网络缓存的零拷贝优化*)

周敬利　王晓锋　余胜生　夏洪涛
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　数据的冗余拷贝是影响网络缓存软件性能的重要因素。本文详细分析了当前广泛使用的网络缓存程序Squid的数据拷贝流程,利用快速文件传输和数据流拼接两种策略在Linux平台上实现了Squid的零拷贝优化。快速文件传输是将文件数据从文件系统内核缓存直接发送到远程主机,数据流拼接则是在网络协议栈内核缓存之间直接交换数据,二者都能够减少数据在内核空间和用户空间流动带来的冗余拷贝。实验证明,经过优化的Squid在主机CPU占用率和请求响应时间等方面的性能均有显著提高。

关键词　网络缓存,零拷贝,优化

Zero Copy Optimization for Web Cache

ZHOU Jing-Li　WANG Xiao-Feng　YU Sheng-Sheng　Xia Hong-Tao
(Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　The poor performance of current Web cache attributes to redundant data copies significantly. Squid, which is well known as a widely used Web cache program, is analyzed in detail in this paper from the point of view of its data movement. Zero copy optimization to Squid is implemented based on two strategies: faster file transfer and data stream splicing. The former makes file data sent to remote host from kernel buffer of local file system directly, and the latter means that input and outgoing data can be exchanged between kernel buffers of protocol stacks smoothly. Both of them reduce data copies resulted from data movement between kernel space and user space. Experiments prove that the optimized Squid predominates over original version remarkably in both CPU utilities and mean request response time.

Keywords　Web cache, Zero copy, Optimization

1　引言

网络缓存是特殊功能的代理服务器。代理服务器打破了传统的客户服务器模式,介入到客户和服务器的连接当中,接收来自任意连接方的数据包,进行处理后发送给另一方。处理算法的多样化,使代理服务器能够充当各种不同的角色,例如应用层网关防火墙、网络缓存以及移动运算代理等等[1]。本文将要讨论的网络缓存是基于因特网协议的网络缓存程序。

代理服务器的工作原理类似于路由器,负责数据的接收、处理和转发。不同之处在于,前者在网络层处理数据包,而后者则将处理数据的过程往上移到了传输层或应用层。作为网络数据流的集散地,代理服务器软件的性能显得至关重要。许多研究表明,频繁的系统调用以及网络数据流的频繁拷贝是降低代理服务器软件性能的重要因素[2]。在现有的基于应用层实现的代理服务器软件中,数据在代表网络协议栈或文件系统的内核缓存和代表应用程序的用户缓存之间存在多次流动,这些流动造成的不必要的数据拷贝浪费了系统资源,严重影响了软件性能。

本文所指的零拷贝优化是指分析现有网络缓存程序中的数据拷贝流程,最大限度地减少数据拷贝,达到性能优化的目的。实验证明,经过优化的网络缓存软件在主机的CPU占用率、用户请求的响应时间等方面均有明显改善。本文第2节简要介绍了网络缓存的工作原理,深入分析了广泛使用的网络缓存软件Squid在处理用户请求时需要经历哪些数据拷贝;第3节介绍数据流拼接的概念;第4节讲述了如何在Squid中实现数据流拼接,从而减少数据拷贝;第5节是Squid的原始版本和几个优化版本的性能对比测试;最后是本文的结论。

2　网络缓存简介

2.1　功能与原理

本文主要讨论基于HTTP协议的网络缓存。这类网络缓存的作用在于将客户(通常是Web浏览器)频繁请求的URL文件保存到本地,下次处理同样的请求时可以从本地读取数据,而不必与Web服务器建立连接,提高了网络性能。网络缓存的工作原理是:接受并分析Web浏览器发出的URL请求;判断请求的URL文件是否在本地文件系统中有一份副本,如果有而且该副本没有过期,则称为命中,按照HTTP协议将该文件直接发往Web浏览器;否则建立与浏览器指定的目的Web服务器的连接,这时网络缓存代替浏览器扮演客户的角色,取得URL文件并保存在本地文件系统,然后转发给客户浏览器。

