基于流水线的高性能 Web 服务器*

姚念民+, 郑名扬, 鞠九滨

(吉林大学 计算机科学与技术学院,吉林 长春 130012)

A High Performance Web Server Based on Pipeline

YAO Nian-Min+, ZHENG Ming-Yang, JU Jiu-Bin

(College of Computer Science and Technology, Jilin University, Changchun 130012, China)

+ Corresponding author: Phn: 86-431-5169350, E-mail: nianminyao@sina.com.cn

http://www.jlu.edu.cn

Received 2002-07-26; Accepted 2002-10-22

Yao NM, Zheng MY, Ju JB. A high performance Web server based on pipeline. *Journal of Software*, 2003, 14(6):1127~1133.

http://www.jos.org.cn/1000-9825/14/1127.htm

Abstract: Architectures of servers have very important effects on their performance. Some shortcomings of the existing design of servers' architectures are indicated and some design principals are presented in this paper. A new architecture of servers 'pipeline' and a design pattern 'resource manager' based on the above analysis are proposed in this paper. A real Web server is implemented by using these new techniques. The testing results show that it is much better than existing servers on performance.

Key words: Web; server; performance; pipeline; concurrency

摘 要: 服务器的结构对服务器性能有着至关重要的影响.指出了现行服务器结构设计中的缺陷,提出了一些服务器结构的设计原则,设计了流水线结构和资源管理者模式,应用这两个技术实现了一个 Web 服务器,测试了该服务器的性能,结果证明,该服务器明显优于现存 Web 服务器.

关键词: 万维网;服务器;性能;流水线;并发

中图法分类号: TP393 **文献标识码**: A

近 10 年来,Internet 在传播信息的范围和数量上都呈指数级增长,其中占绝大部分的是 Web 服务和信息.这给 Internet 上的各种 Web 服务器提出了前所未有的挑战.比如 AOL 的 Web 缓存每天服务超过一百亿次点击[1].同时,HTTP 请求经常以爆发的方式到达 Web 服务器[2].文献[3]指出,高峰时的 HTTP 请求率超过平均值的 8~10 倍,这时的负载一般超过 Web 服务器的负载能力,使其吞吐量下降.上述情况迫切要求 Web 服务器具有更高的性能,Web 服务器不仅需要满足非高峰时的工作负载需求,而且在高峰时期依然要保持较高的吞吐量.

提高 Web 服务器的性能有两种方法.一种是用多个服务器节点组成服务器群,由服务器群中的各个节点共同来提供服务.另一种是充分挖掘单个服务器硬件和软件的潜力,使其对于特定的服务发挥最大的效能.本文讨

* Supported by the National Natural Science Foundation of China under Grant No.60073040 (国家自然科学基金)

第一作者简介: 姚念民(1974－),男,黑龙江大庆人,博士生,主要研究领域为分布式系统,服务器性能.

论后者.对 Web 服务器性能最重要的要求是 Web 服务器在高负载情况下依然保持较高的吞吐量.当前,大多数 Web 服务器在用户访问高峰时期的性能表现低于人们的期望,其中有多方面的原因,分析如下:

(1) 对于采用进程池或者线程池模型的 Web 服务器,如 Apache,因为它使用一个进程来服务单个 HTTP 请求,所以服务器中同时服务的客户数就等于非空闲的进程数.在服务器繁忙的时候,同时运行的进程数达到最大值,此时,进程间的通信和进程上下文切换开销也达到最大,这些开销严重影响了 Web 服务器的性能.同时,由于受到操作系统的限制,服务器可以创建的进程数有一个最大值,这也就限制了服务器进程池中的最大进程数,即同时能够服务的客户数,当到来的 HTTP 请求数超过最大进程数时,后来的客户就只有等待了.上述问题主要来源于进程数和同时服务的客户数之间的线性关系,本文第 1 节提出一个流水线结构,可以解决这个问题.

