一种高性能面向交互式系统管理的 HTTP/CGI SERVER 设计方法的研究*)

The Research of A High Performance HTTP/CGI SERVER for Interactive System Management

陈 健 陈俊良 徐永森

(南京大学计算机软件新技术国家重点实验室 南京 210093)(南京大学网络中心 南京 210093)

Abstract Web-based management systems are increasingly replacing traditional terminal-based and Xwindows-based systems, Producing such a system seems to require a tremendous amount of laborious low-level coding due to the primitive nature of CGI programming, it also suffes from the overhead of CGI processing, perceived latency of Web respond and lack of security, we present ideas for the whole new HTTP/CGI Server to solve these problems and give a high performance to the management system.

Keywords Web, CGI, HTTP, Session, Management

1. 引言

基于 Web 方式的管理系统由于其易用性和友好的界面，逐渐取代了传统的字符方式的管理系统，其具备的通过网络远程管理及与平台无关的特点，在与基于视窗界面的管理系统的比较中，也占据了明显的优势；与基于 SNMP 的管理系统相比较，由于 SNMP 需要管理对象具有一个 MIB 库和一个解释 SNMP 字符串的代理，这样就极大地限制了它所能管理对象的范围，在这一方面，Web 方式的管理系统体现了很好的包容性。

目前有许多软件实现了针对 UNIX 系统、服务和安全等各个方面的基于 Web 的图形化管理，但由于其功能的单一和缺乏对多用户的支持，不能被称为完善的管理系统，采用体系化方法设计的管理系统中比较著名的有 WEBMIN[1]，它实现了多用户、模块化的设计方法，但上述这些软件共同的特点就是都使用现有的 Web Server 再通过 CGI 调用来实现，这样的实现方法带来的问题有：

1)受限于现有 Web Server 传递给 CGI 脚本的环境变量范围，不能实现基于事务处理的 CGI 调用；

2)管理系统是一种面向用户的服务，对用户需要有严格的审核，对系统资源需要分级的处理，现有 Web Server 缺乏对这一方面的支持；

3)管理系统是一种交互式的服务，由于 CGI 编程自身比较单纯的交互式处理能力，使得完成基于 Web 的管理系统需要大量繁琐的底层代码的编写；

4)现有 Web Server 对 CGI 的调用方式及对 CGI 计算产生数据的传输方式在处理大用户量的访问时会严重影响系统的负荷并产生 Web 响应的延迟；

5)现有 Web Server 对传输数据安全性的处理不具备面向用户的功能。

由于基于 Web 方式的管理系统是一个完全架构于 CGI 调用之上的交互式服务，session 的处理方式，系统数据的安全性，用户信息的管理都直接和 CGI 相关，如何实现合理并且高效的 CGI 处理，将直接影响到整个管理系统的效率和维护的代价。本文我们提出了设计一种架构于 CGI 之上的基于 session 处理方式的高性能 Web 服务器用于取代传统的 HTTP/CGI 服务器的想法，我们将它命名为 IWMS(Integrated Web-based Management System)，该系统目前已经实现并且用于完成江苏省高技术研究项目“基于 Web 的 Linux 可视化集成管理系统”。

2. 动机

在以下几节我们将分析目前实现基于 Web 的应用程序的常用方法的优缺点，从而说明为什么它们不适合应用在基于 Web 的管理系统的实现中。

2.1 通用网关接口 CGI[4]

CGI 事实上是第一个 Web 应用程序的标准接口，CGI/1.1 的第一个版本早在 1997 年就已推出，并在 NCSA 服务器上实现，它的特点如下：

1)简单性，容易理解；

2)语言独立性，CGI 应用程序可以用几乎所有语言来编写；

3)进程分离，CGI 应用程序运行在与 Web 服务器不同的进程中，因而其运行的故障不会影响到服务器的运行并且它也不能访问服务器进程的私有空间；

4)结构独立，CGI 应用程序的结构不依赖于 Web 服务器的结构。

CGI 同时也有很多不足：

1)从管理系统的角度来看，CGI 是一个无状态的协议，对 CGI 的执行仅仅持续到服务器向客户端输出页面为止，这意味着如果要将 CGI 应用到 session 的处理过程中，我们必须将一个连续的计算过程分解为许多小的计算单位，每一个计算单位处理一次用户的交互过程并且调用相应的 CGI 脚本，这是一个非常繁琐的过程；

