网络编程

在tcp应用中，server事先在某个固定端口监听，client主动发起连接，经过三路握手后建立tcp连接。那么对单机，其最大并发tcp连接数是多少？

如何标识一个TCP连接

在确定最大连接数之前，先来看看系统如何标识一个tcp连接。系统用一个4四元组来唯一标识一个TCP连接：{local ip, local port,remote ip,remote port}。

client最大tcp连接数

client每次发起tcp连接请求时，除非绑定端口，通常会让系统选取一个空闲的本地端口（local port），该端口是独占的，不能和其他tcp连接共享。tcp端口的数据类型是unsigned short，因此本地端口个数最大只有65536，端口0有特殊含义，不能使用，这样可用端口最多只有65535，所以在全部作为client端的情况下，最大tcp连接数为65535，这些连接可以连到不同的server ip。

server最大tcp连接数

server通常固定在某个本地端口上监听，等待client的连接请求。不考虑地址重用（unix的SO\_REUSEADDR选项）的情况下，即使server端有多个ip，本地监听端口也是独占的，因此server端tcp连接4元组中只有remote ip（也就是client ip）和remote port（客户端port）是可变的，因此最大tcp连接为客户端ip数×客户端port数，对IPV4，不考虑ip地址分类等因素，最大tcp连接数约为2的32次方（ip数）×2的16次方（port数），也就是server端单机最大tcp连接数约为2的48次方。

实际的tcp连接数

上面给出的是理论上的单机最大连接数，在实际环境中，受到机器资源、操作系统等的限制，特别是sever端，其最大并发tcp连接数远不能达到理论上限。在unix/linux下限制连接数的主要因素是内存和允许的文件描述符个数（每个tcp连接都要占用一定内存，每个socket就是一个文件描述符），另外1024以下的端口通常为保留端口。在默认2.6内核配置下，经过试验，每个socket占用内存在15~20k之间。

影响一个socket占用内存的参数包括：

rmem\_max

wmem\_max

tcp\_rmem

tcp\_wmem

tcp\_mem

grep skbuff /proc/slabinfo

对server端，通过增加内存、修改最大文件描述符个数等参数，单机最大并发TCP连接数超过10万 是没问题的，国外 Urban Airship 公司在产品环境中已做到 50 万并发 。在实际应用中，对大规模网络应用，还需要考虑C10K 问题。

原文：

<http://wanshi.iteye.com/blog/1256282>

<http://www.cnblogs.com/Solstice/archive/2011/07/01/2095411.html>

<http://unix.stackexchange.com/questions/30509/what-is-the-formula-to-determine-how-much-memory-a-socket-consumes-under-linux>

<http://serverfault.com/questions/10852/what-limits-the-maximum-number-of-connections-on-a-linux-server>

<http://soft.chinabyte.com/os/285/12349285.shtml>

曾几何时我们还在寻求网络编程中[C10K](http://www.kegel.com/c10k.html)问题的解决方案，但是现在从硬件和操作系统支持来看单台服务器支持上万并发连接已经没有多少挑战性了。

我们先假设单台服务器最多只能支持万级并发连接，其实对绝大多数应用来说已经远远足够了，但是对于一些拥有很大用户基数的互联网公司，往往面临的并发连接数是百万，千万，甚至腾讯的上亿（注：QQ默认用的UDP协议）。虽然现在的**集群，分布式技术**可以为我们将并发负载分担在多台服务器上，那我们只需要扩展出数十台电脑就可以解决问题，但是我们更希望能更大的挖掘单台服务器的资源，先努力垂直扩展，再进行水平扩展，这样可以有效的节省服务器相关的开支（硬件资源，机房，运维，电力其实也是一笔不小的开支）。

那么到底一台服务器能够支持多少TCP并发连接呢？

**常识一：文件句柄限制**

在linux下编写网络服务器程序的朋友肯定都知道每一个tcp连接都要占一个文件描述符，一旦这个文件描述符使用完了，新的连接到来返回给我们的错误是“Socket/File:Can't open so many files”。