Squid作为一个广泛使用的企业级网络缓存软件,采用单进程事件驱动模型来处理并发的多个请求,避免了多进程模型的调度开销[3]。系统调用select()提供了异步通知机制来同时维护多个句柄,保证并发的用户请求能够得到及时响应。

作为一个可选项,Squid从2.0版开始加入了多线程异步IO机制,将文件系统的创建、打开以及读写操作放入线程池

*)本研究由国家自然科学基金(60373088)和国防研究基金(4131605)资助。周敬利　教授,博士生导师,研究方向为多媒体计算与网络技术。王晓锋　博士研究生,研究方向为网络与信息安全。余胜生　教授,博士生导师,研究方向为多媒体计算与网络技术。夏洪涛　博士研究生,研究方向为网络与信息安全。

中处理，在高请求率时极大地减少了请求响应时间。本文实现的优化版本都采用了多线程异步 IO 机制。

Squid 被设计成能够工作于多种操作系统，本文选择了 Linux 作为工作平台，所有的优化、测试工作都是基于 Linux 的。以下的讨论除特别指明外，都针对 Linux 操作系统。

2.2 数据拷贝分析

Squid 是一个典型的基于应用层实现的网络缓存程序。从 Web 浏览器发出的请求数据包以及从 Web 服务器发出的响应数据包到达 Squid 站点后经由 TCP/IP 协议栈从内核空间拷贝到用户空间，读入应用程序缓冲区的数据经 Squid 分析处理后又被拷贝回内核空间，或者是网络协议栈缓存，或者是文件系统缓存。

图 1 是 Squid 在处理请求本地命中时数据流经历的拷贝示意图。Squid 发现请求中 URL 所指定的文件在本地存在有效副本时，就从文件系统中每次读取 4k 字节的数据存入临时分配的程序缓存 1 中，然后再拷贝到程序缓存 2，最后由系统调用 write 发送到本地网络接口。由此可见，Squid 中的数据流除了在内核空间和用户空间的移动外，在用户空间内部也存在不必要的拷贝。