(2) 对于单线程的 Web 服务器,如 Zeus,thttpd,Flash[4]等,它们使用单个线程,异步 I/O 方式同时服务多个客户请求,称为 SPED(单进程事件驱动)模型的服务器.由于此类服务器没有进程间通信和进程上下文切换的开销,所以在高负载下表现出很高的性能.但是这种类型的服务器具有由操作系统带来的难以克服的缺陷,文献[5]指出,大部分操作系统对于 select 调用和文件描述符分配的实现具有较弱的可扩展性,SPED 型服务器在对大量的同时连接进行管理时性能低下.同时,操作系统对异步 I/O 支持不足,以及单个线程会发生缺页中断等不可避免的阻塞,这些都会严重影响这种类型的服务器性能.SPED 型服务器的另一个不可克服的缺点是不能利用多处理器特性,我们认为在将来 SMP(对称多处理器)服务器流行的情况下,SPED 型服务器不会成为主流.本文第 1 节提出的流水线结构避免了 select 调用,并可以充分利用 SMP 的优势.

(3) 在采用多线程模型的服务器中,为了提高性能,一个重要的设计思想就是要尽量减少线程间通信.但是有些线程之间的互斥操作又是必不可少的,如对 Web 缓存的操作.当 Web 服务器处于高负载情况下,该类服务器中会有多个线程在等待进行 Web 缓存的查找和更新操作,这样就减少了并行性,限制了服务器的性能.本文第 2 节提出一个资源管理者模式,可以解决这个问题.

(4) 提高服务器性能的另一个重要设计思想是尽量减少系统调用的频率.在服务器繁忙的情况下,服务器进程如果对于每一次客户请求都重新申请资源,如进程、内存、文件描述符等来提供服务,则会遇到频繁的用户空间和内核空间的切换,这样不仅带来额外的开销,而且使 CPU 的指令流水线中断,利用率下降.在 Apache 中,进程池和基于池的内存分配都是这一设计思想的体现.本文第 1 节提出的流水线结构,有效地使资源循环回收利用,大量减少了系统调用,而且提高了资源的利用率.

本文第 1 节和第 2 节叙述流水线结构、资源管理者模式.第 3 节给出根据上面的技术实现的一个 Web 服务器 PRWS(pipeline and resource manager Web server)的性能测试结果.第 4 节叙述相关工作.第 5 节是结论.

1 流水线结构

在以下论述中,对进程和线程不加以区分.就性能来说,用线程更有优势,而对安全性来说,用进程更好一些,就功能来说,进程和线程可以互换.例如:在当前版本的 Linux 上,可以使用线程编程,但在内核中实际上是利用 Linux 特有的系统调用 clone 来创建共享内存空间的进程模拟实现的.所以使用进程还是线程,是对于性能和安全性取舍的问题,不涉及结构的根本改变.下面的叙述均采用线程一词,如不特别说明,同样适用于进程.根据上面的分析可以得出对于高性能服务器的设计原则:

(1) 使用多线程,可以发挥 SMP 主机的优势;

(2) 所需资源尽量在初始化时申请,在以后的运行中要循环利用,以便充分控制资源的使用,增加服务的实时性,减少系统调用的频率,还可以避免内存碎片的产生;

(3) 避免服务器中运行的线程数和同时服务的客户数的线性关系;

(4) 避免使用可扩展性弱的系统调用,如 select;

(5) 尽量减少线程间的同步和相互等待,增加并行性.

本文提出一种流水线结构的服务器设计模型,它符合上述第(1)~(4)条设计原则.第 2 节提出的资源管理者模式符合第(5)条设计原则.流水线模型使用以下基本概念:

(1) 数据块.它是指具有状态的包含资源的结构.编程时可以将它声明为结构(struct)或类(class).根据数据块

的状态可以知道这个数据块处于哪个处理阶段,即经过了哪些处理以及将要进行哪些处理.数据块中包含的资源主要是内存资源,这些资源一般是在系统初始化时申请的,并且可以满足一个请求在所有处理阶段的资源需求.

(2) 任务池.它是指数据块的集合,编程时可以用链表或队列实现.任务池中可以只包含相同状态的数据块,也可以包含状态不同的数据块,这时,在处理从这个任务池中取出的数据块时,必须判断它的状态,然后作相应的处理.

(3) 工作者线程(worker thread).它是指处理数据块的线程.程序流程一般是从一个任务池中取得一个数据块,进行处理,然后将它放入另一个任务池.说两个工作者线程分工不同,是指它们分别处理不同状态的数据块.

