适应的策略选择算法如算法 2.它具有以下性质:

性质 1. 基于 SMU 的差分服务没有饥饿.

RRTP 具有最小线程数 $RRT_{\min}$,这保证了始终有一定的处理线程公平地轮转调度处理所有请求的事件.除非由于负载过大导致事件被丢弃,进入 Web 容器的请求一定会被处理.

性质 2. 同类请求的顺序处理.

算法 2 总是调度分类事件队列队首的请求事件进行处理,因此若请求事件不被丢弃,同类请求事件将被顺序处理.但是对于不同分类的请求,由于其优先级不同可能造成后到达的高优先级请求被优先处理,这符合差分服务的要求.

算法 2. 自适应策略选择算法.

```
foreach (Q_k) {
  if (L(k,t) > 0.8 × L_max(Q_k)) {
  /* 当前差分服务策略已经不能满足该队列处理 */
  if (RRT(t) > RRT_min) { /* 还有轮转调度线程可以分给优先级调度 */
    从 RRTP 中分配 RRT(t) - RRT_min 的线程放入 PSTP;
  }
  else if ((RRT(t) == RRT_min) {
    /* 已经无法将轮转线程调拨成优先级调度线程 */
    /* 将一部分优先级调度线程分给该类请求作为其预留线程 */
    从 PSTP 分配 RST_max(k)/3 的线程 TQ_k 放入 RSTP(k);
  }
  else if(RST_max(k) != RST(k,t) && PST(t) != PST_min) { /* 可以为该分类增加预留线程 */
    从 PSTP 分配 min(RST(k,t), RST_max(k) - RST(k,t)) 的线程放入 RSTP(k);
  }
  else if ((RST(k,t) == RST_max(k)) { /* 该类请求预留线程已达上限 */
    采用基于准入控制的差分服务,丢弃其后续事件.
  }
}
```

4 实 验

DSWC 在中科院软件所的 Web 应用服务器产品 OnceAS[5] 中进行了原型实现.为与传统 Web 容器对比,DSWC 的实现也同时提供了对传统体系结构的支持(仅设置初始自管理单元的线程数,将后续自管理单元的最大线程数设置为 0).本小节将通过实验来验证 DSWC 的差分服务能力.

4.1 实验设计

实验环境配置如下:服务器和客户机通过 100Mbps 局域网互连,服务器为 Dell PC GX270,2GB 内存,CPU Intel 3.04GHz;客户机为 Dell PC GX270,512MB 内存,CPU Intel 2.8GHz;两台机器运行的都是 Windows 2000 Server 操作系统,采用两种设置的 OnceAS 作为对比部署平台,Pet Store[9] 作为部署于服务器的应用.两种设置的 OnceAS 平台为 OnceAS-Old 与 OnceAS-DSWC,两者的 Web 容器处理模块组成如图 2 所示,前者为传统 Web 容器的体系结构实现,后者采用 DSWC 实现.

```
〈SLA〉
  〈classfier〉org.sla.classfier〈/classfier〉
    〈service〉
      〈request-type〉Manager〈/request-type〉
      〈priority〉2〈/priority〉
      〈smu-config〉
        〈smu-name〉Servlet〈/smu-name〉
        〈property〉
          〈name〉log〈/name〉
          〈value〉warn〈/value〉
        〈/property〉
      〈/smu-config〉
    〈/service〉
    〈service〉
      〈request-type〉Worker〈/request-type〉
      〈priority〉1〈/priority〉
      〈smu-config〉
        〈smu-name〉Servlet〈/smu-name〉
        〈property〉
          〈name〉log〈/name〉
          〈value〉info〈/value〉
        〈/property〉
      〈/smu-config〉
      〈smu-config〉
        〈smu-name〉URL Mapping〈/smu-name〉
        〈target〉Servlet〈/target〉
      〈/smu-config〉
    〈/service〉
〈/SLA〉
```

Fig. 6 SLA of pet store.

图 6 Pet Store 的 SLA

实验模拟两类用户 Manager 和 Worker,他们通

过访问 Product. jsp 获取产品信息. 两类用户的 SLA 配置如图 6 所示,Web 容器每次从 session 中取得 *request-type* 参数值来区分两类用户.

