提高 Linux 上 socket 性能

加速网络应用程序的 4 种方法

M. Tim Jones (mtj@mtjones.com), 资深软件工程师, Emulex

简介： 使用 Sockets API，我们可以开发客户机和服务器应用程序，它们可以在本地网络上进行通信，也可以通过 Internet 在全球范围内进行通信。与其他 API 一样，您可以通过一些方法使用 Sockets API，从而提高 Socket 的性能，或者限制 Socket 的性能。本文探索了 4 种使用 Sockets API 来获取应用程序的最大性能并对 GNU/Linux® 环境进行优化从而达到最好结果的方法。

本文的标签： linux, opt, socket, 性能

标记本文！

发布日期： 2006 年 2 月 13 日

级别： 中级

访问情况 ： 15405 次浏览

评论： 2 (查看 | 添加评论 - 登录)

平均分 (30 个评分)

为本文评分

在开发 socket 应用程序时，首要任务通常是确保可靠性并满足一些特定的需求。利用本文中给出的 4 个提示，您就可以从头开始为实现最佳性能来设计并开发 socket 程序。本文内容包括对于 Sockets API 的使用、两个可以提高性能的 socket 选项以及 GNU/Linux 优化。

为了能够开发性能卓越的应用程序，请遵循以下技巧：

- 最小化报文传输的延时。
- 最小化系统调用的负载。
- 为 Bandwidth Delay Product 调节 TCP 窗口。
- 动态优化 GNU/Linux TCP/IP 栈。

技巧 1. 最小化报文传输的延时

在通过 TCP socket 进行通信时，数据都拆分成了数据块，这样它们就可以封装到给定连接的 TCP payload（指 TCP 数据包中的有效负荷）中了。TCP payload 的大小取决于几个因素（例如最大报文长度和路径），但是这些因素在连接发起时都是已知的。为了达到最好的性能，我们的目标是使用尽可能多的可用数据来填充每个报文。当没有足够的数据来填充 payload 时（也称为*最大报文段长度（maximum segment size）*或 MSS），TCP 就会采用 Nagle 算法自动将一些小的缓冲区连接到一个报文段中。这样可以通过最小化所发送的报文的数量来提高应用程序的效率，并减轻整体的网络拥塞问题。

尽管 John Nagle 的算法可以通过将这些数据连接成更大的报文来最小化所发送的报文的数量，但是有时您可能希望只发送一些较小的报文。一个简单的例子是 telnet 程序，它让用户可以与远程系统进行交互，这通常都是通过一个 shell 来进行的。如果用户被要求用发送报文之前输入的字符来填充某个报文段，那么这种方法就绝对不能满足我们的需要。另外一个例子是 HTTP 协议。通常，客户机浏览器会产生一个小请求（一条 HTTP 请求消息），然后 Web 服务器就会返回一个更大的响应（Web 页面）。

解决方案

您应该考虑的第一件事情是 Nagle 算法满足一种需求。由于这种算法对数据进行合并，试图构成一个完整的 TCP 报文段，因此它会引入一些延时。但是这种算法可以最小化在线路上发送的报文的数量，因此可以最小化网络拥塞的问题。
但是在需要最小化传输延时的情况中，Sockets API 可以提供一种解决方案。要禁用 Nagle 算法，您可以设置 TCP_NODELAY socket 选项，如清单 1 所示。

清单 1. 为 TCP socket 禁用 Nagle 算法

```
int sock, flag, ret;
/* Create new stream socket */
sock = socket( AF_INET, SOCK_STREAM, 0 );
/* Disable the Nagle (TCP No Delay) algorithm */
flag = 1;
ret = setsockopt( sock, IPPROTO_TCP, TCP_NODELAY, (char *)&flag, sizeof(flag) );
if (ret == -1) {
  printf("Couldn't setsockopt(TCP_NODELAY)\n");
  exit(-1);
}
```