在我们实现的流水线 Web 服务器 PRWS 中,一个基本的 HTTP 请求的处理过程可以分为接受客户连接、读请求、分析请求、读文件和发送回答几个阶段.阶段的划分不是随意的,尽量将可能发生阻塞的操作单独划分为一个阶段,并使代码具有好的模块化和易读性.因为易发生阻塞的操作一般是 I/O 操作,将它们单独划分出来可以很容易地应用不同的 I/O 策略(同步或异步 I/O),而不影响其他阶段的处理.如果该阶段发生阻塞,也不会阻塞其他不会发生阻塞的操作,这样可以使系统的资源,如磁盘和 CPU 重叠地并行利用,发挥最大的效能.

PRWS 用一个数据结构来表示一个 HTTP 请求,称为 HTTP 数据块.它包含各阶段处理所需的内存资源,比如包含容纳客户发来的 HTTP 请求的内存空间.HTTP 数据块是在系统初始化时生成的,它们的初始状态为 Free,首先被放在一个空闲任务池中,由处理接受客户连接的工作者线程从空闲任务池中取得一个 HTTP 数据块,进行 accept 操作,然后把它放到具有 Accepted(接受连接完毕)状态的任务池中.进行读请求操作的工作者线程取得 accepted 状态的 HTTP 数据块,进行读请求操作,再放入具有 RequestRead(读请求完毕)状态的任务池中,如此类推.每当一个HTTP数据块依次经过以上几个处理阶段时,就完成了一个 HTTP 请求.在每个处理阶段,都有一个或多个线程(工作者线程)来处理 HTTP 数据块,改变它的状态,使之进入下一个处理阶段.在最末状态时(FileRead,读文件完毕),由发送回答数据的工作者线程进行处理,完成发送数据后,将它的状态改为 Free,再放入空闲任务池中,如此循环.如图 1 所示,方框代表各个处理过程,其中,圆角方框代表起点和终点,方框中的曲线代表工作者线程.在 PRWS 中的每个工作者线程的流程大体相同,可以用下面的伪代码表示其中的一个线程函数:

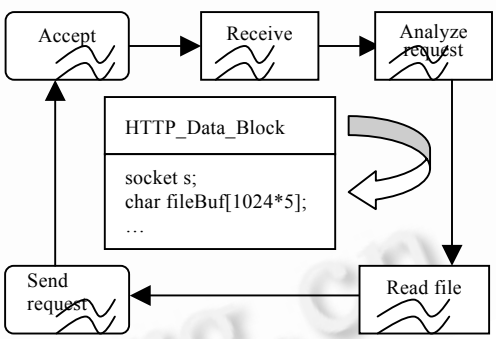

Fig.1 Pipeline architecture
图 1 流水线结构

```
HTTP_Data_Block*hdb;
while(1)   //进入无限循环
{
    getq(hdb);     //从当前的任务池中取得一个数据块
    process(hdb); //进行处理
    putq(hdb);     //将处理完的数据放入下一个任务池
}
```

从上面的叙述中可以看出,流水线 Web 服务器的优点是:

(1) 使线程数和同时服务的客户数无关,可以有效控制线程数,减少线程切换和通信的开销.同时,流水线结构使同时服务的客户数等于服务器中的 HTTP 数据块的数量,这虽然受到内存大小的限制,但在目前内存价格不断下降的情况下,无疑是一个优点.通过动态增加或减少 HTTP 数据块的数量还可以调节服务器的负载能力.

(2) 减少内存分配开销.可以在 HTTP 数据块中定义必要的空间,循环利用,不必每次都进行内存分配,减少了系统调用的频率和内存碎片的产生,而且增加了服务的实时性,它不会使一个 HTTP 请求在处理的中间阶段因为资源不足而中途夭折.

(3) 可以方便地应用最小文件优先算法.文献[6]提出采用最短连接优先的算法可以提高服务器的性能,并且对于长时间的连接只有很小的牺牲.

(4) 可以动态地改变流水线中不同阶段的工作者线程数或优先权,在易发生阻塞的瓶颈阶段多设置一些工作者线程或设置高的优先权,从而减少瓶颈.

