过载服务器的性能研究*

姚念民+, 鞠九滨

(吉林大学 计算机科学与技术学院,吉林 长春 130012)

Study on the Performance of Overloaded Servers

YAO Nian-Min+, JU Jiu-Bin

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

+ Corresponding author: Phn: 86-10-62795215, E-mail: nianminyao@sina.com.cn

http://hpc.cs.tsinghua.edu.cn/

Received 2002-09-17; Accepted 2002-12-16

Yao NM, Ju JB. Study on the performance of overloaded servers. *Journal of Software*, 2003,14(10): 1781~1786.

http://www.jos.org.cn/1000-9825/14/1781.htm

Abstract: Related source code of Linux is analyzed for the performance of overloaded servers. Then, the phase of receiving packets of the kernel is analyzed by using queuing theory and several conclusions on the performance of overloaded servers are drawn. Based on these analyses, some methods to improve the performance of overloaded servers are presented and implemented on Linux. The results of the tests prove that these methods can avoid livelock effectively and improve the performance of overloaded servers greatly.

Key words: livelock; server; performance; overload; queuing theory

摘 要: 针对服务器过载时的性能对 Linux 的相关源代码进行了分析.用排队论分析了内核的收包过程,得出了关于服务器在过载时性能的几个结论.基于上述分析提出并在 Linux 上实现了提高服务器在过载时性能的方法.实验证明这些方法能有效防止活锁现象,极大地提高服务器在高负载情况下的性能.

关键词: 活锁;服务器;性能;过载;排队论

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,这对 Internet 上的各种服务器提出了前所未有的挑战,尤其是 Web 服务器.比如,AOL 的 Web 缓存每天服务超过 100 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].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.

人们不仅要求服务器满足正常的工作负载需求,而且在高峰时期,依然要保持较高的吞吐量.但是,服务器在高负载下的性能表现往往远低于人们的期望.我们将服务器的过载情况分为两种.一种是瞬时超载,即服务器暂时的、短时间的超载.这种情况主要是由服务器的负载特点引起的.文献[3]指出,对于 Web 请求的网络通信量

* 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,主要研究领域为分布式系统,服务器性能,网络存储.

© 中国科学院软件研究所 http://www.jos.org.cn

分布(traffic)是自相似的,即 Web 请求的通信量可以在很大范围内有显著的变化,这常常造成服务器短时间的超载.但是,这种情况持续的时间一般很短,服务器可以很快地恢复到正常的性能水平,绝大多数客户对此不 wfc 察觉.另一种是服务器长时间的超载.这种情况一般是由某一特殊事件引起的.比如,服务器受到拒绝服务攻击或者某个热点新闻的发布等等.这种情况会造成严重的后果,比如,客户会觉得服务器反应迟缓,甚至不能连通,服务器可能会崩溃,给服务提供商造成经济损失.在这种情况下,服务器性能急剧下降,其主要原因是服务器发生了“活锁(livelock)”现象.本文第 1 节将结合 Linux 源代码对活锁现象进行详细解释.

本文专注于提高服务器在过载情况下的性能.我们在分析了 Linux 相关源代码以及对服务器收包过程进行建模分析的基础上提出了几个改进方法,实验证明,经过我们的改进,服务器的性能得到了大幅度的提高,尤其是在服务器超载的情况下.

1 建模和分析

主机操作系统收包部分的具体实现对服务器在过载情况下的性能会产生重大的影响.总体上,收包过程分为 3 个处理阶段,按照接收包所经过的顺序依次为网卡硬中断处理部分、TCP/IP 软中断处理部分和应用程序处理部分,它们的优先级依次递减.每两个相邻的处理部分之间都由一个或多个输入队列相联系.例如,网卡硬中断处理部分与 TCP/IP 软中断处理部分之间由名为 input_pkt_queue 的队列相联系.为了减少在 SMP 中 CPU 之间的互斥操作,该队列的个数与主机中的 CPU 的个数相等.每个处理阶段流程都是将接收包作一定的处理之后,放入下一级的输入队列中,由下一个处理阶段将接收包从该队列中取出,作进一步的处理.