Worker 用户并不需要进行登录验证,因此类型为 Worker 的事件被直接分发到 Servlet 自管理单元(SMU4),而类型为 Manager 的事件需要经过所有的自管理单元处理,这使得 Worker 和 Manager 具有不同的请求处理流程(见表 1). 同时,自管理单元 Servlet(SMU4)还为两类请求事件提供了基于自管理单元的差分服务,其中 $RRT_{\min}=10$, $PST_{\min}=5$, 其他配置如表 1 所示:

Table 1　The Configuration for Differentiated Service of Servlet Module

表 1　Servlet 处理模块差分服务配置

Request-Type		Manager	Worker
SMU chain		SMU1, SMU2, SMU3, SMU4, SMU5	SMU1, SMU2, SMU4, SMU5
Servlet Module	*Priority*	2	1
	$L_{\max}(Q_k)$	$L_{\max}(Q_2)=100$	$L_{\max}(Q_1)=50$
	$RST_{\max}(k)$	$RST_{\max}(2)=10$	$RST_{\max}(1)=5$
	Context	*log* = *warn*	*log* = *info*

将 OnceAS-Old 的 SMU1 最大线程个数设为 75,其他自管理单元设为 0,则 OnceAS-Old 可以被看做采用传统体系结构实现的 Web 容器. 将 OnceAS-DSWC 中的 SMU4(Servlet 单元)单元的最大线程个数设为 30,其他单元取默认值 5,则 OnceAS-DSWC 也同样具有 $30+9\times5=75$ 个处理线程,这样使得两者具有相同的处理资源. 测试进程开始时,模拟两类用户各 100 个并发访问 Web 容器,以后每 4s 增加 2 个用户,持续访问 10min 后采用相反的趋势减少用户,继续持续 10min 后停止,运行得到的实验结果如图 7 所示.

从图 7(a)我们可以看出,OnceAS-Old 对于两类用户不加区分,它们有着几乎相同的响应时间曲线. 而在 OnceAS-DSWC 中,Worker 用户的响应时间要比 OnceAS-Old 中的 Worker 用户高 29% 左右,但 Manager 用户的响应时间比 OnceAS-Old 中的对应用户低 23%,这证实了 OnceAS-DSWC 能够区分两类用户并为其提供差分服务. 造成这种现象的原因是 OnceAS-DSWC 为 Manager 用户提供了更多的处理线程预留,同时优先级调度线程也优先调度 Manager 的请求事件进行处理,因此在 DSWC 中 Manager 用户获得了更好的性能指标.

由图 7(b)可知,在 Servlet 自管理单元中,开始时仅有轮转调度线程处理请求事件,随着负载的增加,轮转调度已经不能满足差分服务的需要时,部分轮转调度线程进入优先级调度线程池和资源预留线程池以优先处理类型为 Manager 的请求事件. 反之,当 Web 容器的负载下降时,优先级调度线程和资源预留线程数也随之减少,进入轮转调度线程池. 这证实了自管理单元能够根据其内部状态自主选择合适的差分服务策略.

从图 7(c)可以看出,在 SMU3 中,没有 Worker 的请求事件被处理,这证实了 OnceAS-DSWC 为来自 Worker 的请求提供了与 Manager 不同的处理流