(5) 发生阻塞的涉及 I/O 的操作单独划分为一个处理阶段,这样,在编程时可以集中处理,易于应用各种 I/O 策略,不必使用 select 系统调用.

(6) 单个工作者线程流程简单,易于编程,代码模块化和可维护性强.

(7) 应该指出,流水线结构中在处理客户请求的每一阶段,工作者线程都要对任务池进行额外的互斥和对链表的操作,这无疑会对服务器性能有一些负面影响.但基于以下原因,我们认为这种情况利大于弊:首先,大多数服务器的性能瓶颈在于对 I/O 的操作,而互斥和对链表的操作不涉及 I/O 操作,所以它们对系统整体性能影响很小;其次,由于操作系统(如 Linux 和 Windows 2000)对互斥操作的高效实现以及流水线结构中对链表的操作只有取链头节点和插入节点到链尾等简单操作,这些操作不会造成工作者线程大的延迟,即它们对其他工作者线程的并行度影响很小;最后,由于任务池的引入,流水线结构可以把整个处理过程分割成相对独立的几个阶段,每个工作者线程都可以专注于属于自己阶段的简单的工作,提高 CPU 对指令缓存的命中率,而且使系统的并行度和模块化得到大幅度的提高.

2 资源管理者模式

在多线程 Web 服务器的设计中,对于缓存的处理应该慎重考虑.因为对于缓存的操作必须是互斥的,即要求所有处理缓存的工作者线程必须串行处理缓存.而在更新缓存时,有可能发生 I/O 操作,这时就可能发生阻塞,从而会发生多个工作者线程等待处理缓存的情况,降低了并行性,本文提出资源管理者模式以解决这个问题.资源管理者模式由以下参与者组成:

(1) 互斥资源.它是指多个线程需要互斥访问的资源,比如内存,在 Web 服务器中具体指 Web 缓存.当一个线程对互斥资源进行操作时,我们称这个线程进入了临界区.

(2) 任务池和结果池.分别指存放任务块和结果块的集合,可以用链表或队列实现.任务块描述了对互斥资源的操作,结果块描述了对互斥资源处理的结果,它们都可以用结构或类来实现.

(3) 资源管理者.它是指对互斥资源进行操作的单个线程.它的流程是从任务池中取得任务块,然后执行任务,最后将处理结果填入结果块放入结果池中,如此循环.可见,资源管理者始终在临界区中,而且由于它只有一个线程,所以它在对互斥资源操作时,不必考虑与其他线程进行同步.

(4) 任务提交者.它是指想要对互斥资源进行操作的线程.它把要对互斥资源进行的操作描述封装在一个任务块中,将任务块放入任务池,然后再进行其他操作.

(5) 结果取出者.它是指从结果池中取得处理互斥资源结果的线程.在取出一个结果块之后,再根据结果作相应的处理.

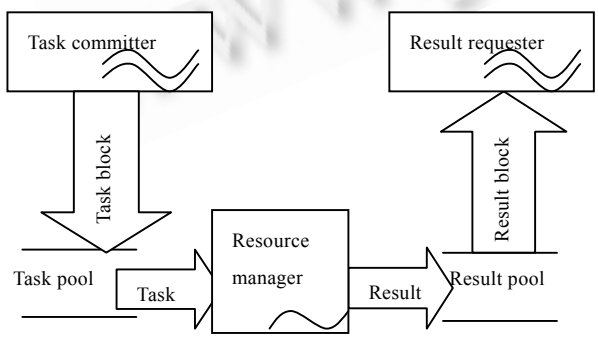

Fig.2 Design pattern of resource manager
图 2 资源管理者模式

各个参与者之间的关系如图 2 所示.由上面的叙述可以看出,资源管理者模式将原来对互斥资源的操作分为 3 个阶段,第 1 阶段由任务提交者提交任务,第 2 阶段由资源管理者进行对互斥资源的处理,第 3 阶段由结果取出者取出处理结果.整个过程需要对任务池和结果池进行两次互斥操作,比原来多了一次,但是对任务池和结果池的操作简单、快速,不会造成较大的延迟和等待.经过这样的改变,可以看到,任务提交者不必等待对互斥资源的处理完成就可以进行下一步工作,增加了并行度.同时,资源管理者在进行对互斥资源的

操作时,不必考虑同步问题,减少了编程的复杂性.应用资源管理者模式的前提是有多个线程需要互斥地访问一个共享资源,而且对共享资源的操作可以分为上面的 3 个阶段.它适用于以下几种情况:

(1) 多线程在对共享资源进行操作时,可能发生阻塞或费时过长.使用资源管理者模式可以提高并行性和效率.