所谓活锁现象即主机在高负载情况下发生的一种现象.在这种情况下,由于某个处理阶段具有绝对的执行优先权,主机把资源都花费在该处理阶段上,使得低优先级的处理得不到执行.然而每一件有用的工作必须经过所有的处理阶段,因此,这时候主机虽然在高负荷地运转,但做了很少或根本没有做有用的工作.以在 Linux 的收包过程为例,如果主机处于重负载情况下,发来的数据包不停地到达主机,每一个包到达,都会引起网卡硬中断处理过程 rtl8139_interrupt()的调用,在硬中断处理过程中,系统分配了 sk_buff,拷贝了包的内容,在要将 sk_buff 放入输入队列时,如果发现队列已满,就会释放 sk_buff,丢弃包的内容,这样,以前做的工作就白做了.如果这种情况非常频繁地发生,那么系统虽然在高速运转,但只有很少的包被真正接收,这时就发生了活锁现象.以此类推,收包的应用程序处理阶段比软中断处理阶段的优先级更低,这两个阶段之间也会发生活锁现象.主机在处理接收包过程中遇到的活锁现象又被称为“收包活锁(receive livelock)”.

我们将主机对接收包的处理过程模拟为如图 1 所示的系统模型,该系统模型由两个服务子系统 S_1,S_2 组成,分别是高优先级和低优先级服务子系统.假设接收包的到达为泊松流,两个子服务系统的服务时间为负指数分布,接收包到达率为λ;高优先级服务子系统的服务率为μ_1,最多能容纳 l_1 个顾客;低优先级的服务子系统的服务率为μ_2,最多能容纳 l_2 个顾客.可以看出,图 1 类似于服务结构为纵列系统[4]的排队模型,但是实际上它们有很大的差别.本模型的两个服务子系统共享一个服务员,即 CPU,它们互斥地占用这个单一服务员(这里我们没有涉及 SMP 主机,后面我们将对此说明),而且高优先级的服务子系统可以无条件地抢占服务员,使处在低优先级服务子系统的顾客受到不公平的对待.注意,我们定义的μ_2 是 S_2 在独占 CPU 情况下的服务率.

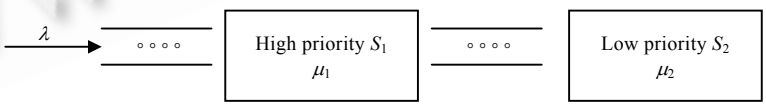

Fig.1 The queuing model of receiving packets

图 1 主机收包的随机服务系统模型

我们首先来分析 S_1 的系统性能指标.

S_1 模拟的是网卡的硬件中断处理部分.它能容纳的顾客数 l_1 一般很小,但是μ_1 却很大. 因此, S_1 一般不会出现由于队列满顾客离开的情况.又由于 S_1 可以无条件地抢占 CPU,所以它的处理可以不受 S_2 的影响.因此 S_1 可以认为是一个 M/M/1 型的随机服务系统.S_1 没有顾客的概率是[5]

$$p_0 = 1 - \lambda / \mu_1, \tag{1}$$

© 中国科学院软件研究所 http://www.jos.org.cn

则 S_1 忙的概率是

$$p_b = 1 - p_0 = \lambda / \mu_1.$$

在网络包爆发式地到来或者系统处于极重负载时,网卡即 S_1 也会产生丢包现象.此时,S_1 可以看作是 M/M/1/l_1 型随机服务系统.它的丢包率为

$$p_{1d} = \rho_1^{l_1} \frac{1-\rho_1}{1-\rho_1^{l_1+1}}, \quad \rho_1 = \lambda / \mu_1. \tag{2}$$

下面,我们来分析 S_2.

文献[4]证明了如下定理:

定理 1. M/M/1 队列的离去过程是一个泊松过程,这个过程与到达过程相同.