这时你需要明白操作系统对可以打开的最大文件数的限制。

**进程限制**

执行 ulimit -n 输出 1024，说明对于一个进程而言最多只能打开1024个文件，所以你要采用此默认配置最多也就可以并发上千个TCP连接。

临时修改：ulimit -n 1000000，但是这种临时修改只对当前登录用户目前的使用环境有效，系统重启或用户退出后就会失效。

重启后失效的修改（不过我在CentOS 6.5下测试，重启后未发现失效）：编辑 /etc/security/limits.conf 文件， 修改后内容为

* soft nofile 1000000
* hard nofile 1000000

永久修改：编辑/etc/rc.local，在其后添加如下内容

ulimit -SHn 1000000

**全局限制**

执行 cat /proc/sys/fs/file-nr  
输出: 9344 0 592026，  
分别为：1.已经分配的文件句柄数，2.已经分配但没有使用的文件句柄数，3.最大文件句柄数。  
但在kernel 2.6版本中第二项的值总为0，这并不是一个错误，它实际上意味着已经分配的文件描述符无一浪费的都已经被使用了 。

我们可以把这个数值改大些，用 root 权限修改 /etc/sysctl.conf 文件:

fs.file-max = 1000000

net.ipv4.ip\_conntrack\_max = 1000000

net.ipv4.netfilter.ip\_conntrack\_max = 1000000

**常识二：端口号范围限制？**

操作系统上端口号1024以下是系统保留的，从1024-65535是用户使用的。由于每个TCP连接都要占一个端口号，所以我们最多可以有60000多个并发连接。我想有这种错误思路朋友不在少数吧？（其中我过去就一直这么认为）

我们来分析一下吧

如何标识一个TCP连接：系统用一个4四元组来唯一标识一个TCP连接：{local ip, local port,remote ip,remote port}。好吧，我们拿出《UNIX网络编程：卷一》第四章中对accept的讲解来看看概念性的东西，第二个参数cliaddr代表了客户端的ip地址和端口号。而我们作为服务端实际只使用了bind时这一个端口，说明端口号65535并不是并发量的限制。

**server最大tcp连接数：**server通常固定在某个本地端口上监听，等待client的连接请求。不考虑地址重用（unix的SO\_REUSEADDR选项）的情况下，即使server端有多个ip，本地监听端口也是独占的，因此server端tcp连接4元组中只有remote ip（也就是client ip）和remote port（客户端port）是可变的，因此最大tcp连接为客户端ip数×客户端port数，对IPV4，不考虑ip地址分类等因素，最大tcp连接数约为2的32次方（ip数）×2的16次方（port数），也就是server端单机最大tcp连接数约为2的48次方。

**总结**

TCP/IP 协议规定的，只用了2个字节表示端口号。容易让人误解为1个server只允许连接65535个Client。

typedef struct \_NETWORK\_ADDRESS\_IP

{

USHORT sin\_port;//0~65535

ULONG in\_addr;

UCHAR sin\_zero[8];

} NETWORK\_ADDRESS\_IP, \*PNETWORK\_ADDRESS\_IP;

(1)其实65535这个数字，只是决定了服务器端最多可以拥有65535个Bind的Socket。也就是说，最多可以开65535个服务器进程，但是你要知道这个能够连接客户端的数量没有任何关系，Accept过来的Socket是不需要Bind任何IP地址的，也没有端口占用这一说。作为Server端的Socket本身只负责监听和接受连接操作。

(2)TCP协议里面是用[源IP+源Port+目的IP+目的 Port]来区别两个不同连接,所以连入和连出是两个不同的概念。连出Connect就不错了，需要生成随机端口，这个是有限的连入的话， 因SOCKET的分配受内存分页限制，而连接受限制（WINDOWS）。

(3)所以，千万不要误以为1个server只允许连接65535个Client。记住，**TCP连出受端口限制,连入仅受内存限制**。

例如:

server，IP：192.168.16.254，Port：8009

Client1：IP：192.168.16.1，Port：2378

Client2：IP：192.168.16.2，Port：2378

Client1和Client2虽然Port相同，但是IP不同，所以是不同的连接。

(4)想让1个server并发高效得连接几万个Client，需要使用IOCP“完成端口(Completion Port)”的技术。

详情请参考文章：<http://blog.csdn.net/libaineu2004/article/details/40087167>

上面给出的结论都是理论上的单机TCP并发连接数，实际上单机并发连接数肯定要受硬件资源（内存）、网络资源（带宽）的限制，至少对我们的需求现在可以做到数十万级的并发了，你的呢？

这种单台机器10w并发，不考虑内存cpu的实现，主要是程序网络模型的选择。项目在Github上有提供<https://github.com/yaocoder/HPNetServer>