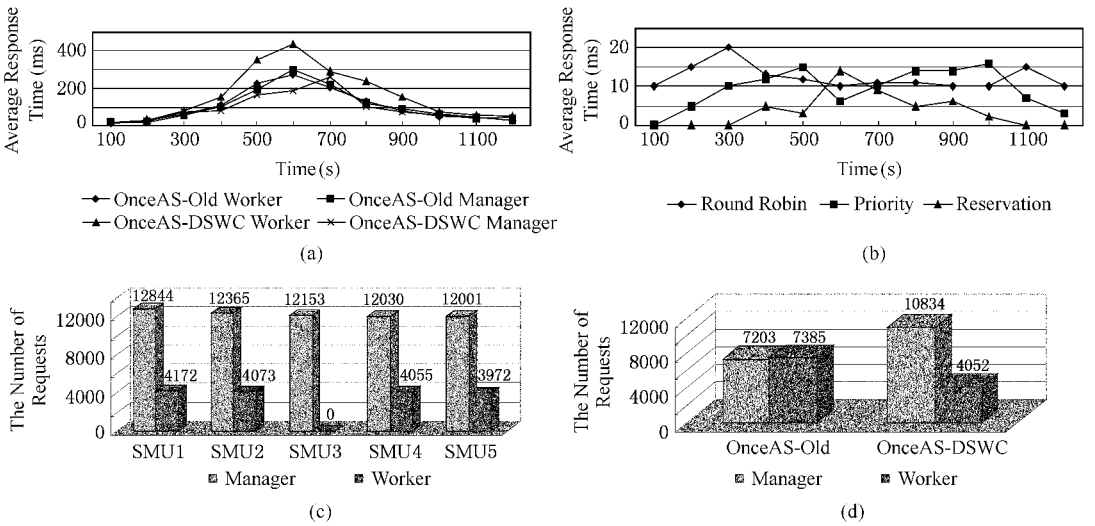


Fig. 7　Results of Experiment. (a) Average response time; (b) The number of threads of SMU4; (c) The number of completed request events in SMUs; and (d) The number of completed requests.

图 7　实验结果.(a) 平均响应时间;(b) SMU4 线程个数;(c) 各 SMU 完成请求事件数;(d) 完成请求数

程.图7(c)中的不同SMU完成的请求个数不同是由于部分请求事件在自管理单元达到最大负载时被准入控制策略所丢弃.

由图7(d)可以看出,OnceAS-Old对两类请求不加区分,完成的请求个数大致相当.而OnceAS-DSWC为两类请求提供了差分服务,因此完成的Manager请求个数是Worker请求的267%,这个比值大于两类请求的优先级之比2/1=2.这是因为处理线程处理请求时,轮转调度处理线程平等对待两类请求,基于资源预留的处理线程数之比为2:1,而优先级调度线程总是优先调度Manager的请求,这使得总是有超过Worker请求2倍的Manager请求被处理.

根据图6的差分服务定义,Manager用户需要记录较高级别(warn)的日志,而Worker用户仅需记录最低级别(info)的日志.我们将当前运行结果(OnceAS-DSWC(context))与未设置上下文差分服务的运行结果(OnceAS-DSWC)进行对比,结果如表2所示.可以看出,通过设置不同请求的上下文,DSWC能够仅记录有效的日志信息,其总量仅是没有提供上下文差分策略时的51%,这使得DSWC记录更有效日志的同时也减轻了其负载,提高了性能.

Table 2 The Number of Log
表2 日志条数

request-type	OnceAS-DSWC		OnceAS-DSWC(context)	
	info	warn	warn	info
Manager	23483	13088	24889	13507
Worker	15301	59733	18521	0

5 相关工作

Tomcat[4]和Jetty[10]是目前被大量采用的Web容器,它们均采用传统Web容器体系结构,对所有请求不加区分,从惟一的线程池中取得一个处理线程并按照固定的处理流程处理到来的请求.因此他们无法为不同的请求提供不同的处理流程和差分服务.OnceAS的原有Web容器采用类似的体系结构,同样存在以上问题.

Abdelzaher等人使用冗余以及服务资源质量降级的方法实现差分服务[11],其基本思想是提供不同质量的服务实例(如不同分辨率的相同图片)为不同的请求服务,但是这种方法只适用于面向静态资源的Web服务器,无法满足以动态请求为主的Web容器的要求.同时,部署多个应用的副本也浪费了一定的资源.

Kanodia和Chen等人采用准入控制策略为Web服务器提供差分服务[12-13],而Eggert和Bhatti等人采用基于严格优先级的调度算法为不同请求提供不同的准入控制策略以提供差分服务[7,14].但是,以上工作仅仅简单地限制低优先级的请求进入服务器,没有为通过准入控制的请求提供有差别的处理流程或不同的处理线程资源.而本文的DSWC在更细的粒度级别(处理模块)提供了差分服务支持,同时也为请求提供了不同的处理流程.