根据定理 1,我们可以得出,对于 S_2,顾客的到达仍然是强度为λ的泊松流.我们在如下两种情况下来分析 S_2 的运行情况.

第 1 种情况,当系统运行在正常负载时.此时,由于 S_2 可以获得足够的 CPU 资源,所以,实际的 μ_{2r} 只略大于 μ_2, 可以认为μ_{2r}=μ_2, 因此 S_2 是一个 M/M/1/l_2 型的随机服务系统.S_2 系统损失的概率是

$$p_{2d} = \rho_2^{l_2} \frac{1-\rho_2}{1-\rho_2^{l_2+1}}, \quad \rho_2 = \lambda / \mu_2. \tag{3}$$

第 2 种情况,当系统运行在超载情况时.此时,由于 CPU 经常被 S_1 占用,S_2 实际的服务率μ_{2r} 要远小于μ_2.系统在重负载情况下,用户空间的 CPU 占用率非常小,特别是 Web 服务器,所以可以假设 CPU 所有时间都花在 S_1 和 S_2 上,则在单位时间内,S_1 占用 p_b ,S_2 占用 p_0 ,所以由式(1)得到 S_2 的实际服务强度是

$$\mu_{2r} = p_0 \mu_2 = (1 - \lambda / \mu_1)\mu_2, \tag{4}$$

则 S_2 的吞吐量为

$$T_2 = \lambda \left(1 - \frac{\rho_{2r}^{l_2}(1-\rho_{2r})}{1-\rho_{2r}^{l_2+1}}\right), \quad \rho_{2r} = \lambda / \mu_{2r}. \tag{5}$$

式(5)展开后过于复杂,难以进行分析.我们可以作如下简化.实际上,在重负载情况下,

$$\because \rho_{2r} \text{ 远大于 } 1, \therefore 1 - \rho_{2r}^{l_2+1} \approx -\rho_{2r}^{l_2+1}. \tag{6}$$

将式(6)代入式(5),并且结合式(4)得到

$$T_2 \approx T_2' = \mu_{2r} = (1 - \lambda / \mu_1)\mu_2. \tag{7}$$

式(7)可以这样理解:在系统处于重负载情况下,S_2 中没有顾客的概率接近于 0,服务员几乎总是在不停地工作,所以它的吞吐量近似等于其实际的服务率.

从上面的分析,我们可以得出以下结论:

(1) 由式(2)得出,提高 l_1 和μ_1, 可以降低系统在网络包爆发式的到来或者系统处于极重负载时的网卡丢包率和提高效率.

(2) 由式(3)得出,在正常负载和瞬时超载情况下,为了减少 S_2 的丢包率并提高效率,应该提高队列长度 l_2 和服务强度μ_2.

(3) 由式(7)得出,在超载情况下,提高μ_1 和μ_2 都可以提高系统的吞吐量,但是提高μ_2 的效果更好,它与系统的吞吐量呈线性关系.

(4) 由式(7)我们还可以得出,在超载情况下,系统负载λ对性能有很大的影响,λ越大,系统的性能越低.如果 μ_1 和μ_2 不变,则它的性能曲线是以$-\mu_2/\mu_1$ 为斜率的一条直线,由于μ_2 远小于μ_1,性能曲线是斜率远大于–1、小于 0 的直线.

以上分析结果与我们的直觉相符合.

需要说明的是,在以上分析中,我们没有考虑多 CPU 的情况.可以看出,造成系统在超载情况下性能下降的根本原因是高优先级处理阶段对 CPU 的不公平抢占.因此,我们认为,如果限制高优先级处理阶段对 CPU 的占用率,或者在 SMP 主机中限制处理高优先级的 CPU 个数,都可以减轻或消除收包活锁现象.