**常见设置**

**1、修改用户进程可打开文件数限制**

在Linux平台上，无论编写客户端程序还是服务端程序，在进行高并发TCP连接处理时，最高的并发数量都要受到系统对用户单一进程同时可打开文件数量的限制(这是因为系统为每个TCP连接都要创建一个socket句柄，每个socket句柄同时也是一个文件句柄)。可使用ulimit命令查看系统允许当前用户进程打开的文件数限制：

[speng@as4 ~]$ ulimit -n

1024

这表示当前用户的每个进程最多允许同时打开1024个文件，这1024个文件中还得除去每个进程必然打开的标准输入，标准输出，标准错误，服务器监听 socket，进程间通讯的unix域socket等文件，那么剩下的可用于客户端socket连接的文件数就只有大概1024-10=1014个左右。也就是说缺省情况下，基于Linux的通讯程序最多允许同时1014个TCP并发连接。  
对于想支持更高数量的TCP并发连接的通讯处理程序，就必须修改Linux对当前用户的进程同时打开的文件数量的软限制(soft limit)和硬限制(hardlimit)。其中软限制是指Linux在当前系统能够承受的范围内进一步限制用户同时打开的文件数；硬限制则是根据系统硬件资源状况(主要是系统内存)计算出来的系统最多可同时打开的文件数量。通常软限制小于或等于硬限制。  
修改上述限制的最简单的办法就是使用ulimit命令：

[speng@as4 ~]$ ulimit -n

上述命令中，在中指定要设置的单一进程允许打开的最大文件数。如果系统回显类似于“Operation notpermitted”之类的话，说明上述限制修改失败，实际上是因为在中指定的数值超过了Linux系统对该用户打开文件数的软限制或硬限制。因此，就需要修改Linux系统对用户的关于打开文件数的软限制和硬限制。  
第一步，修改/etc/security/limits.conf文件，在文件中添加如下行：

...

# End of file

speng soft nofile 10240

speng hard nofile 10240

root soft nofile 65535

root hard nofile 65535

\* soft nofile 65535

\* hard nofile 65535

[test@iZwz9e1dh1nweaex8ob5b7Z config]$

其中speng指定了要修改哪个用户的打开文件数限制，可用’\*'号表示修改所有用户的限制；soft或hard指定要修改软限制还是硬限制；10240则指定了想要修改的新的限制值，即最大打开文件数(请注意软限制值要小于或等于硬限制)。修改完后保存文件。  
第二步，修改/etc/pam.d/login文件，在文件中添加如下行：

session required /lib/security/pam\_limits.so

这是告诉Linux在用户完成系统登录后，应该调用pam\_limits.so模块来设置系统对该用户可使用的各种资源数量的最大限制(包括用户可打开的最大文件数限制)，而pam\_limits.so模块就会从/etc/security/limits.conf文件中读取配置来设置这些限制值。修改完后保存此文件。  
第三步，查看Linux系统级的最大打开文件数限制，使用如下命令：

[speng@as4 ~]$ cat /proc/sys/fs/file-max

12158

这表明这台Linux系统最多允许同时打开(即包含所有用户打开文件数总和)12158个文件，是Linux系统级硬限制，所有用户级的打开文件数限制都不应超过这个数值。通常这个系统级硬限制是Linux系统在启动时根据系统硬件资源状况计算出来的最佳的最大同时打开文件数限制，如果没有特殊需要，不应该修改此限制，除非想为用户级打开文件数限制设置超过此限制的值。修改此硬限制的方法是修改/etc/rc.local脚本，在脚本中添加如下行：

echo 22158 > /proc/sys/fs/file-max

这是让Linux在启动完成后强行将系统级打开文件数硬限制设置为22158。修改完后保存此文件。  
完成上述步骤后重启系统，一般情况下就可以将Linux系统对指定用户的单一进程允许同时打开的最大文件数限制设为指定的数值。如果重启后用 ulimit-n命令查看用户可打开文件数限制仍然低于上述步骤中设置的最大值，这可能是因为在用户登录脚本/etc/profile中使用ulimit -n命令已经将用户可同时打开的文件数做了限制。由于通过ulimit-n修改系统对用户可同时打开文件的最大数限制时，新修改的值只能小于或等于上次 ulimit-n设置的值，因此想用此命令增大这个限制值是不可能的。所以，如果有上述问题存在，就只能去打开/etc/profile脚本文件，在文件中查找是否使用了ulimit-n限制了用户可同时打开的最大文件数量，如果找到，则删除这行命令，或者将其设置的值改为合适的值，然后保存文件，用户退出并重新登录系统即可。  
通过上述步骤，就为支持高并发TCP连接处理的通讯处理程序解除关于打开文件数量方面的系统限制。