(2) 线程在对共享资源进行操作的流程由于需要同步而编程复杂.使用资源管理者模式可以简化编程,使代码模块化,增加易读性和可维护性.

我们实现的 PRWS 在处理 Web 缓存时应用了资源管理者模式.在具体实现过程中,为了达到统一和简单,我们将资源管理者模式融入到流水线结构中.即将任务块和结果块都定义为 HTTP 数据块,任务池和结果池都定义为流水线结构中的任务池,任务提出者、资源管理者和结果提出者都看成是工作者线程,则资源管理者模式成为流水线结构的一个特例,其特征是资源管理者为单个工作者线程.

3 性能测试

我们应用上述技术实现了一个流水线 Web 服务器 PRWS,且分别对 Apache,Flash[4]和 PRWS 的性能进行了测试.它们分别是进程池、单线程和流水线结构的服务器,在服务器结构上具有代表性.我们使用的 Web 服务器性能测试工具是 WebStone2.5[7].我们对 WebStone 作了少许的修改,使它可以在 Linux 上运行以及在长时间的测试中更加稳定、可靠.用于测试的硬件配置如下:客户端使用 3 台 Pentium III 733,128M 以上的机器,服务器使用一个 Pentium 133,32M 内存的机器,每台机器都使用百兆网卡与百兆交换机相连.这样的配置可以使客户机和网络带宽不会成为测试的瓶颈,使测得的数据真实地反映服务器的性能.例如,之前我们将一台 Pentium 733 的机器作为服务器进行测量,它的最高吞吐量达到了 80Mbit/s 以上,占到网络理论带宽的 80%多,这时,网络带宽已经明显成为瓶颈,而在下面的测试中,服务器的最高吞吐量处于 42Mbit/s 以下,不到网络理论带宽的 50%.

测试中使用的每台机器的操作系统是 Redhat 7.2.测试用的文件集是将 WebStone 提供的标准文件集扩大 20 倍得到的.对标准文件集中的每个文件的访问频度是根据对大量的 Web 流量进行统计分析得到的[7],它符合大部分真实的 Web 服务器访问情况.但是该标准文件集很小,在测试中它会全部缓存在内存中,而很少进行磁盘操作,这不符合实际情况.文献[8,9]中仅仅使用标准文件集进行测试,我们认为,这种测试方式不能反应服务器的真实性能.所以我们实际使用的测试集是标准测试集的 20 倍,使测试集不能全部缓存于内存中.实际上,在我们的大量测试中,我们发现测试集越大,开发的 PRWS 服务器在性能上就越有优势.

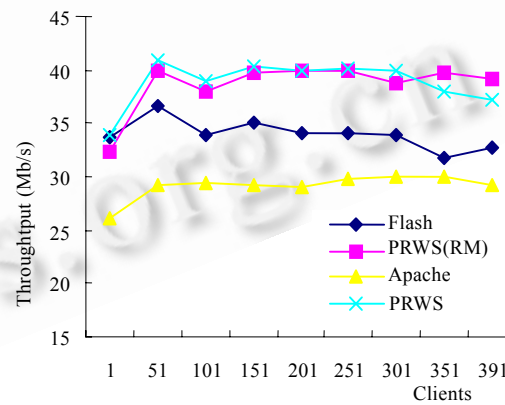

Fig.3 Test results of servers's performance

图 3 各个服务器性能测试结果

首先,我们对每个服务器包括没有应用资源管理者模式的 PRWS 进行性能测试,其中每个服务器都去掉了记录请求的功能.在测试中,并行客户数从 1 逐步增加到 391,每次增加 10 个客户,每次测试 5 分钟.测得的性能数据如图 3 所示.