© 中国科学院软件研究所 http://www.jos.org.cn

2 对 Linux 收包过程的改进

基于第 1 节对收包过程的分析,我们对 Linux 的收包过程作了以下改进:

(1) 首先,我们注意到,在过程 rtl8139_rx_interrupt()中提前分配了 sk_buff,并拷贝了包的内容.但是在过程 netif_rx()中,如果判断接收队列已满或正处于扼流状态,就会释放 sk_buff,这样就白白分配了内存,浪费了系统资源.如果在重负载情况下,这种情况就会频繁出现,从而降低了系统性能.我们将分配 sk_buff 的过程移到了 netif_rx()中,并且在确信该接收包一定会被放入接收队列后才分配内存和拷贝包的内容.可以看出,我们的改进避免了系统资源的浪费,从第 1 节的分析可以得出,我们提高了μ_1,从而提高了系统在高负载情况下的性能.

(2) 在网络流量中,大部分 TCP/IP 包都相对较小.比如在建立连接的 3 次握手过程中,3 个包都没有具体内容,大小为 14(帧头)+20(IP 包头)+20(TCP 包头)+4(帧尾)=58 字节[6].对于一个 HTTP 请求,一般请求包也很小,它只包含 URL 和其他一些信息,只有回答包比较大.同时,对 Web 流量的分析[7]指出,Web 流量大部分是对大约 5K 字节的网页的请求.由于最大 TCP/IP 包的大小为 1 518 字节[6],HTTP 回答包大约会占用 5 个包,这样,再加上结束连接的 3 个包,平均一次 HTTP 请求的全过程共有 12 个包,较小的包有 7 个,占总数的 58.3%.如果只针对服务器的收包来看,接收到的包接近 100%是小包.基于以上分析,我们建立了一个 sk_buff 缓冲池,其中的每个 sk_buff 大小为 128 字节,每当需要分配小于 128 字节的 sk_buff 时,先从缓冲池中分配,如果池为空,再调用 dev_alloc_skb()来分配 sk_buff.当需要释放 sk_buff 时,如果它的大小等于 128 字节,就先放入缓冲池中,如果池已满,再真正释放.经过这样的处理,我们看到系统分配和释放 sk_buff 的效率增加了,而且减少了内存碎片的产生,因此从整体上提高了系统效率.经过实验证实,sk_buff 缓冲池的命中率在 92%左右,它可以提高系统性能.

(3) 从第 1 节的结论(4)中可以得到,S_2 的到达率λ越大,系统的性能越低,所以,我们试图当系统超载时在硬中断处理中丢弃包来降低λ.这一改进的关键是如何判断系统处于超载状态,以及丢弃什么类型的包.在我们的实验环境中,根据观察可以得出,服务器在内核 CPU 占用率超过 75%以及系统分配 socket 数超过 128 时开始出现性能下降(其他实验环境可能有所不同,比如耗费内存较大的服务器可能需要以内存占用量为判定依据).因此,我们选择这个时候丢弃某些包.另外,我们选择只丢弃开始建立连接的 SYN 包.因为系统每接收到一个 SYN 包就会准备建立一个连接,所以它对系统的性能影响最大,而且如果盲目丢弃其他类型的包,可能会将系统正在等待的包丢弃,从而造成某些处理过程的等待时间延长,降低系统效率,而丢弃 SYN 包就没有这种顾虑.实验证明,这个改进可以非常有效地抑止服务器的收包活锁现象.

通过以上对 Linux 收包过程的改进,我们使 Linux 服务器的性能得到大幅度的提升,尤其是在重负载情况下.同时,我们的改进也适用于非服务器以及其他操作系统,如 FreeBSD.

3 实验结果

我们使用 1 台 Pentium 133,内存 32M 的主机作为 Web 服务器,5 台 Pentium733,内存 128M 的主机作为客户端进行测试,各个主机都安装了百兆网卡,且由 100M 交换机相连.各个客户端安装的操作系统为 Red Hat Linux 7.2,Web 服务器使用的是 Apache1.3.20.我们之所以使用较低配置的主机作为服务器,是因为要避免网络带宽成为瓶颈.在实验中我们发现,如果选用一台 Pentium 600 以上的主机作为服务器,它的吞吐量可以轻易地达到 70~80Mb/s,此时,网络的带宽已经成为瓶颈,所以测试结果不能真实反映服务器的性能.