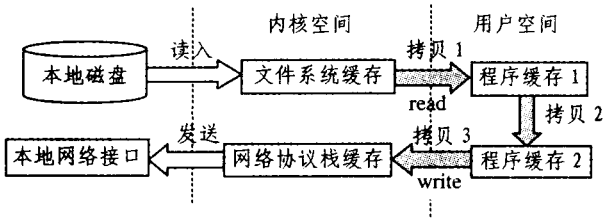

图 1 请求命中时数据拷贝

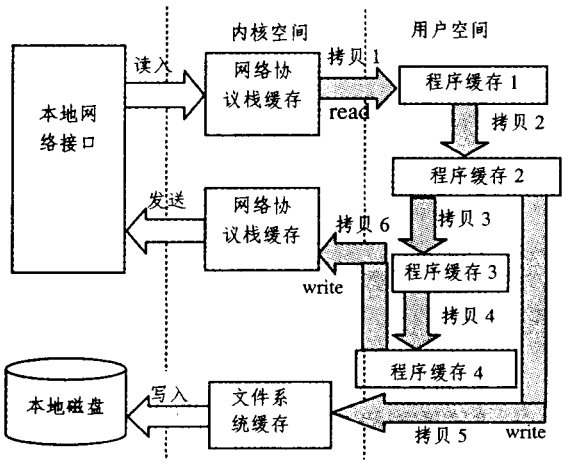

图 2 请求未命中时数据拷贝

图 2 是请求未命中时，假设 Squid 已经和浏览器指定的目的 Web 服务器建立了连接，并且来自该 Web 服务器响应数据包时已经被 Squid 捕获时的数据流拷贝示意图。程序缓存 1 是一个固定长度的静态内存区，专门用来接受响应数据。程序缓存 2 是 Squid 维护的 URL 文件的内存副本，如果命中的 URL 文件在内存中存在副本，那么文件数据可以直接从该内存副本中读取，而不必使用文件系统从本地磁盘读取；如果 HTTP 协议头指定该 URL 文件是可缓存的，那么内存副本中的数据还要写入到文件系统缓存，最终存入本地磁盘。程序缓存 3 是一个临时分配的大小为 4k 字节的内存区，每次从内存副本中读取 4k 字节，然后拷贝至程序缓存 4。程序缓存 4 利用系统调用 write 将数据写入网络协议栈缓存，最终发送至网络接口。可见 Squid 在处理一个未能命中的请求时，来自目的 Web 服务器的响应数据被发送到浏览器并缓存到本地文件系统需要经历六次拷贝，三次发生在内核与用户空间之间，三次仅仅是用户空间内部的数据移动。

3 数据流拼接

如上文所述，许多代理服务器软件是在应用层实现的，因此需要首先将所有数据从文件系统或网络协议栈完整地拷贝到应用层，这是数据频繁拷贝的主要原因。一个解决办法是将整个软件转移到内核实现，另一个则是数据流拼接。代理服务器的应用层算法往往并不需要所有的数据，将所有数据拷贝到应用层显然没有必要。只拷贝少量必须的数据，以满足应用层算法的需要，剩下的数据则只在内核空间移动，这就是数据流拼接的基本思想。例如，Squid 在处理 HTTP 协议服务时，只需要在应用层解析请求或响应数据包的 HTTP 协议头，而数据包的主体部分转发即可。

3.1 快速文件传输

快速文件传输是文件系统和网络协议栈之间的数据流拼接。当把一个本地文件传送到网络中的另一台主机时，不用将文件数据首先读入到用户空间，而是直接从文件系统缓存移动到网络协议栈缓存中，避免了多余的数据拷贝。

许多操作系统提供了专门的用户接口实现快速文件传输，例如 Windows NT 提供的 TransmitFile()、AIX 和 Linux 提供的 sendfile()等[4]。目前 Linux 提供的 sendfile()需要在文件系统缓存和网络协议栈缓存之间执行一次拷贝，而且只能从文件系统转送到网络协议栈，还不是真正意义上的全功能零拷贝用户接口。

3.2 传输控制协议数据流拼接(TCP Splicing)

代理服务器软件的传统实现方法是，程序同时维护与客户端以及与服务器端的两个网络连接，并在应用层实现这两个连接之间的数据交换，这实际上是一种应用层的数据流拼接，会带来数据频繁拷贝的问题。

传输控制协议数据流拼接则是网络协议栈内部两个不同网络连接之间的数据流拼接。如图 3 所示，传输控制协议数据流拼接是指应用层以下内核空间的数据交换，包括 IP 层拼接和套接层拼接两种实现方法。

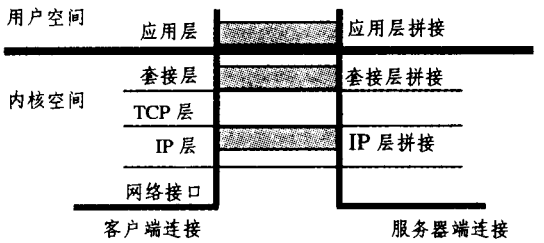

图 3 数据流拼接

IP 层拼接需要在网络协议栈的 IP 层上部修改来自一个连接的 IP 数据包中的数据序列号以及确认序列号，然后发往另一个连接。而套接口拼接则需要将一个连接的套接口接收队列中的数据包进行少量修改移动到另一个连接的发送队列，然后发送出去。两种实现方法都需要修改数据包的目的 IP 地址和目的端口号。

IP 层拼接在网络协议栈的 IP 层实现数据包在两个连接之间的交换，避免了数据包流经 TCP 层及以上的网络协议

层，减少了系统开销。IP层拼接有几点不足[5]：首先，参与拼接的两个连接必须具有相同的TCP层连接参数，例如最大报文段长度(MSS)、时间戳、窗口尺寸等等[5]；其次，应用层很难得到必要的信息对数据拼接进行有效的控制，例如，网络代理缓存软件无法及时捕获浏览器发出的取消指令，造成当前进行的数据拼接无法从服务器端获得完整的数据，已经取得的部分数据只能丢弃，浪费了网络带宽；再次，应用层得不到数据的完整拷贝，无法在本地文件系统中保存URL文件的副本，这对于网络代理缓存软件来说是难以接受的。