2)为了保持在 session 处理过程中本地状态的持续性，在 CGI 调用之间必须频繁地存储和恢复该状态，对于简单的交互式应用来说，这种做法是可行的，但对于一个大型的管理系统来说，显然是不充分的；

*)本课题得到江苏省高技术研究项目基金资助(编号:BG2001014)。陈 健 研究生，主要研究方向为计算机网络管理和网络安全研究。陈俊良 教授，主要研究方向为计算机网络。徐永森 教授，主要研究方向为软件工程和软件理论。

3)CGI 在 session 的处理过程中只是充当一个简单的响应者的角色,它不能进入一个请求过程的其它阶段,如验证和日志阶段;

4)从性能的角度来看,每当客户端发送来一个请求,服务端都将产生一个新的进程来调用 CGI 应用程序处理该请求,当客户端结束请求时,又立刻释放该进程,这样的运行方式效率是很低的。

2.2 服务器 API

为了解决 CGI 的性能问题,很多厂商开发了自己服务器的 API,最著名的有 Netscape 的 NSAPI 和 Microsoft 的 ISAPI,Apache 服务器也有它自己的 API,使用 API 的优点如下:

1)应用程序直接连接到服务器的 API,从而改善了启动和初始化 CGI 的代价,提高了运行的性能;

2)通过使用 API,应用程序可以连接到一个请求过程的不同阶段,从而提供接入控制、日志访问等其他功能。

虽然它解决了 CGI 的性能问题,但却带来了其它几个缺陷:

1)复杂性,实现和维护使用 API 的应用程序代价很高;

2)语言依赖性,必须要使用 API 指定的语言来编写应用程序;

3)进程共享,应用程序并不产生它自己的进程,而是在 Web 服务进程内部运行,这样一个 API 应用程序的错误将会导致整个 Server 运行的中断,并且也会带来安全性的问题;

4)产品依赖,每个公司的 API 实现之间不能互相兼容。

2.3 FASTCGI

FASTCGI 基本上结合了 CGI 和 API 的一些优点,它使用一个独立的进程来持续处理用户的多个请求,并且对实现的语言没有限制,但其带来的问题也是明显的,对于一个管理系统来说,必然会有大量的 CGI 应用程序存在,如果都让它们长驻运行,必然会影响 Web 服务器的性能,而如果将所有的管理功能集中到一个应用程序中来,则会带来请求处理的瓶颈问题。

2.4 RUNTIME 系统

RUNTIME 系统提出了基于 session 方式来处理交互式服务的想法,它将 CGI 脚本分解为两个组成部分:一个连接器和一个 session 线程。连接器是一个小型的、暂态的 CGI 脚本用于在客户端和 session 线程之间传递数据。Session 线程是一个长驻服务端的进程,它代替 CGI 脚本的功能用来提供一个涉及到多个用户交互的长时间的计算处理过程,同时提供了一个控制器用来处理 CGI 脚本之间的并发控制,除此以外,session 线程并不将计算结果直接输出给客户端,而是将该结果写到文件中,将该文件的一个链接返回给用户,用这种简单的方式保持本地状态的持续性。

RUNTIME 系统的问题如下:

1)对于每一个用户的 session 都将启动两个进程,一个连接器,一个 session 线程,随着访问用户数的增加,进程数目将加倍;

2)频繁的磁盘读写,响应将写入文件,用户通过自动刷新方法反复地读取文件;