我们首先使用 sclient[8]对服务器的性能进行测试.测试结果如图 2 所示.Sclient 可以产生超过服务器负载能力的 Web 请求,所以该测试可以清楚地显示服务器在高负载情况下的性能变化.由图 2 我们看到,未改进内核的服务器性能在超载情况下急剧下降,它的性能曲线近似为斜率为−0.09 的直线(图中已标出),这与第 1 节的结论(4)相符合.改进内核的服务器性能随着负载增加也有所下降,但坡度渐缓,最后趋于水平.同时,我们注意到,两个内核的服务器的最大吞吐量相差无几,说明内核的修改对系统在最佳状态下的性能影响甚微.在要求每秒发送 2 850 个 Web 请求的重负载情况下,服务器吞吐量提高了 134.7%.如果负载继续增加,它们的差距还会加大.可以看到,改进后的内核在重负载情况下具有明显的优势.

我们又使用 WebStone2.5[7]对两个内核的 Web 服务器进行了性能测试.测试结果如图 3 所示.WebStone 是早

© 中国科学院软件研究所 http://www.jos.org.cn

期标准的 Web 服务器性能测试工具,它的测试文件集中各个文件的访问频率是根据对大量 Web 请求统计得出的,但是由于它的测试方式决定了它很难产生使服务器超载的负载[8].所以在这个测试中,系统始终没有达到超载状态,我们所作的改进大多数没有起作用.从图 3 我们可以看出,虽然改进的服务器性能达不到未改进服务器的最大值,但是它的性能更加稳定,而且总体上更优.这些观察的结果可以从以下统计结果得到验证.未改进服务器的吞吐量平均值为 32.22Mb/s,标准差为 1.25,而改进后的服务器吞吐量平均值为 32.95Mb/s,标准差为 0.46.我们认为,由于改进后的内核需要周期性地判断是否处于活锁状态,使系统的最大吞吐量有所下降,但是增加的 sk_buff 缓冲池对系统性能的提高和稳定有一定的帮助.

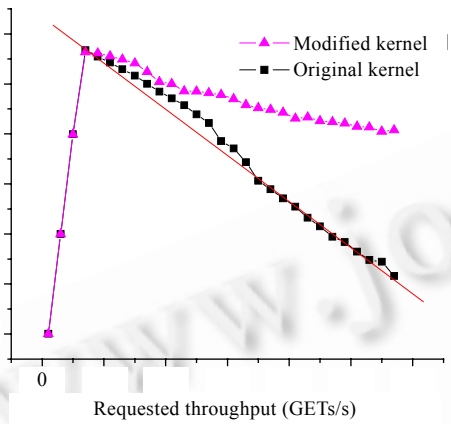

Fig.2 Test results of using sclient

图 2 用 sclient 测试服务器性能结果

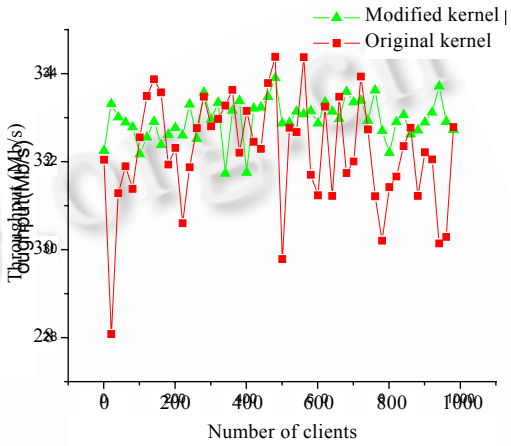

Fig.3 Test results of using WebStone

图 3 用 WebStone 测试服务器性能结果

4 相关工作

文献[8]的作者提出了一种新的方法来测试过载时的服务器性能,并且实现了一个 Web 服务器性能测试工具——sclient.它可以使用较少的测试机器产生使服务器超载的负载,给我们研究服务器在过载时的性能带来了方便.