提示：使用 Samba 的实验表明，在从 Microsoft® Windows® 服务器上的 Samba 驱动器上读取数据时，禁用 Nagle 算法几乎可以加倍提高读性能。

回页首

技巧 2. 最小化系统调用的负载

任何时候通过一个 socket 来读写数据时，您都是在使用一个*系统调用（system call）*。这个调用（例如 read 或 write）跨越了用户空间应用程序与内核的边界。另外，在进入内核之前，您的调用会通过 C 库来进入内核中的一个通用函数（system_call()）。从 system_call() 中，这个调用会进入文件系统层，内核会在这儿确定正在处理的是哪种类型的设备。最后，调用会进入 socket 层，数据就是在这里进行读取或进行排队从而通过 socket 进行传输的（这涉及数据的副本）。
这个过程说明系统调用不仅仅是在应用程序和内核中进行操作的，而且还要经过应用程序和内核中的很多层次。这个过程耗费的资源很高，因此调用次数越多，通过这个调用链进行的工作所需要的时间就越长，应用程序的性能也就越低。
由于我们无法避免这些系统调用，因此惟一的选择是最小化使用这些调用的次数。幸运的是，我们可以对这个过程进行控制。

解决方案

在将数据写入一个 socket 时，尽量一次写入所有的数据，而不是执行多次写数据的操作。对于读操作来说，最好传入可以支持的最大缓冲区，因为如果没有足够多的数据，内核也会试图填充整个缓冲区（另外还需要保持 TCP 的通告窗口为打开状态）。这样，您就可以最小化调用的次数，并可以实现更好的整体性能。

回页首

技巧 3. 为 Bandwidth Delay Product 调节 TCP 窗口

TCP 的性能取决于几个方面的因素。两个最重要的因素是*链接带宽（link bandwidth）*（报文在网络上传输的速率）和 *往返时间（round-trip time）* 或 RTT（发送报文与接收到另一端的响应之间的延时）。这两个值确定了称为 *Bandwidth Delay Product*（BDP）的内容。给定链接带宽和 RTT 之后，您就可以计算出 BDP 的值了，不过这代表什么意义呢？BDP 给出了一种简单的方法来计算理论上最优的 TCP socket 缓冲区大小（其中保存了排队等待传输和等待应用程序接收的数据）。如果缓冲区太小，那么 TCP 窗口就不能完全打开，这会对性能造成限制。如果缓冲区太大，那么宝贵的内存资源就会造成浪费。如果您设置的缓冲区大小正好合适，那么就可以完全利用可用的带宽。下面我们来看一个例子：

```
BDP = link_bandwidth * RTT
```

如果应用程序是通过一个 100Mbps 的局域网进行通信，其 RRT 为 50 ms，那么 BDP 就是：

```
100MBps * 0.050 sec / 8 = 0.625MB = 625KB
```

注意：此处除以 8 是将位转换成通信使用的字节。

因此，我们可以将 TCP 窗口设置为 BDP 或 1.25MB。但是在 Linux 2.6 上默认的 TCP 窗口大小是 110KB，这会将连接的带宽限制为 2.2MBps，计算方法如下：

```
throughput = window_size / RTT

110KB / 0.050 = 2.2MBps
```

如果使用上面计算的窗口大小，我们得到的带宽就是 12.5MBps，计算方法如下：

```
625KB / 0.050 = 12.5MBps
```

差别的确很大，并且可以为 socket 提供更大的吞吐量。因此现在您就知道如何为您的 socket 计算最优的缓冲区大小了。但是又该如何来改变呢？

解决方案

Sockets API 提供了几个 socket 选项，其中两个可以用于修改 socket 的发送和接收缓冲区的大小。清单 2 展示了如何使用 SO_SNDBUF 和 SO_RCVBUF 选项来调整发送和接收缓冲区的大小。

注意：尽管 socket 缓冲区的大小确定了通告 TCP 窗口的大小，但是 TCP 还在通告窗口内维护了一个拥塞窗口。因此，由于这个拥塞窗口的存在，给定的 socket 可能永远都不会利用最大的通告窗口。