套接层拼接将两个连接之间的数据交换上移到套接层，数据包需要流经网络协议栈TCP层，虽然系统开销略微增加，但是这种实现方法更为灵活，适用面更广。套接层拼接独立于TCP层之上，因此并不要求TCP层连接参数必须一致[5]；应用层能够得到有效信息控制数据拼接，例如，网络代理缓存软件在捕获到浏览器的取消指令后不通知服务器，而是继续从服务器得到完整的数据，只是不再将数据继续发送给客户；应用层能够很容易地得到数据的完整拷贝并保存到本地，这使得对网络代理缓存软件的优化适用于所有处理未命中请求时的情况，而不只限于改善对不可缓存的URL文件的单纯转发操作。

4 实现

4.1 sendfile实现的快速文件传输

Squid在处理命中请求时，如果内存中没有指定URL文件的副本，则会通过文件系统从磁盘读取数据。图4是采用sendfile实现快速文件传输的数据移动示意图，对照图1，节省了两次拷贝。

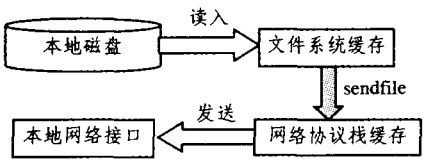

图4 快速文件传输

Squid在本地缓存中保存的文件由元数据、HTTP协议头和文件内容组成。Squid首先读入文件的元数据以及HTTP协议头，检查命中的URL文件是否有效，若有效则将HTTP协议头和文件内容发送给请求客户。快速文件传输的实现思路是，首先按照Squid原来的代码路径执行，直到能够确定URL文件有效，然后将执行流程切换到优化后的快速文件传输代码路径，利用Linux提供的sendfile系统调用完成文件剩余部分的传输。

Squid每次从文件中读取4k字节的数据发送到网络接口，数据发送完毕后执行一个回调函数clientWriteComplete()，该函数判断是否所有数据已经发送完，是则关闭客户连接，否则再从文件中读取4k字节发送出去。快速文件传输代码路径的入口点就在这个回调函数中，因为从文件中读取的第一批数据被发送到客户足以证明命中的URL文件有效。

4.2 套接层拼接的实现

套接层拼接是针对Squid处理未命中请求时所做的优化。套接层拼接在Squid中的实现与快速文件传输相似，首先执行Squid原来的代码路径分析来自Web服务器的响应HTTP协议头，如果有效，则切换到优化后的套接层拼接代码路径，利用新创建的Linux内核模块sksp提供的系统调用splice()完成对来自Web服务器剩余数据的处理。

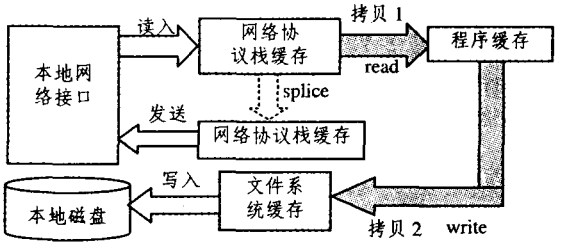

图5 套接层拼接

sksp是基于Linux平台的可加载内核模块(LKM)，提供一个新的系统调用splice()，用来将一个套接口接收队列中的若干个数据包直接移动到另一个套接口的发送队列中。如果splice()的调用者提供足够的用户空间缓冲区，本次移动的所有数据包中的数据还会拷贝到这个用户缓冲区中，称为拷贝模式。Squid的优化代码在拷贝模式下调用splice()，既能实现数据包的零拷贝转发，又能将URL文件数据保存到本地文件系统缓存。采用splice()后的数据移动状态如图5所示，一个可缓存的URL文件只需要二次拷贝就能够转发给请求客户并保存到本地文件系统缓存，对照图2所示优化前的Squid，减少了四次拷贝。优化前的Squid处理一个不可缓存的URL文件时避免了图2中的拷贝5，优化后的Squid还能进一步避免图5中所示的拷贝1和拷贝2，仅仅需要执行一次splice()系统调用，共减少了五次拷贝。