3)易受攻击,通过文件的 URL 就已经足够鉴别一个 session。

针对以上这些实现方法的缺陷,我们提出了设计一个全新的系统 IWMS 来代替现有 HTTP/CGI 的实现方法。

3. IWMS 系统的组成和运行机制

3.1 IWMS 系统的组件

Session 代理 接受 User Agent 的请求,调用用户信息库中的数据鉴别用户的身份并产生相应的 session thread 来处理该请求;

用户信息库 这是一个小型的用户 DBM 数据库,每个用户的信息包括:用户 ID、用户密码、session ID、安全级别、最后一次访问时间、最后一次访问状态;

请求分析器 对请求的信息进行解析,分解出查询的内容和传递的参数;

协调控制器 利用分解的数据查询 CACHE 或是将数据传递给 CGI PLUGIN 并监控 CGI PLUGIN 的运行;

并发控制器 设置运行状态检测,确保 CGI PLUGIN 的运行满足设定的状态安全;

重载器 实现 CGI 对象的绑定,以保持和传统 CGI 编程的兼容性;

模块库 实现系统管理的主体,为存储在磁盘上的程序段,其编写方法保持和传统 CGI 的一致;

CGI PLUGIN 将通过重载器调用的模块直接在 session thread 内部运行的模式;

CACHE 在 session thread 内定义的缓存空间,用于保存从 CGI PLUGIN 输出的数据。

除此之外,在 IWMS 系统中还包括一个 session thread 管理器,用于垃圾收集、session 超时和释放已结束的 session thread;一个服务发现代理,用于实现一个集成的系统管理环境;一个日志管理器,用于记录用户访问的信息和系统的状态以及在运行过程中产生的错误。

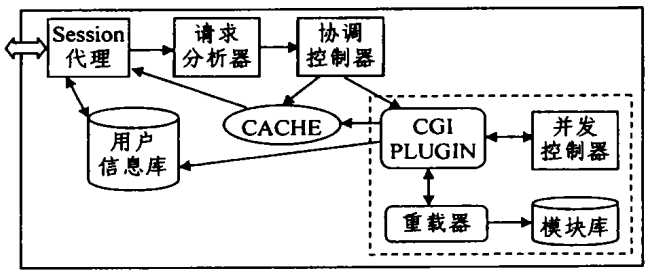

图 1 IWMS 系统

3.2 运行机制及主要特点

在这一节我们将描述 IWMS 系统的动态行为,首先说明一个 session thread 的完整运行过程,接着阐述该系统的主要特点。

3.2.1 session thread 的状态及其运行过程 一个 session thread 在运行过程中分为五个状态:启动、运行、等待、输出和结束。

当 session 代理接受到用户的请求并通过用户信息库的鉴别通过后,session 代理将首先检查其在用户信息库中保留的最后运行状态,如其存在并且时间有效(系统提供一个可调整的时间参数),将直接将该用户恢复到其上次运行的状态并启动 session thread,否则 session 代理将启动一个新的 session thread 来处理该用户的请求,此时 session thread 的状态从启动状态进入到运行状态。

在运行状态中,session thread 首先调用请求分析器完成对用户请求信息的分析,接着将分析结果传递给协调控制器,

协调控制器首先查看该 session 的 CHCHE 中是否已有该请求的结果(CACHE 只保留最新请求的结果),如有,则直接将结果返回给用户,如没有,则将请求数据传递给 CGI PLUGIN,CGI PLUGIN 通过重载器调用存储在磁盘上的模块库中相应的服务,直接嵌入到 session thread 中运行,如该嵌入模块在运行过程中进入安全状态区域并且触发了检测点,则并发控制器将暂停它的运行,此时 session thread 从运行状态进入到等待状态。

当并发控制器允许其进入安全状态之后,session thread 将从等待状态重新返回运行状态,并计算出结果,此时 session thread 从运行状态转变为输出状态。