Zhang等人借用控制理论的方法设计了自适应的Web服务器,通过检测系统的某些状态动态地调整服务器参数[15],然而他们的目的是设计自适应的服务器,没有考虑差分服务的支持.DSWC提供差分服务支持的同时也支持差分服务策略的自适应选择.

Keller等人基于SLA提供了对Web服务器的差分服务支持[8,16],他们的工作重点是管理SLA的生命周期,而本文的工作从服务器的体系结构基础上提供了对服务差分的支持.近年来的一些研究者通过事件驱动的中间件体系结构设计提高服务器并发能力、降低系统设计耦合度[8,17].其中SEDA[8]提供了自适应的资源调整,大大提高了Web服务器的并发能力,但它并没有给出具体的差分服务解决方案.李振东提出了一个提供多类服务的多服务器Web平台的服务质量确保体系,但是其仅能应用于静态的Web服务器[18].

综上所述,传统Web容器因其体系结构的限制仍缺乏对差分服务的支持.而一些对Web服务器与Web容器的研究也仅通过资源降级、准入控制以及优先级调度等方式为请求提供差分服务,很少有研究考虑设计支持差分服务的服务器体系结构.本文提出了一个基于自管理单元支持差分服务的Web容器DSWC,它能够根据用户定义的SLA为请求提供基于自管理单元链与基于自管理单元两个层次的差分服务.同时,基于自管理单元的差分服务还能够根据自管理单元状态自适应的选择差分服务策略,为不同请求提供差分服务.

6 结束语

Web容器是当前运行Web应用的主要平台.用户角色、业务逻辑以及服务付费的差异性要求Web容器能够提供对不同请求的差分服务.本文提

出了一个基于自管理单元支持差分服务的 Web 容器 DSWC,并在 OnceAS 中实现了原型系统,并且通过实验验证了它的有效性.

进一步的工作主要包括:1)在集群 Web 容器环境中应用上述研究成果;2)与 EJB 容器、Web Services 引擎的协同工作.

参 考 文 献

[1] Bill Shannon. JavaTM 2 Platform Enterprise Edition Specification v1.4 [G]. Santa Clara, CA, USA: Sun Microsystems Inc, 2003

[2] Xiaobo Zhou, Yu Cai, Ganesh K Godavari. An adaptive process allocation strategy for proportional responsiveness differentiation on Web servers [C]. In: Proc of IEEE Int'l Conf on Web Services. Los Alamitos, CA: IEEE Computer Society Press, 2004. 142-149

[3] S Blake, D Black, M Carlson. An architecture for differentiated services [S]. RFC 2475. 1998. http://www.ietf.org/rfc/rfc2475.txt?number=2475

[4] Jakarta Tomcat Servlet Container [OL]. http://jakarta.apache.org/tomcat, 2006

[5] Huang Tao, Chen Ningjiang, Wei Jun, *et al*. OnceAS/Q: A QoS-enabled Web application server [J]. Journal of Software, 2004, 15(12): 1787-1799 (in Chinese)
(黄涛, 陈宁江, 魏峻, 等. OnceAS/Q:一个面向 QoS 的 Web 应用服务器[J]. 软件学报, 2004, 15(12): 1787-1799)

[6] K Alexander, L Heiko. The WSLA framework-specifying and monitoring service level agreements for Web services [R]. International Business Machine Corporation, Tech Rep: RC22456, 2002

[7] N Bhatti, R Friedrich. Web server support for tiered services [J]. IEEE Network, 1999, 13(5): 64-71

[8] M Welsh, D Culler, E Brewer. SEDA: An architecture for well conditioned scalable Internet service. The 18th ACM Symp on Operating Systems Principles, Banff, Canada, 2001

[9] The Java Pet Store [OL]. http://java.sun.com/developer/releases/petstore/, 2005

[10] Jetty Web Server [OL]. http://jetty.mortbay.org, 2006

[11] T Abdelzaher, K G Shin, N Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach [J]. IEEE Trans on Parallel and Distributed Systems, 2002, 13(1): 80-96