为讨论方便，将客户请求分为三类：命中请求是指被请求的URL文件在本地存在完整有效的副本，前文提到的快速文件传输就是对这类请求的优化；首客户请求是指Squid首次遇到的对某个URL文件的请求，显然首客户请求不会命中；首客户请求还未处理完毕，又有新的客户请求同一个URL文件，且该URL文件是可缓存的，称为后续请求。图5正是优化的Squid处理一个首客户请求的可缓存URL文件时的数据流移动示意图。

Squid在处理首客户请求时会创建一个新的结构store_entry，代表一个被请求的URL文件，store_entry保存了该文件的大小、数据完整性等信息，优化代码为该结构增加一个状态域splice。splice域有四种状态：预备、等待、拼接和完成。

预备：首客户请求创建的新的store_entry结构初始化为预备态。Squid在splice域的预备态按照原来的代码路径执行。

等待：Squid发现首客户请求未命中，建立起和Web服务器的连接，读取到足够的HTTP响应信息并分析完HTTP协议头后，转为等待状态。在splice域转为等待状态后，Squid执行流程切换到优化后的套接层拼接代码路径。Squid在等待态下执行优化后的代码将已经从Web服务器接收到的第一批响应数据按照原来的方法转发给客户，并在本地文件系统缓存中保存URL文件的部分副本。处于等待态时Squid不再从Web服务器端套接口读取数据至用户空间，因为剩下的响应数据全部交给优化的套接层拼接代码处理。

拼接：等待态的第一批响应数据处理完毕后，转入拼接态。Squid使用select()同时监听多个套接口，每次发现有新的数据包到达与Web服务器连接的套接口时，调用splice()将数据包转发给客户。如果请求的URL文件是可缓存的，则拷贝模式的splice()会提供用户空间的数据拷贝，用于将

URL 文件保存到本地缓存。

完成:splice()返回零表示 Web 服务器已经发送完所有数据。如果 URL 文件不可缓存,那么 splice()返回零意味着 splice 域转化为完成态。对于可缓存的 URL 文件,要等到所有数据都已经正确地保存到了本地文件系统,才最终转化为完成态。

对完成态 store_entry 结构代表的可缓存 URL 文件的请求是命中请求。而对预备、等待或拼接态 store_entry 结构代表的可缓存 URL 文件的请求则是后续请求。所有后续请求都需要直接从本地文件系统中读取数据,在首客户请求完成前,文件系统中只包含所请求 URL 文件的部分数据。

5 测试

5.1 测试环境和方法

测试网络包括三台主机,分别作为 Web 服务器、客户机以及代理服务器。三台主机的配置如表 1 所示。

表 1 测试环境

	CPU	主频	内存
客户机	PentiumIV	1.7G	256M
Web 服务器	PentiumIII	1G	256M
代理服务器	PentiumIII	800M	384M

Web 服务器和客户机运行 Polygraph 程序。Polygraph 是测试各类网络缓存软件性能的专用测试工具,能够高性能地模拟 HTTP 服务器和客户机的行为。Polygraph 客户端模仿浏览器发出 HTTP GET 请求指令,修改配置文件调整每秒钟发出的请求数、请求命中率、被请求 URL 文件大小以及缓存属性等,可以模拟出各种工作负载。