**2、修改网络内核对TCP连接的有关限制（参考对比下篇文章“优化内核参数”）**

在Linux上编写支持高并发TCP连接的客户端通讯处理程序时，有时会发现尽管已经解除了系统对用户同时打开文件数的限制，但仍会出现并发TCP连接数增加到一定数量时，再也无法成功建立新的TCP连接的现象。出现这种现在的原因有多种。  
**第一种原因**可能是因为Linux网络内核对本地端口号范围有限制。此时，进一步分析为什么无法建立TCP连接，会发现问题出在connect()调用返回失败，查看系统错误提示消息是“Can’t assign requestedaddress”。同时，如果在此时用tcpdump工具监视网络，会发现根本没有TCP连接时客户端发SYN包的网络流量。这些情况说明问题在于本地Linux系统内核中有限制。其实，问题的根本原因在于Linux内核的TCP/IP协议实现模块对系统中所有的客户端TCP连接对应的本地端口号的范围进行了限制(例如，内核限制本地端口号的范围为1024~32768之间)。当系统中某一时刻同时存在太多的TCP客户端连接时，由于每个TCP客户端连接都要占用一个唯一的本地端口号(此端口号在系统的本地端口号范围限制中)，如果现有的TCP客户端连接已将所有的本地端口号占满，则此时就无法为新的TCP客户端连接分配一个本地端口号了，因此系统会在这种情况下在connect()调用中返回失败，并将错误提示消息设为“Can’t assignrequested address”。有关这些控制逻辑可以查看Linux内核源代码，以linux2.6内核为例，可以查看tcp\_ipv4.c文件中如下函数：

static int tcp\_v4\_hash\_connect(struct sock \*sk)

请注意上述函数中对变量sysctl\_local\_port\_range的访问控制。变量sysctl\_local\_port\_range的初始化则是在tcp.c文件中的如下函数中设置：

void \_\_init tcp\_init(void)

内核编译时默认设置的本地端口号范围可能太小，因此需要修改此本地端口范围限制。  
第一步，修改/etc/sysctl.conf文件，在文件中添加如下行：

net.ipv4.ip\_local\_port\_range = 1024 65000

这表明将系统对本地端口范围限制设置为1024~65000之间。请注意，本地端口范围的最小值必须大于或等于1024；而端口范围的最大值则应小于或等于65535。修改完后保存此文件。  
第二步，执行sysctl命令：

[speng@as4 ~]$ sysctl -p

如果系统没有错误提示，就表明新的本地端口范围设置成功。如果按上述端口范围进行设置，则理论上单独一个进程最多可以同时建立60000多个TCP客户端连接。  
**第二种无法建立TCP连接的原因**可能是因为Linux网络内核的IP\_TABLE防火墙对最大跟踪的TCP连接数有限制。此时程序会表现为在 connect()调用中阻塞，如同死机，如果用tcpdump工具监视网络，也会发现根本没有TCP连接时客户端发SYN包的网络流量。由于 IP\_TABLE防火墙在内核中会对每个TCP连接的状态进行跟踪，跟踪信息将会放在位于内核内存中的conntrackdatabase中，这个数据库的大小有限，当系统中存在过多的TCP连接时，数据库容量不足，IP\_TABLE无法为新的TCP连接建立跟踪信息，于是表现为在connect()调用中阻塞。此时就必须修改内核对最大跟踪的TCP连接数的限制，方法同修改内核对本地端口号范围的限制是类似的：  
第一步，修改/etc/sysctl.conf文件，在文件中添加如下行：

net.ipv4.ip\_conntrack\_max = 10240

这表明将系统对最大跟踪的TCP连接数限制设置为10240。请注意，此限制值要尽量小，以节省对内核内存的占用。  
第二步，执行sysctl命令：

[speng@as4 ~]$ sysctl -p

如果系统没有错误提示，就表明系统对新的最大跟踪的TCP连接数限制修改成功。如果按上述参数进行设置，则理论上单独一个进程最多可以同时建立10000多个TCP客户端连接。

**3、使用支持高并发网络I/O的编程技术**

在Linux上编写高并发TCP连接应用程序时，必须使用合适的网络I/O技术和I/O事件分派机制。  
可用的I/O技术有同步I/O，非阻塞式同步I/O(也称反应式I/O)，以及异步I/O。《[BIO,NIO,AIO的理解](http://www.cnblogs.com/duanxz/p/5230856.html)》