在输出状态中,session thread 将结果页面首先存储在 CACHE 中,而不是直接将结果输出给用户,这样做的理由我们将在后面说明,接着将 CACHE 中的数据输出给用户并且同时存储在用户信息库中,此时 session thread 从输出状态再返回到等待状态,等待用户发送下一次请求使其再重返运行状态,最后当用户结束这次的访问或者该 session thread 超时,该 thread 将从运行状态进入结束状态。

3.2.2 主要特点

1)重载器的使用。使用重载器在 session thread 中虚拟出 CGI 执行的环境,通过对象绑定重定向标准的输入输出,通过语义修改重新定义常用的函数,使 CGI 脚本直接在 session thread 内部运行,就如同是 thread 内部的一部分,这样做的好处是避免了系统反复启动和初始化 CGI 脚本的代价,同时在单独的 thread 中运行 CGI 脚本,也避免了象 API 方法那样因为 CGI 脚本的故障而影响整个 Server 的运行。

2)CGI CACHE 的使用。在 RFC2068[5]中提出了 persistent connection 的标准,这一标准用于在一个持续的连接中传递多个静态对象而不用对每个对象启动一个连接,HTTP 协议使用三种方法来判断一个对象的结束:使用 content-length 响应标题,使用 content-type 响应标题中定义的边界分割符或者是利用 TCP 连接的终止;但对于 CGI 脚本的输出,由于并不知道结果数据的长度,因此不能使用 content-length 标题,同时由于也不能了解输出数据的类型,因此也不能使用 content-type 标题,目前的处理方式都是通过简单的将数据直接输出给客户端并且发送 TCP 连接终止信号结束连接,对于以静态页面为主的应用服务来说,这种简单的处理方式影响并不是很大,但对于完全通过交互方式提供服务的管理系统来说,这种方式使得 session 的处理方式失去了真正的意义,在本系统中通过采用 CGI CACHE 的方式解决了该问题,session thread 的计算结果并不直接发送给用户,而是先存储在 CGI 缓存中,计算出 content-length 之后再将结果转发给用户,由于采用了高效的存储结构,存储转发的代价很小。

3)并发控制器的使用。每个 session thread 都有内建的检测点设置在临界代码之前,当执行到该检测点时,thread 必须向并发控制器发出询问以获得允许继续运行,并发控制器根据设定的安全需要限制 thread 的运行并且只允许那些不会破坏安全要求的 thread 继续运行。

在目前基于 Web 的管理系统中大都是采用了对文件锁定和解锁的方式来处理并发控制,这种方式安全性差而且只能处理简单的并发控制,在本系统中我们采用了 Claus Brabrand 提出的并发控制概念[3],将并发控制器分为三个组成部分:控制逻辑单元,检测点事件队列和超时队列,其结构如图 2 所示。

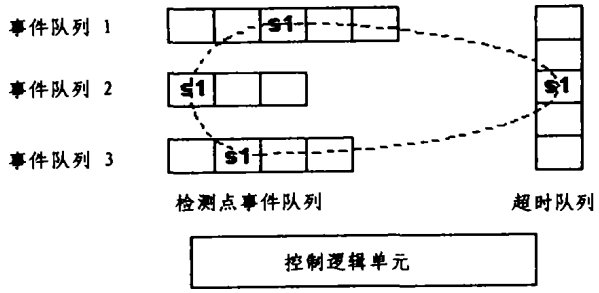

图 2 并发控制结构

控制逻辑单元:是真正用来表达安全需求的组件,它控制事件是否被激活并且判断 session thread 是否可以在检测点继续运行;

检测点事件队列:对每一个可能的检测点事件都有对应的一个事件队列,当一个 session thread 到达检测点的时候,它将把它的 session ID 加入到对应的事件队列中等待控制逻辑单元的处理,一个 session thread 可以同时进入多个事件队列;

超时队列:当在指定的时间内该 session ID 还未被允许执行,控制逻辑单元将把该 ID 放入超时队列并且返回给 session thread 一个超时信息。