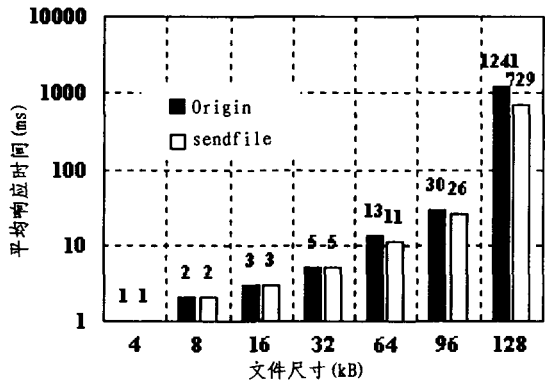

图 6 响应时间:90req/s 100%hits

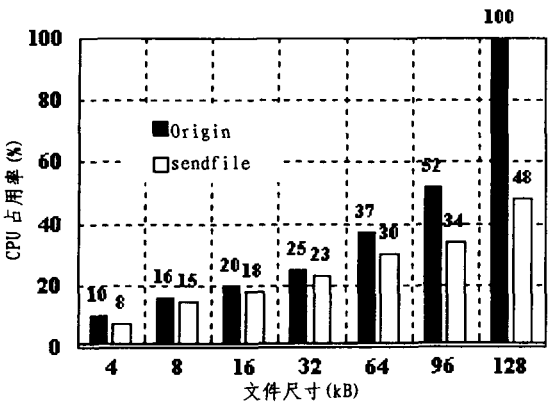

图 7 CPU 占用率:90%req/s 100%hits

代理服务器运行 Squid2.5.4 版及基于该版本的三种修改版本,分别是快速文件传输优化版、套接层拼接优化版和整合两种优化的综合版,为讨论方便,分别称为 sendfile 版、splice 版和 sendfile+splice 版,优化前的原始版称为 origin 版。所有版本都采用 Squid 定义的 aufs 文件系统。

本次测试的主要目的是分析优化后的 Squid 在 CPU 占用率和请求响应时间等方面的性能有何改进。CPU 占用率来源于 vmstat 程序,响应时间来源于 Polygraph 的分析报告。

5.2 测试分析

快速文件传输对 Squid 性能的优化主要体现在浏览器的 HTTP 请求命中时。图 6 和图 7 是请求率固定为每秒 90 个,命中率和缓存率为 100%时,origin 版和 sendfile 版的请求响应时间和 CPU 占用率对比图,请求的 URL 文件大小分别固定为 4k、8k、16k、32k、64k、96k 和 128k。从图 6 可以看到,在请求的 URL 文件较小,Squid 服务器负担较轻时,经过快速文件传输优化后的版本在请求响应时间方面没有明显优势,随着 URL 文件尺寸的增加,优化版的请求响应时间明显小于原始版。图 7 说明,优化版在 CPU 占用率方面始终低于原始版,随着服务器负载的加大,这种优势更加明显,在 URL 文件尺寸增大到 128k 时,原始版的 CPU 利用率已经达到 100%,而优化版只有 48%。综合图 6 和图 7 发现,在较低负载时,CPU 占用率都比较小,这时影响响应时间的主要因素是本地文件系统的 IO 操作,因此快速文件传输优化带来的二次拷贝的节省不能显著降低请求响应时间。在负载加大时,原始版的多余拷贝占用了太多的 CPU 资源,造成响应明显变慢,这时减少拷贝带来的性能优势才表现出来。