清单 2. 手动设置发送和接收 socket 缓冲区大小

```
int ret, sock, sock_buf_size;
sock = socket( AF_INET, SOCK_STREAM, 0 );
sock_buf_size = BDP;
ret = setsockopt( sock, SOL_SOCKET, SO_SNDBUF,
                (char *)&sock_buf_size, sizeof(sock_buf_size) );
ret = setsockopt( sock, SOL_SOCKET, SO_RCVBUF,
                (char *)&sock_buf_size, sizeof(sock_buf_size) );
```

在 Linux 2.6 内核中，发送缓冲区的大小是由调用用户来定义的，但是接收缓冲区会自动加倍。您可以进行 `getsockopt` 调用来验证每个缓冲区的大小。

巨帧（jumbo frame）

我们还可以考虑将包的大小从 1,500 字节修改为 9,000 字节（称为巨帧）。在本地网络中可以通过设置最大传输单元（Maximum Transmit Unit，MTU）来设置巨帧，这可以极大地提高性能。

就 window scaling 来说，TCP 最初可以支持最大为 64KB 的窗口（使用 16 位的值来定义窗口的大小）。采用 window scaling（RFC 1323）扩展之后，您就可以使用 32 位的值来表示窗口的大小了。GNU/Linux 中提供的 TCP/IP 栈可以支持这个选项（以及其他一些选项）。

提示：Linux 内核还包括了自动对这些 socket 缓冲区进行优化的能力（请参阅下面 表 1 中的 `tcp_rmem` 和 `tcp_wmem`），不过这些选项会对整个栈造成影响。如果您只需要为一个连接或一类连接调节窗口的大小，那么这种机制也许不能满足您的需要了。

回页首

技巧 4. 动态优化 GNU/Linux TCP/IP 栈

标准的 GNU/Linux 发行版试图对各种部署情况都进行优化。这意味着标准的发行版可能并没有对您的环境进行特殊的优化。

解决方案

GNU/Linux 提供了很多可调节的内核参数，您可以使用这些参数为您自己的用途对操作系统进行动态配置。下面我们来了解一下影响 socket 性能的一些更重要的选项。

在 `/proc` 虚拟文件系统中存在一些可调节的内核参数。这个文件系统中的每个文件都表示一个或多个参数，它们可以通过 `cat` 工具进行读取，或使用 `echo` 命令进行修改。清单 3 展示了如何查询或启用一个可调节的参数（在这种情况中，可以在 TCP/IP 栈中启用 IP 转发）。

清单 3. 调优：在 TCP/IP 栈中启用 IP 转发

```
[root@camus]# cat /proc/sys/net/ipv4/ip_forward
0
[root@camus]# echo "1" > /poc/sys/net/ipv4/ip_forward
[root@camus]# cat /proc/sys/net/ipv4/ip_forward
1
[root@camus]#
```

表 1 给出了几个可调节的参数，它们可以帮助您提高 Linux TCP/IP 栈的性能。

表 1. TCP/IP 栈性能使用的可调节内核参数

可调节的参数	默认值	选项说明
`/proc/sys/net/core/rmem_default`	"110592"	定义默认的接收窗口大小；对于更大的 BDP 来说，这个大小也应该