[12] V Kanodia, E Knightly. Ensuring latency targets in multiclass Web servers [J]. IEEE Trans on Parallel and Distributed Systems, 2002, 3(10): 12-14

[13] H Chen, P Mohapatra. Session-based overload control in QoS-aware Web servers [C]. In: Proc of IEEE INFOCOM '02. Los Alamitos, CA: IEEE Computer Society Press, 2002. 516-524

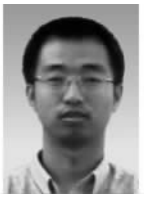
[14] L Eggert, J Heidemann. Application-level differentiated services for Web servers [J]. World Wide Web Journal, 1999, 2(3): 133-142

[15] R Zhang, C Lu, T Abdelzaher, *et al*. ControlWare: A middleware architecture for feedback control of software performance [C]. In: Proc of the 22nd Int'l Conf on Distributed Computing Systems. Los Alamitos: IEEE Computer Society Press, 2002. 301-310

[16] D Asit, L Heiko, G Pacifici. Web Services Differentiation with Service Level Agreements [G]. New York: International Business Machine Corporation, 2003

[17] A Chankhunthod, P Danzig, C Neerdaels, *et al*. A hierarchical Internet object cache [C]. USENIX Technical Conf, San Diego, 1996

[18] Li Zhendong, Xie Li. Research on ensuring QoS and its admission control in Web servers [J]. Journal of Computer Research and Development, 2005, 42(4): 662-668 (in Chinese)
(李振东, 谢立. Web 服务器群的 QoS 确保及其接纳控制研究[J]. 计算机研究与发展, 2005, 42(4): 662-668)



Li Yang, born in 1979. Ph. D. candidate in the Institute of Software, the Chinese Academy of Sciences. His main research interests include middleware and distributed computing.
李　洋,1979 年生,博士研究生,主要研究方向为中间件和分布式计算.

Chen Ningjiang, born in 1975. Associate professor of the College of Computer, Electronics and Information, Guangxi University. His main research interests include software engineering, middleware and distributed computing.
陈宁江,1975 年生,副教授,主要研究方向为软件工程、中间件、网络分布式计算.

Jin Beihong, born in 1967. Professor in the Institute of Software, the Chinese Academy of Sciences. Her main research interests include distributed computing and software engineering.
金蓓弘,1967 年生,研究员,主要研究方向为分布式计算、软件工程技术.

Zuo Lin, born in 1975. Ph. D. candidate in the Institute of Software, the Chinese Academy of Sciences. His main research interests include software engineering, fault-tolerant middleware and distributed computing.
左　林,1975 年生,博士研究生,主要研究方向为软件工程、容错中间件和分布式计算.

Huang Tao, born in 1965. Professor and Ph. D. supervisor in the Institute of Software, the Chinese Academy of Sciences. Senior member of China Computer Federation. His main research interests include software engineering and distributed computing.

黄 涛,1965 年生,研究员,博士生导师,中国计算机学会高级会员,主要研究方向为软件工程、分布式计算.

Research background

Web container adopting“ best-effort ”service mode does not satisfy complex business requirement to provide differentiated services for requests from different clients according to role or payment *etc*. Some approaches such as admission control and priority scheduling have been applied to the solution of the above problem, but they do not consider differentiated services in finer granularity so that they can only provide monotone strategy, and do not adapt the changing environment to change the strategies of differentiated services. In this paper, we provide DSWC, a differentiated service Web container based on self-management unit (SMU). In DSWC, process modules are encapsulated into SMUs, which are connected to compose a SMU chain. DSWC also provides SMU-based and SMU chain-based differentiated services for requests according to SLA (service level agreement). The experiments of prototype show DSWC can effectively provide differentiated services according to SLA for different type of requests. This work is supported by the National Natural Science Foundation of China under grant No. 60573126; the National Basic Research Program of China (973) under grant No. 2002CB312005; the National High-Tech R&D Plan of China (863) under grant No. 2006AA01Z19B, No. 2006AA01Z180; and the National Key Technology R&D Program of China under grant No. 2006BAH02A01.