4)session thread 的超时机制。与大多数基于 Web 的管理系统通过在服务端与客户端之间发送"你好-再见"这样的信息包来保持 session 的持续性和确定超时时间不同,IWMS 系统通过将用户最近的连接状态放入用户信息库,利用 session 管理器定期检查信息库来确定用户 session thread 的超时,这样的方式极大地提高了系统的吞吐量和服务的效率。

5)模块编写与 CGI 完全兼容。与 API 的实现方式不同,本系统通过重载机制实现 CGI 的运行,所以在编写模块时可以完全按照传统的 CGI 编写方法,减轻了用户的学习曲线。

6)服务发现代理。基于 UDP 协议的内置的服务发现代理可以用来组建一个集成化的管理平台,通过模块库的互连,实现在一致的界面下对指定网络环境下系统的统一管理。

4. 性能分析

我们从两个方面对 IWMS 系统进行了初步的性能分析:响应时间和吞吐量。

测试环境:服务段为 100M 全双工以太网接口的 HP NetServer LC2000 服务器;客户端为连接到 10M 共享 Hub 上的 PIII800PC 机,为了测试在一定负载情况下服务的响应和吞吐量,服务器为一个平均有 50 人访问的 FTP Server。

1)响应时间

我们在服务端编写了一个简单的 CGI 应用程序,它首先调用用户数据库进行用户的验证,接着调用磁盘上的数据库并产生 10K 字节大小的输出,客户段的程序通过发送连续的请求来接受数据并将数据抛弃,使用 tcpdump 工具统计响应时间,其统计结果如表 1 所示。

表 1 吞吐量

静态文件	11.29ms/per kbyte
IWMS	16.16ms/per kbyte
Apache/FastCGI	16.25ms/per kbyte
Apache/API	17.54ms/per kbyte
Apache/CGI	34.21ms/per kbyte

从表中可以看出，IWMS 系统的响应时间与 FastCGI 大致相当，稍好于 API 的时间，是 CGI 方法响应时间的两倍，这主要是因为 IWMS 系统对 CGI 采用了持续连接的处理方法，并且对 CGI 采用嵌入运行的模式，考虑到由于 FastCGI 不支持对 CGI 的持续连接处理，在采用 TCP 慢启动[9]的网络环境中并且应用程序产生大量的小文件的时候，IWMS 系统的表现应当会优于 FastCGI 的表现。

2)吞吐量

为了测试吞吐量，我们将客户端程序直接放在服务器上运行，这样可以不用考虑网络延迟的影响，客户端程序在运行后会产生 30 个进程以模拟用户的连接访问服务器，测试结果如表 2 所示。

表 2 吞吐量

IWMS	9.2ms/per request	109 requests/per second
Apache/Fast CGI	8.5ms/per request	118 requests/per second
Apache/API	15ms/per request	67requests/per second
Apache/CGI	35ms/per request	29 reques/per second

从表 2 可以看出，IWMS 的表现要好于 API 的方式，并且是 CGI 方式的四倍，但要稍差于 FastCGI 的吞吐量，这主要是因为 IWMS 中加入了对 session 的支持，对每个用户的连接都要产生一个 session，并且不是采用一个持续运行的 CGI 程序，而是通过 CGI PLUGIN 在每个 session 内部运行，而 FastCGI 不支持 session 的处理方法，这方面的开销节省了。

5. 系统安全性的讨论

对于管理系统来说，网络传输的数据安全性是非常重要的，目前基于 Web 的管理系统都是通过 SSL 或 TLS 的方法实现 HTTP 数据的加密，IWMS 系统也采用该方法来实现数据安全，该方法的优点是实现简单，缺点是应用层协议需要提供额外的端口来支持 TLS，对于加密的信息和普通信息采用两种不同的服务并且不能实现面向用户的加密，Rohit khare[6]提出了通过使用 http/1.1 的 Upgrade 机制来在一个已经存在的 TCP 连接上发起 TLS，从而使加密和普通的 HTTP 通信共享一个端口的想法，该想法的另一个好处就是使得提供面向用户、面向需求的加密服务变为可能，用户不需要在连接的开始就协商加密方法，而可以在连接的任何时间提出申请，经过双方的协商进行数据加密，服务端也可以通过发送强制加密的 Upgrade 标题来拒绝协商，这种灵活的加密方法可以说是以后基于 Web 方式数据加密的一个方向。