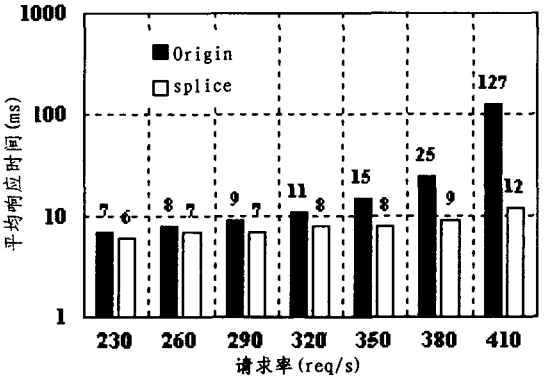

图 8 响应时间:13kB 0%hits 0%cachable

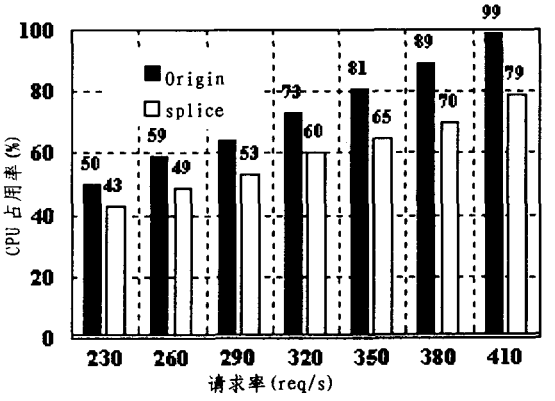

图 9 CPU 占用率:13kB 0%hits 0%cachable

套接层数据流拼接优化发生在在浏览器的 HTTP 请求未命中时。图 8 和图 9 是 origin 版和 splice 版的性能对比,被

请求的 URL 文件都是不可缓存的，文件尺寸固定为 13kB，请求命中率为 0%，这样所有的请求都会经由套接层数据流拼接优化处理。从图 8 中可以看到，随着请求率的增加，优化版的请求响应时间对请求率的增加并不敏感，始终保持在低水平，而原始版的响应时间则迅速增加，图 9 也显示出优化版在 CPU 占用方面始终低于原始版。正如前文分析，优化的 splice 版本在处理不可缓存 URL 文件请求时节省五次拷贝带来的性能提升是非常显著的。

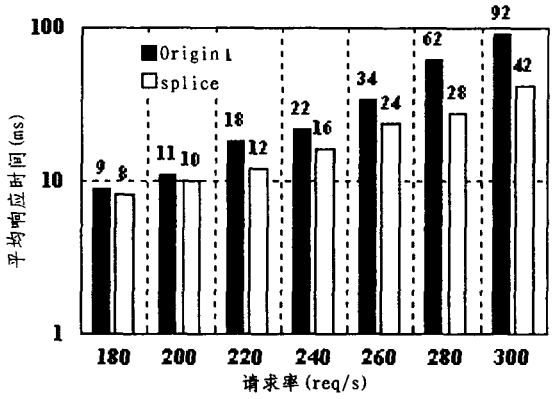

图 10　响应时间：13kB 0%hits 100%cachable

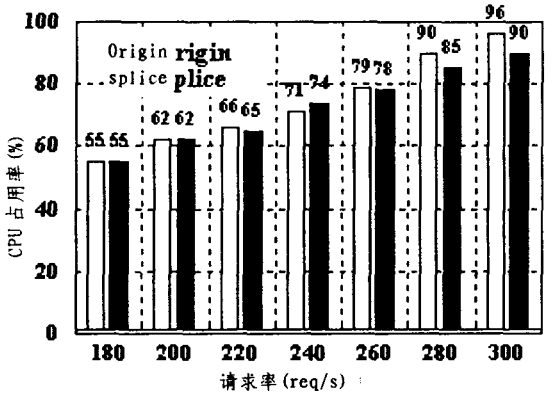

图 11　CPU 占用率：13kB 0%hits 100%cachable

图 10 和图 11 中，文件尺寸固定为 13kB，请求命中率为 0%，缓存率为 100%，即所有被请求的 URL 文件均会保存在本地缓存。尽管涉及到文件系统，文件 IO 操作也是影响 Squid 性能的重要因素，但是优化版本节省了四次数据拷贝，在响应请求方面仍然具有明显优势，在 CPU 占用方面大体持平，优化版略占优势。

