调整参数和不调整差别非常大，默认参数都设置比较小，在大并发，高负载时基本不能满足。调整之后，问题基本解决。  
vi /etc/sysctl.conf 编辑sysctl.conf  
net.ipv4.ip\_local\_port\_range = 1024  65000  
net.ipv4.tcp\_timestamps=1  
kernel.core\_uses\_pid = 1  
net.ipv4.tcp\_syncookies=1  
net.ipv4.tcp\_syn\_retries=1  
net.ipv4.tcp\_synack\_retries=1  
net.ipv4.tcp\_tw\_recycle=1  
net.ipv4.tcp\_tw\_reuse=1  
net.ipv4.tcp\_fin\_timeout=5  
net.ipv4.route.gc\_timeout=20  
net.ipv4.tcp\_keepalive\_time=20  
net.ipv4.tcp\_max\_syn\_backlog= 655360  
net.ipv4.ip\_conntrack\_max = 655360  
先简单介绍下 背景吧  
通 过webbench压nginx+memcache时，被压的服务器nginx，memcache的cpu使用非常低，而且服务器负载也很小，但是 webbench的结果却不乐观，失败非常多，通过分析，压力根本没到nginx和memcache，可能在服务器级就挡住了压力，并且发现/var /log/message里面有关于ip\_conntrack\_max(一个连接hash表)满了的报错，然后顺藤摸瓜，通过查看linux的man tcp 手册，一个参数一个参数的研究，终于总结了上面的一些配置，解决了问题。  
  
现象：/var/log/message里面有关于ip\_conntrack\_max 如ip\_conntrack: table full, dropping packet.

解决方法：

观察链接跟踪表是(/proc/net/ip\_conntrack)

IP\_conntrack 表示连接跟踪数据库(conntrack database)，代表NAT机器跟踪连接的数目，连接跟踪表能容纳多少记录是被一个变量控制的，他可由内核中的ip- sysctl函数配置。每一个跟踪连接表会占用350字节的内核存储空间，时间一长就会把默认的空间填满，那么默认空间时多少？我以redhat为例在内 存为64MB的机器上时4096,内存为128MB是 8192,内存为256MB是16376，那末就能在/proc/sys/net/ipv4/ip\_conntrack\_max里查看、配置。  
针对于我们的业务模式，比较重要的是  
net.ipv4.ip\_local\_port\_range = 1024 65000 表示用于向外连接的端口范围。缺省情况下很小：32768到61000，改为1024到65000。  
net.ipv4.tcp\_tw\_reuse = 1 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；  
net.ipv4.tcp\_tw\_recycle = 1 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

net.ipv4.ip\_conntrack\_max = 655360

net.ipv4.netfilter.ip\_conntrack\_tcp\_timeout\_established = 21600

这个值是控制 红色数字 这个时间的，其他相关参数都在/proc/sys/net/ipv4/netfilter目录下

 cat /proc/net/ip\_conntrack  
udp      17 21 src=192.168.0.18 dst=211.147.6.3 sport=24889 dport=53 packets=1 bytes=71 src=211.147.6.3 dst=192.168.0.18 sport=53 dport=24889 packets=1 bytes=152 use=1  
tcp      6 52 ESTABLISHED src=192.168.0.18 dst=192.168.0.17 sport=52318 dport=3306 packets=44 bytes=2474 src=192.168.0.17 dst=192.168.0.18 sport=3306 dport=52318 packets=43 bytes=2716 [ASSURED] use=1

vi /etc/modprobe.conf   
options ip\_conntrack hashsize=855360  更改  conntrack\_buckets

===========================================================  
===========================================================  
我写的不是很好，还是转一篇写的好多把，哈哈，文章所涉及的英文解释，都是摘自linux  man tcp的内容  
下面是摘自eve‘s blog，总结的很好，借用一下，不过有的参数可能针对的业务不同，不太一样。  
  
三次握手以及其中的各种状态：  
  
SYN(Synchronize Sequence Numbers)。  
  
同步序列编号  
  
ACK (ACKnowledge Character）  
  
在数据通信传输中，接收站发给发送站的一种传输控制字符。它表示确认发来的数据已经接受无误。  
=================================  
  
client发送syn至server  
  
此时客户端的状态变为SYN\_SENT  
  
client(syn=j)====>server  
  
server收到syn,并发送syn+ack到client,  
  
这种过程server状态由listen变为SYN\_RECV,并等待客户端再次发来ack数据  
  
client<=========server(syn=k,ack=j+1)  
  
client接收到server发过来的syn+ack,并向服务端发送ACK,服务器接收后由SYN\_RECV变为ESTABLISHED  
  
client(ACK(ack=k+1))========>server  
  
此种情况下，服务端在三次握手的变迁是  
  
LISTEN->SYN\_RECV ->ESTABLISHED  
  
客户端的三次握手的变迁是  
  
SYN\_SENT ->ESTABLISHED  
  
====================================  
  
在这种过程中要注意的  
  
一、首先server有个用来接收client发送的syn并对syn进行排队的队列，如果队列满了，新的请求不被接受。  
  
此队列长度控制参数:  
  
net.ipv4.tcp\_max\_syn\_backlog   
  
对应文件(/proc/sys/net/ipv4/tcp\_max\_syn\_backlog ) 默认是1024  
view sourceprint?  
1     [root@web]# cat /proc/sys/net/ipv4/tcp\_max\_syn\_backlog  
2     1024  
  
二、然后是SYN-ACK重传：当server向client发送syn+ack没有收到相应，server将重传，然后再重传。。。控制这个重传次数的参数是  
  
tcp\_synack\_retries  
  
对应文件(/proc/sys/net/ipv4/tcp\_synack\_retries )默认值是5，对应于180秒左右时间  
view sourceprint?  
1     [root@web ~]# cat /proc/sys/net/ipv4/tcp\_synack\_retries  
2     5  
view sourceprint?  
关于tcp\_synack\_retries的英文解释：  
The maximum number of times a SYN/ACK segment for a passive TCP connection will be retransmitted. This number should not be higher than 255. The default value is 5.  
  