		更大。
/proc/sys/net/core/rmem_max	"110592"	定义接收窗口的最大大小；对于更大的 BDP 来说，这个大小也应该更大。
/proc/sys/net/core/wmem_default	"110592"	定义默认的发送窗口大小；对于更大的 BDP 来说，这个大小也应该更大。
/proc/sys/net/core/wmem_max	"110592"	定义发送窗口的最大大小；对于更大的 BDP 来说，这个大小也应该更大。
/proc/sys/net/ipv4/tcp_window_scaling	"1"	启用 RFC 1323 定义的 window scaling；要支持超过 64KB 的窗口，必须启用该值。
/proc/sys/net/ipv4/tcp_sack	"1"	启用有选择的应答（Selective Acknowledgment），这可以通过有选择地应答乱序接收到的报文来提高性能（这样可以让发送者只发送丢失的报文段）；（对于广域网通信来说）这个选项应该启用，但是这会增加对 CPU 的占用。
/proc/sys/net/ipv4/tcp_fack	"1"	启用转发应答（Forward Acknowledgment），这可以进行有选择应答（SACK）从而减少拥塞情况的发生；这个选项也应该启用。
/proc/sys/net/ipv4/tcp_timestamps	"1"	以一种比重发超时更精确的方法（请参阅 RFC 1323）来启用对 RTT 的计算；为了实现更好

		的性能应该启用这个选项。
/proc/sys/net/ipv4/tcp_mem	"24576 32768 49152"	确定 TCP 栈应该如何反映内存使用；每个值的单位都是内存页（通常是 4KB）。第一个值是内存使用的下限。第二个值是内存压力模式开始对缓冲区使用应用压力的上限。第三个值是内存上限。在这个层次上可以将报文丢弃，从而减少对内存的使用。对于较大的 BDP 可以增大这些值（但是要记住，其单位是内存页，而不是字节）。
/proc/sys/net/ipv4/tcp_wmem	"4096 16384 131072"	为自动调优定义每个 socket 使用的内存。第一个值是为 socket 的发送缓冲区分配的最少字节数。第二个值是默认值（该值会被 wmem_default 覆盖），缓冲区在系统负载不重的情况下可以增长到这个值。第三个值是发送缓冲区空间的最大字节数（该值会被 wmem_max 覆盖）。
/proc/sys/net/ipv4/tcp_rmem	"4096 87380 174760"	与 tcp_wmem 类似，不过它表示的是为自动调优所使用的接收缓冲区的值。
/proc/sys/net/ipv4/tcp_low_latency	"0"	允许 TCP/IP 栈适应在高吞吐量情况下低延时的情况；这个选项应该禁用。
/proc/sys/net/ipv4/tcp_westwood	"0"	启用发送者端的拥塞控

		制算法，它可以维护对吞吐量的评估，并试图对带宽的整体利用情况进行优化；对于 WAN 通信来说应该启用这个选项。
/proc/sys/net/ipv4/tcp_bic	"1"	为快速长距离网络启用 Binary Increase Congestion；这样可以更好地利用以 GB 速度进行操作的链接；对于 WAN 通信应该启用这个选项。

与任何调优努力一样，最好的方法实际上就是不断进行实验。您的应用程序的行为、处理器的速度以及可用内存的多少都会影响到这些参数影响性能的方式。在某些情况中，您认为有益的操作可能恰恰是有害的（反之亦然）。因此，我们需要逐一试验各个选项，然后检查每个选项的结果。换而言之，我们需要相信自己的经验，但是对每次修改都要进行验证。
提示： 下面介绍一个有关永久性配置的问题。注意，如果您重新启动了 GNU/Linux 系统，那么您所需要的任何可调节的内核参数都会恢复成默认值。为了将您所设置的值作为这些参数的默认值，可以使用 /etc/sysctl.conf 在系统启动时将这些参数配置成您所设置的值。

回页首

GNU/Linux 工具

GNU/Linux 对我非常有吸引力，这是因为其中有很多工具可以使用。尽管其中大部分都是命令行工具，但是它们都非常有用，而且非常直观。GNU/Linux 提供了几个工具 —— 有些是 GNU/Linux 自己提供的，有些是开放源码软件 —— 用于调试网络应用程序，测量带宽/吞吐量，以及检查链接的使用情况。
表 2 列出最有用的几个 GNU/Linux 工具，以及它们的用途。表 3 列出了 GNU/Linux 发行版没有提供的几个有用工具。有关表 3 中工具的更多信息请参阅 参考资料。