在高TCP并发的情形下，如果使用同步I/O，这会严重阻塞程序的运转，除非为每个TCP连接的I/O创建一个线程。但是，过多的线程又会因系统对线程的调度造成巨大开销。因此，在高TCP并发的情形下使用同步 I/O是不可取的，这时可以考虑使用非阻塞式同步I/O或异步I/O。非阻塞式同步I/O的技术包括使用select()，poll()，epoll等机制。异步I/O的技术就是使用AIO。  
从I/O事件分派机制来看，使用select()是不合适的，因为它所支持的并发连接数有限(通常在1024个以内)。如果考虑性能，poll()也是不合适的，尽管它可以支持的较高的TCP并发数，但是由于其采用“轮询”机制，当并发数较高时，其运行效率相当低，并可能存在I/O事件分派不均，导致部分TCP连接上的I/O出现“饥饿”现象。而如果使用epoll或AIO，则没有上述问题(早期Linux内核的AIO技术实现是通过在内核中为每个 I/O请求创建一个线程来实现的，这种实现机制在高并发TCP连接的情形下使用其实也有严重的性能问题。但在最新的Linux内核中，AIO的实现已经得到改进)。  
综上所述，在开发支持高并发TCP连接的Linux应用程序时，应尽量使用epoll或AIO技术来实现并发的TCP连接上的I/O控制，这将为提升程序对高并发TCP连接的支持提供有效的I/O保证。  
推荐配置（把原/etc/sysctl.conf内容清掉，把下面内容复制进去）：

net.ipv4.ip\_local\_port\_range = 1024 65536

net.core.rmem\_max=16777216

net.core.wmem\_max=16777216

net.ipv4.tcp\_rmem=4096 87380 16777216

net.ipv4.tcp\_wmem=4096 65536 16777216

net.ipv4.tcp\_fin\_timeout = 10

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.tcp\_timestamps = 0

net.ipv4.tcp\_window\_scaling = 0

net.ipv4.tcp\_sack = 0

net.core.netdev\_max\_backlog = 30000

net.ipv4.tcp\_no\_metrics\_save=1

net.core.somaxconn = 262144

net.ipv4.tcp\_syncookies = 0

net.ipv4.tcp\_max\_orphans = 262144

net.ipv4.tcp\_max\_syn\_backlog = 262144

net.ipv4.tcp\_synack\_retries = 2

net.ipv4.tcp\_syn\_retries = 2

这个配置参考于cache服务器varnish的推荐配置和SunOne 服务器系统优化的推荐配置。

varnish调优推荐配置的地址为：<http://varnish.projects.linpro.no/wiki/Performance>

不过varnish推荐的配置是有问题的，实际运行表明“net.ipv4.tcp\_fin\_timeout = 3”的配置会导致页面经常打不开；并且当网友使用的是IE6浏览器时，访问网站一段时间后，所有网页都会打不开，重启浏览器后正常。可能是国外的网速快吧，我们国情决定需要调整“net.ipv4.tcp\_fin\_timeout = 10”，在10s的情况下，一切正常（实际运行结论）。

修改完毕后，执行：

/sbin/sysctl -p /etc/sysctl.conf

/sbin/sysctl -w net.ipv4.route.flush=1

命令生效。为了保险起见，也可以reboot系统。

调整文件数：  
linux系统优化完网络必须调高系统允许打开的文件数才能支持大的并发，默认1024是远远不够的。

执行命令：

Shell代码

echo ulimit -HSn 65536 >> /etc/rc.local

echo ulimit -HSn 65536 >>/root/.bash\_profile

ulimit -HSn 65536

备注：  
对mysql用户可同时打开文件数设置为10240个；  
将Linux系统可同时打开文件数设置为1000000个（一定要大于对用户的同时打开文件数限制）；  
将Linux系统对最大追踪的TCP连接数限制为20000个（但是，建议设置为10240；因为对mysql用户的同时打开文件数已经限制在10240个；且较小的值可以节省内存）；  
将linux系统端口范围配置为1024～30000（可以支持60000个以上连接，不建议修改；默认已经支持20000个以上连接）；

综合上述四点，TCP连接数限制在10140个。  
这10240个文件中还得除去每个进程必然打开的标准输入，标准输出，标准错误，服务器监听 socket，进程间通讯的unix域socket等文件。

因此，当需要对TCP连接数进行调整时只需要调整ulimit参数。

Linux下查看tcp连接数及状态命令：

netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

作者：Caiaolun  
链接：https://www.jianshu.com/p/71d554222f9e  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。