从图 3 中我们可以看出,服务器的吞吐量在并行客户数为 11 时就达到饱和,这是由 WebStone 的测试方式决定的,它的每一个客户进程都不停地向服务器发送请求.PRWS 服务器相对于其他 Web 服务器具有明显的性能优势.其中 PRWS(RM)代表使用资源管理者模式的 PRWS 服务器,它在小于 131 个并行客户时,吞吐量略小于没有使用资源管理者模式的 PRWS,这是因为存在它额外做的互斥操作的影响,可以看出,该额外互斥操作对服务器性能的影响不大.当并行客户数继续增加时,PRWS(RM)已经和 PRWS 的吞吐量相差无几,并且有时还超过 PRWS,这说明,随着并行客户数的增加,资源管理者模式可以增加并行度的优势逐渐显示出来.但是,资源管理者模式的优势在这组测试中并没有充分显示出来.因为在实现中,我们设置 PRWS 可以缓存 1 024 个文件的内存映像地址,而测试用的文件集中的文件数为 5×20=100 个,这些文件的内存映像地址很快会被缓存起来,从而在后

来对缓存的查找中,每次都会命中,不会涉及到替换缓存操作,即对于互斥资源的操作时间很短,由第 2 节的叙述可以得出,在这种情况下,资源管理者模式不能充分发挥优势.在实际的互联网环境中,Web 服务器的文件集一般远远大于缓存的文件数,所以在对缓存操作时会频繁遇到替换缓存以及文件操作情况,这时资源管理者模式就会充分发挥优势.下面的另一组测试证实了我们的想法.同时,我们在图 3 中可以看到,Flash 随着并行客户数的增多,它的吞吐量下降得最快,而且在 300 和 400 个并行客户之间,它的吞吐量很不稳定,有时其吞吐量还会低于 Apache 的吞吐量,我们推测,这是由于 Flash 使用的 select 系统调用可扩展性较差造成的.同时我们认为,Flash 使用的异步文件实现方式[4]存在一定的缺陷,它会造成 Flash 中的主服务进程阻塞,Flash 在高负载情况下吞吐量不稳定的现象初步验证了我们的想法.

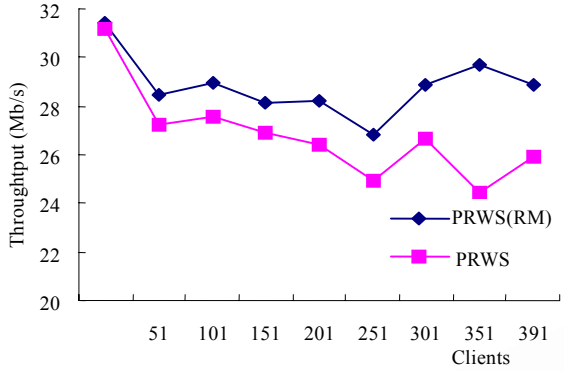

Fig.4 Test results of the servers decreased cache entries
图 4 减少缓存文件数的服务器性能测试结果

为了验证资源管理者模式的优越性,我们又对 PRWS 和 PRWS(RM)进行了一组实验.其中客户端没有改变,只是将 PRWS 和 PRWS(RM)服务器的缓存文件数改为 10,这样,在测试中会频繁发生缓存的替换操作,这也是更真实的情况.测得的性能数据如图 4 所示.由图 4 我们看出,使用资源管理者模式的 PRWS 在性能上有一定的优势.而且随着并行客户数的增加,这种优势愈加明显.同时我们看到,将缓存文件数减少以后,服务器的性能大为下降,这说明了缓存对服务器性能的重要性.另外,我们还看到,没有使用资源管理者模式的 PRWS 的吞吐量随着并行客户的增加而降低,这可能是因为多个工作者线程等待对缓存的操作而造成的.使用资源管理者模式的 PRWS 则没有这种情况,相反,它的吞吐量在高负载情况下达到顶峰.