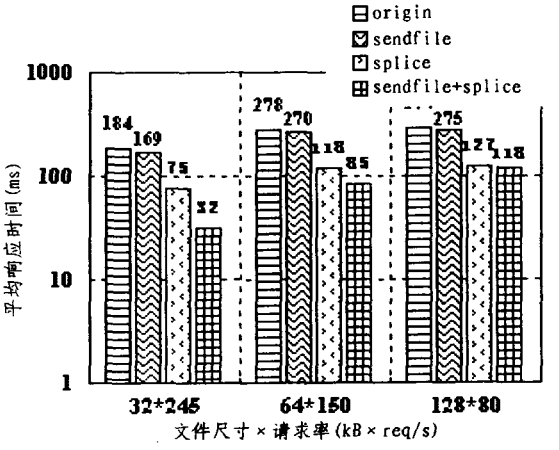

图 12　响应时间：50%hits80%cachable

图 12 和图 13 是 Squid 的原始版和三个修改版在三种文件尺寸和请求率组合下的请求响应时间和 CPU 占用对比图，请求命中率为 50%，缓存率为 80%，这样快速文件传输和套接层数据流拼接优化都能体现出来。图形显示在三种文件尺寸和请求率组合下，优化后的版本均有性能提升，组合版集成了两种优化策略，无论是请求响应时间还是 CPU 占用，优势都非常明显。

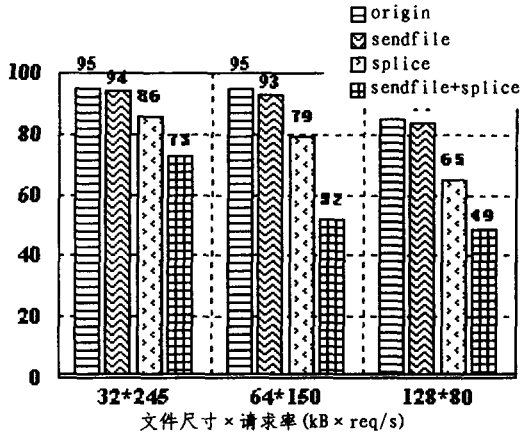

图 13　CPU 占用率：50%hits80%cachable

结论　网络缓存事实上是网络数据流的存储转发程序，因此数据的冗余拷贝是造成网络缓存性能瓶颈的重要因素。本文从减少数据拷贝的角度探讨了对现有的网络缓存软件优化的可能性，并在 Linux 平台上实现了广泛使用的网络缓存软件 Squid 的数据拷贝优化。

快速文件传输是指将数据直接从文件系统缓存发送到网络协议栈缓存，避免了数据在内核空间和用户空间的拷贝。对于 Squid 而言，在处理命中请求时，利用快速文件传输能够显著降低程序的 CPU 占用率，在负载较重时，请求相应时间也会显著减少。

数据流拼接使网络缓存软件的数据接收端和发送端能够在网络协议栈缓存之间直接交换数据。本文针对 Squid 实现的套接层数据流拼接，不仅减少了内核空间和用户空间的数据拷贝，同时还避免了用户空间内部出现的多次冗余数据拷贝。如果网络缓存对数据进行单纯的转发，例如 Squid 在处理不可缓存的未命中 URL 文件请求时，数据流拼接优化的优势非常明显。即使是进行完整的存储转发，优化的 Squid 在响应时间和 CPU 占用方面仍然具有明显优势。实验显示，在特定的测试环境下，结合快速文件传输和数据流拼接优化的 Squid 代理服务器对 HTTP 请求的响应时间比原始版本减少了 55%至 83%，CPU 占用率减少了 18 至 43 个百分点，优化效果十分显著。下一步将采用 PolyMix-4 工作负载模拟一般网络环境下的 HTTP 请求，进一步测试优化版的性能。

参 考 文 献

1　Spatscheck O, et al. Peterson: Optimizing TCP forwarder performance. IEEE/ACM Trans. Netw. 2000, 8(2): 146～157

2　Maltz D, Bhagwat P. TCP splicing for application layer proxy performance: [Technical Report RC-21139]. IBM, March 1998. ftp://ftp.cs.cmu.edu/user/dmaltz/Doc/spliceperf-tr.ps

3　Maltzahn C, Richardson K J, Grunwald D. Performance Issues of Enterprise Level Web Proxies. SIGMETRICS, 1997. 13～23

4　Joubert P, et al. High-Performance Memory-Based Web Servers: Kernel and User-Space Performance. In: USENIX Annual Technical Conf. General Track, 2001. 175～187

5　Rosu M-C, Rosu D. An evaluation of TCP splice benefits in Web proxy servers. WWW 2002. 13～24