备注:与此相对应的client的参数是:  
tcp\_syn\_retries  
The maximum number of times initial SYNs for an active TCP connection attempt will be retransmitted. This value should not be higher than 255. The default value is 5, which corresponds to approximately 180 seconds.  
  
三、关于tcp\_syncookies  
  
SYN Cookie原理及其在Linux内核中的实现  
  
http://www.ibm.com/developerworks/cn/linux/l-syncookie/?ca=dwcn-newsletter-linux  
  
SYN Cookie是对TCP服务器端的三次握手协议作一些修改，专门用来防范SYN Flood攻击的一种手段。它的原理是，在TCP服务器收到TCP SYN包并返回TCP SYN+ACK包时，不分配一个专门的数据区，而是根据这个SYN包计算出一个cookie值。在收到TCP ACK包时，TCP服务器在根据那个cookie值检查这个TCP ACK包的合法性。如果合法，再分配专门的数据区进行处理未来的TCP连接。  
view sourceprint?  
1     [root@web~]# cat /proc/sys/net/ipv4/tcp\_syncookies  
2     1  
  
===========================================  
  
典型故障处理  
  
如果服务器syn\_recv的条数过多，可以采取的操作是:  
  
减少server==>client重传syn+ack的次数。  
  
加大syn队列长度，防止无法响应新的连接  
view sourceprint?  
1     echo "net.ipv4.tcp\_max\_syn\_backlog = 4096" >>/etc/sysctl.conf  
2     echo "net.ipv4.tcp\_synack\_retries = 1" >>/etc/sysctl.conf  
3     sysctl -p  
  
当受到syn攻击的时候，启用syn-cookie(默认启用，在/etc/sysctl.conf里本身就有参数配置)  
view sourceprint?  
1     echo 1 >/proc/sys/net/ipv4/tcp\_syncookies  
  
=================================================  
  
四次握手  
  
下面说tcp/ip的第四次握手，分析主动关闭和被动关闭两种。  
  
A:因为如果是CLIENT端主动断掉当前连接，那么双方关闭这个TCP连接共需要四个packet：  
  
setup  
Client     --->   FIN（M）   --->   Server  
  
client发送一个FIN给server,（说它不跟你玩了）,client由ESTABLISHED->FIN\_WAIT1  
  
Client    <---  ACK(M+1)  <---   Server   
  
SER VER收到fin后发送ack确认（拿出两人信物），状态由ESTABLISHED->close\_wait  
  
  
client收到server的ack确认,只是改变状态ESTABLISHED->FIN\_WAIT1->FIN\_WAIT2，继续等server发送数据。  
  
Client  <--  FIN(N)    <--  Server  
  
server继续发送FIN到client（好就不玩了吧）,状态ESTABLISHED->close\_wait->LAST\_ACK,等待client发送ack做最后的确认  
  
Client  -->  ACK(N+1)  -->  Server  
  
client收到FIN,马上发送ack确认,状态ESTABLISHED->FIN\_WAIT1->FIN\_WAIT2->TIME\_WAIT[2MSL超时]->closed  
  
server收到ack确认，状态ESTABLISHED->close\_wait->LAST\_ACK->CLOSED.  
  
  
  
client关闭连接很好想，有点要搞清楚的是，server端什么时候会发起丢掉连接的操作：  
  
有些是应用程序控制的。nginx.conf为例  
  
keepalive\_timeout  0;  
[root@lvs-2 ~]# curl -I http://www.XXX.com  
Connection: close  
  
  
keepalive\_timeout  600;  
[root@lvs-2 ~]# curl -I http://www.XXX.com  
Connection: keep-alive  
  
这种规定了连接时间的，到了时间连接会断掉。  
  
如果没有规定的，则按照系统keepalived定时器的设置进行，具体参数如下：  
  
[root@web]# sysctl -a|grep tcp\_keepalive  
view sourceprint?  
1     net.ipv4.tcp\_keepalive\_intvl = 75  
2     net.ipv4.tcp\_keepalive\_probes = 9  
3     net.ipv4.tcp\_keepalive\_time = 7200     
4     连接两端一直没发送数据，间隔120分钟，两小时后开始第一次探测，间隔75秒后第二次探测，探测9次，最后放弃连接。有四种探测的情况，详见  
  
  
  
四种状况其实最后一种没什么意义，能考虑到下面三种就行了  
  
1 client正常，每进行一次通讯，net.ipv4.tcp\_keepalive\_time重置一次。  
  
2 一直到7200+75\*9后也无法获取client反馈信息，则认为client已经关闭并终止连接(连接超时)  
  
3 client重启， 收到探测后返回一个复位（RST）信息。server收到后终止连接（连接被对方复位）