4 相关工作

关于流水线结构,文献[6]为了测试应用最短连接优先算法的效果,设计了一个测试结构.它将接受请求连接以后的处理分为 3 步:协议处理、磁盘服务和网络服务.每一步都有多个线程在进行处理.这个测试结构与流水线结构有相似之处.但是它只是为了强调应用最短连接优先算法对服务器的性能改进,而没有作为一个通用的服务器结构提出来.同时,它的流水线结构还不完整,比如它没有强调像 HTTP 数据块这样的内存块的循环利用.而这个内存块的循环利用在流水线结构中处于很重要的地位,它可以有效地控制服务器的资源分配,服务器中分配多少个这样的内存块就可以同时服务多少个客户.而线程池结构则是有多少线程就可以同时服务多少个客户,每个线程都要占用一些操作系统的核内资源,在日趋繁忙的服务器上,这显然是不现实的.文献[10]中提出一个 SEDA 服务器结构.SEDA 结构使用事件队列把多个工作者线程联系起来.每个工作者线程从事件队列中取得事件描述,进行处理.SEDA 与文献[6]中的测试结构类似.它同样没有强调对资源的回收利用.文献[10]使用 JAVA 语言实现了一个 SEDA 型 Web 服务器,利用 JAVA 语言的内存垃圾回收机制对 Web 服务器中的废弃内存进行回收利用,这样,在 Web 服务器长期运行的情况下,容易产生内存碎片,而且频繁的申请内存的调用增加了内核空间和用户空间的切换频率,打断了 CPU 的指令流水线,限制了服务器的性能.

关于资源管理者模式,它与 Window 2000 中提供的完成端口有些类似.Windows 2000 中的完成端口是为了应用异步 I/O 操作而提供的.当应用程序要进行异步 I/O 时,可以向操作系统提交任务,然后进行其他工作,而另一个或多个线程可以在完成端口等待任务的完成.文献[9]中提出的 Proactor 模式与完成端口类似.它们都是为了避免进程等待而实现的异步操作模式.与资源管理者模式不同的是,资源管理者模式中的资源是互斥的,所以要求对互斥资源处理的线程只有一个,而且在处理过程中无须再进行互斥操作.

5 结 论

本文总结了现有的服务器结构中的设计缺陷,提出了基本的服务器结构设计原则.基于上述分析,本文提出了流水线结构和资源管理者模式.为了证实该结构设计的优越性,我们实现了基于上述设计的 Web 服务器 PRWS.对 PRWS 的性能测试结果证明,该结构在性能方面大大优于其他结构的服务器.我们认为流水线结构和资源管理者模式并不限于服务器结构的设计,也可以应用于其他程序设计领域.

References:

[1] AOL. America Online Press Data Points. 2002. http://corp.aol.com/press/press_datapoints.html.

[2] Crovella ME, Bestavros A. Self-Similarity in World Wide Web traffic: Evidence and possible causes. IEEE/ACM Transactions on Networking, 1997,5(6):835~846.

[3] Mogul JC. Network behavior of a busy web server and its clients. Technical Report, WRL 95/5, Palo Alto: DEC Western Research Laboratory, 1995.

[4] Pai VS, Druschel P, Zwaenepoel W. Flash: An efficient and portable Web server. In: Proceedings of the 1999 Annual USENIX Technical Conference. Berkeley, 1999. 199~212.

[5] Banga G, Mogul JC. Scalable kernel performance for Internet servers under realistic loads. In: Proceedings of the USENIX Annual Technical Conference. New Orleans, 1998. 1~12.

[6] Crovella ME, Frangioso R, Harchol-Balter M. Connection scheduling in Web servers. Technical Report, BUCS-TR-99-003, Boulder, 1999.

[7] Trent G, Sake M. WebStone: The first generation in HTTP server benchmarking. 1995. http://www.mindcraft.com/webstone/paper.html.

[8] Hu J, Pyarali I, Schmidt DC. Measuring the impact of event dispatching and concurrency models on Web server performance over high-speed networks. In: Proceedings of the 2nd Global Internet Conference. Phoenix: IEEE, 1997.

[9] Hu J, Pyarali I, Schmidt DC. Applying the proactor pattern to high-performance Web servers. In: Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems, IASTED. LasVegas, 1998.

[10] Welsh M, Culler D, Brewer E. SEDA: An architecture for well-conditioned, scalable Internet services. In: Proceedings of the 18th Symposium on Operating Systems Principles (SOSP-18). Banff, 2001.