表 2. 任何 GNU/Linux 发行版中都可以找到的工具

GNU/Linux 工具	用途
ping	这是用于检查主机的可用性的最常用的工具，但是也可以用于识别带宽延时产品计算的 RTT。
traceroute	打印某个连接到网络主机所经过的包括一系列路由器和网关的路径（路由），从而确定每个 hop 之间的延时。

netstat	确定有关网络子系统、协议和连接的各种统计信息。
tcpdump	显示一个或多个连接的协议级的报文跟踪信息；其中还包括时间信息，您可以使用这些信息来研究不同协议服务的报文时间。

表 3. GNU/Linux 发行版中没有提供的有用性能工具

GNU/Linux 工具	用途
netlog	为应用程序提供一些有关网络性能方面的信息。
nettimer	为瓶颈链接带宽生成一个度量标准；可以用于协议的自动优化。
Ethereal	以一个易于使用的图形化界面提供了 tcpump（报文跟踪）的特性。
iperf	测量 TCP 和 UDP 的网络性能；测量最大带宽，并汇报延时和数据报的丢失情况。

回页首

结束语

尝试使用本文中介绍的技巧和技术来提高 socket 应用程序的性能，包括通过禁用 Nagle 算法来减少传输延时，通过设置缓冲区的大小来提高 socket 带宽的利用，通过最小化系统调用的个数来降低系统调用的负载，以及使用可调节的内核参数来优化 Linux 的 TCP/IP 栈。

在进行优化时还需要考虑应用程序的特性。例如，您的应用程序是基于 LAN 的还是会通过 Internet 进行通信？如果您的应用程序仅仅会在 LAN 内部进行操作，那么增大 socket 缓冲区的大小可能不会带来太大的改进，不过启用巨帧却一定会极大地改进性能！

最后，还要使用 tcpdump 或 Ethereal 来检查优化之后的结果。在报文级看到的变化可以帮助展示使用这些技术进行优化之后所取得的成功效果。

参考资料

学习

- 您可以参阅本文在 developerWorks 全球站点上的 英文原文。
- 两部分的系列文章 “Linux Socket 编程”（developerWorks，2003 年 10 月和 2004 年 1 月）可以帮助您编写 socket 应用程序。
- 请参阅 Pittsburgh Supercomputing Center 有关 TCP 友好的拥塞控制算法 的其他文章。
- 增大 MTU 可以极大地影响性能。请参阅更多有关 巨帧 及其优点的内容。

- 请参阅 ICSI Center for Internet Research 有关 选择性应答 的文章。
- 查看 TCP Westwood 主页，了解更多有关 TCP Westwood 算法的详细内容。
- 研究 North Carolina State University 的 Binary Increase Congestion TCP。
- 请阅读本文作者编写的书 *BSD Sockets Programming from a Multilanguage Perspective*（Charles River Media，2003 年 9 月），其中介绍了使用 6 种不同的语言来编写 socket 程序的技术。
- 在 developerWorks Linux 专区 中可以找到为 Linux 开发人员准备的更多资源。
- 跟踪 developerWorks 技术事件和 Webcasts 的最新进展。

获得产品和技术

- 您可以将 netlog 库 链接到一个应用程序，以便为性能分析提供方便。
- Ethereal 是一个图形化的网络协议分析器，其中包括了用于协议分析的插件架构。
- 请阅读 National Laboratory for Applied Network Research 上更多有关 Iperf 工具 的内容。
- 在您的下一个开发项目中采用 IBM 试用软件，这可以从 developerWorks 上直接下载。

讨论

- 通过参与 developerWorks blogs 加入 developerWorks 社区。

关于作者

Tim Jones 是一名嵌入式软件工程师，他是 *GNU/Linux Application Programming*、*AI Application Programming* 以及 *BSD Sockets Programming from a Multilanguage Perspective* 等书的作者。他的工程背景非常广泛，从同步宇宙飞船的内核开发到嵌入式架构设计，再到网络协议的开发。Tim 是 Emulex Corp. 的一名资深软件工程师。