文献[9~14]分别提出了解决服务器收包活锁问题的方法.这些工作虽然对于解决活锁现象有很大的帮助,但也存在不足.首先,它们大都是对现有操作系统的大范围改造.比如文献[9,10,13]实现较为复杂,易造成与现有的应用程序不兼容的现象.文献[13]实现了对一个路由器的修改,如果将这些修改应用于一个普通操作系统,将更加复杂.文献[11,12]的实验数据是模拟结果,没有实现真正的系统.比较而言,我们在第 3 节叙述的改进实现起来较为简单,且对应用程序透明,即使是网卡驱动程序,也只需稍作修改就可以极大地提高性能.旧的网卡驱动程序在改进后的系统中仍然可以运行,只是不能利用改进系统的先进特性.其次,上述工作没有区分服务器超载的两种不同情况,仅仅专注于活锁现象.而服务器的瞬时超载则更为普遍.文献[13]提出降低 IP 输入队列(即软中断输入队列)的长度,以减轻活锁现象.而从第 2 节的结论(2)可知,如果减少 IP 输入队列长度,将增加瞬时超载情况下的丢包率.而且从式(7)得到,在服务器超载情况下,减少 IP 输入队列对性能影响不大.上述工作对如何判断活锁状态考虑不足或没有涉及.比如文献[14]将控制机制采用的策略留待将来完成.我们则认为可以将核内 CPU 利用率、系统分配 socket 数以及内存占用量等作为判断活锁状态的依据,并应用于 Web 服务器中,取得了良好的效果.

5 结 论

本文的主要贡献是在分析 Linux 收包过程源代码和对服务器收包过程用排队论进行分析的基础上,对 Linux 的收包过程进行了改进,并且给出几个判断活锁状态的依据.实验证实了这些改进.对 Linux 的改进同样可以应用于其他操作系统,这项工作将是非常有益的.在第 1 节的分析中没有包括 SMP 主机的情况以及用户空间

© 中国科学院软件研究所 http://www.jos.org.cn

的收包过程,我们计划进行改进和补充.

References:

[1] America Online. America online press data points. http://corp.aol.com/press/press_datapoints.html.

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

[3] 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.

[4] Hua X. Queuing Theory and Stochastic Service System. Shanghai: Company of Shanghai Translation Press, 1987. 10~11 (in Chinese).

[5] Lu FS. Queuing Theory and Its Application. Changsha: Hu'nan Science and Technology Press, 1984. 122~129 (in Chinese).

[6] Stevens W. TCP/IP Illustrated Volume 1. Addison Wesley. 1993.

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

[8] Banga G, Druschel P. Measuring the capacity of a web server. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems. Monterey, 1997. 61~67.

[9] Druschel P, Banga G. Lazy receiver processing (LRP): A network subsystem architecture for server systems. In: Proceedings of the 2nd Symposium on Operating Systems Design and Implementation (OSDI'96). 1996. 261~275.

[10] Banga G, Druschel P, Mogul JC. Resource containers: A new facility for resource management in server systems. In: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation. New Orleans, 1999. 45~58.

[11] Cherkasova L, Phaal P. Session based admission control: A mechanism for improving the performance of an overloaded Web server. Technical Report, HPL-98-June, 1998.

[12] Carter R, Cherkasova L. Detecting timed-out client requests for avoiding livelock and improving Web server performance. In: Proceedings of the IEEE Symposium on Computers and Communications. Antibes, IEEE, 2000.

[13] Mogul JC, Ramakrishnan KK. Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer System, 1997,15(3):217~252.

[14] Voigt T, Tewari R, Freimuth D, Mehra A. Kernel mechanisms for service differentiation in overloaded Web servers. In: Proceedings of the 2001 USENIX Annual Technical Conference. Boston: USENIX Association, 2001.

附中文参考文献:

[4] 华兴.排队论与随机服务系统.上海:上海翻译出版公司,1987.10~11.

[5] 陆风山.排队论及其应用.长沙:湖南科学技术出版社,1984.122~129.

© 中国科学院软件研究所 http://www.jos.org.cn