结论 本文通过实现一个面向交互式系统管理的 HTTP/CGI Server 来解决目前基于 Web 的管理系统所面临的一些问题，实践证明该系统在处理交互式的管理服务时具有较好的运行性能，它的一些想法如嵌入式 CGI 的运行，session thread 的处理方式，CGI CACHE 的使用同样可以应用在其它的基于 Web 的交互式服务中。

参考文献

1 Jamie Cameron. http://www.webmin.com/webmin/

2 Open Market. Inc. FastCGI: A high-performance web server interface. Technical White Paper. April 1996

3 Brabrand C, et al. Aruntime system for interactive Web services. Computer Networks, 1999, 31: 1391～1401

4 Coar K. http://cgi-spec.golux.com

5 Fielding R, Frystyk H. rfc2068, Jan. 1997

6 Khare R, Lawrence S. Upgrading to TLS Within HTTP/1.1, Network Working Group Internet Draft January 5, 2000

7 Brabrand C. Synthesizing. Safety controllers for interactive web services: [M. Sc. thesis]. Dec. 1998

8 Sandholm A, Schwartzbach M L. Distributed safety controllers for web services. FASE98

9 Jacobsen V. Congestion Avoidance and Control. In: Proc. SIGCOMM'88 Symposium on Communications Architectures and Protocols, Standford, CA, Aug. 1988. 314～329

(上接第 67 页)

4 Xiao X, Hannan A, Bailey B, Ni L. Traffic engineering with MPLS in the Internet. IEEE Network Magazine, March 2000

5 Ashwood-Smith P, et al. Improving Topology Database Accuracy With LSP Feedback. draft-ietf-mpls-te-feed-03.txt, work in progress, Nov. 2001

6 Crawley E, et al. A Framework for QoS-based Routing in the Internet. RFC2386, Aug. 1998

7 Brittain P, Farrel A. MPLS Traffic Engineering: A Choice of Signaling Protocols. Data Connection Limited, Jan. 2000

8 Awduche D O. MPLS and traffic engineering in IP networks. IEEE Communication Magazine, December 1999. 42～47

9 Sharma V, et al. Framework for MPLS-based Recovery. draft-ietf-mpls-recovery--frmwrk-03.txt, work in progress, July 2001

10 Haskin D, et al. A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute. draft-haskin-mpls-fast-reroute-00.txt, work in progress, June 1999

11 A Path Protection/Restoration Mechanism for MPLS Networks. draft-chang-mpls-path--protection-03.txt, work in progress

12 Ahn G, Chun W. MPLS Restoration Scheme using Least-Cost based Dynamic Backup Path

13 Cisco MPLS AutoBandwidth Allocator for MPLS Traffic Engineering: A Unique New Feature of Cisco IOS software. White Paper, Cisco Systems

14 Wright S, et al. Traffic Engineering of Load Distribution. draft-wright-load-distribution-00.txt, work in progress, July 2000

15 Lee S S, Gerla M. Fault tolerance and load balancing in QoS provisioning with multiple MPLS paths. Lecture Notes in Computer Science, vol. 2092

16 Hopps C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC2992, Nov. 2000

17 Villamizar C. OSPF Optimized Multipath (OSPF-OMP). draft-ietf-ospf-omp-02.txt, work in progress, Feb. 1999

18 Wright S, et al. Policy-Based Load-Balancing in Traffic-Engineered MPLS Networks. draft-wright-mpls-te-policy-00.txt, work in progress, June 2000