**第十二章 CPU-内存-IO-网络调优**

**本节所讲内容：**

**12.1 有关CPU的调优**

**12.2 有关内存的调优**

**12.3 有关I/O的调优**

**12.4 有关网络的调优**

**12.5 有关内核参数的调优**

**12.1 关于CPU 中央处理器调优**

**12.1.1 CPU处理方式：**

**1. 批处理，顺序处理请求。(切换次数少，吞吐量大)**

**2. 分时处理。(如同"独占"，吞吐量小)(时间片，把请求分为一个一个的时间片，一片一片的分给CPU处理)我们现在使用x86就是这种架构**

**3. 实时处理**

**例：**

**批处理——以前的大型机（Mainframe）上所采用的系统，需要把一批程序事先写好（打孔纸带），然后计算得出结果   
 分时——现在流行的PC机和服务器都是采用这种运行模式，即把CPU的运行分成若干时间片分别处理不同的运算请求  
 实时——一般用于单片机上，比如电梯的上下控制，对于按键等动作要求进行实时处理**

**12.1.2 查看CPU一分钟有多个切换多少次。**

**查看内核一秒钟中断CPU次数：**

**[root@xuegod63 ~]# grep HZ /boot/config-3.10.0-693.el7.x86\_64**

**CONFIG\_NO\_HZ=y**

**# CONFIG\_HZ\_100 is not set**

**# CONFIG\_HZ\_250 is not set**

**# CONFIG\_HZ\_300 is not set**

**CONFIG\_HZ\_1000=y**

**CONFIG\_HZ=1000 #1秒钟有1000次中断**

**注： 此文件/boot/config-3.10.0-693.el7.x86\_64 是编译内核的参数文件**

**12.1.3 调整进程优先级使用更多CPU**

**调整进程nice值，让进程使用更多的CPU**

**优先级控制：**

**nice值 #范围， -20 ~ 19 越小优先级越高 普通用户0－19**

**nice**

**作用：以什么优先级运行进程 。默认优先级是0**

**语法： nice -n 优先级数字 命令**

**例：**

**# nice -n -5 vim a.txt # vim进程以-5级别运行**

**查看：**

**ps -axu | grep a.txt**

**[root@xuegod63 ~]# ps -axu | grep a.txt**

**Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ**

**root 24318 0.0 0.2 143624 3280 pts/4 S+ 17:00 0:00 vim b.txt**

**[root@xuegod63 ~]# top -p 24318**

**PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND**

**24129 root 15 -5 140m 3336 2200 S 0.0 0.3 0:00.08 vim**

**renice #修改正在运行的进程的优先级**

**#renice -n 5 PID #修改进程优先级**

**例：**

**#renice -n 5 24318**

**[root@xuegod63 ~]# top -p 24318**

**PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND**

**24129 root 15 5 140m 3336 2200 S 0.0 0.3 0:00.08 vim**

**检测一下范围： -20 - 19**

**[root@xuegod63 ~]# renice -n -21 24129**

**24129: old priority 5, new priority -20**

**[root@xuegod63 ~]# renice -n 20 24129**

**24129: old priority -20, new priority 19**

**12.1.4 CPU亲和力**

**taskset 作用：在多核情况下，可以认为指定一个进程在哪颗CPU上执行程序，减少进程在不同CPU之前切换的开销**

**安装：**

**[root@xuegod63 ~]# rpm -qf `which taskset `**

**util-linux-2.23.2-43.el7.x86\_64**

**语法： taskset -c N 命令**

**例1：本机是4核CPU ，指定vim命令在第一个CPU上运行**

**[root@xuegod63 ~]# taskset -c 0 vim a.txt #1号CPU ID是0**

**[root@xuegod63 ~]# ps -axu | grep vim**

**Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ**

**root 2614 1.3 0.2 143696 3332 pts/0 S+ 18:39 0:00 vim a.txt**

**[root@xuegod63 ~]# taskset -p 2614 # -p 要查看的进程ID**

**pid 2614's current affinity mask: 1 #CPU亲和力掩码，1代表第一个CPU核心**

**例2：查sshd进程运行在哪几个CPU上**

**[root@xuegod63 ~]# ps -axu | grep sshd**

**Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ**

**root 2030 0.0 0.0 64068 1140 ? Ss 18:26 0:00 /usr/sbin/sshd**

**[root@xuegod63 ~]# taskset -p 2030**

**pid 2030's current affinity mask: f #说明sshd在4颗CPU上随机进行切换。**

**说明：**  **Cpu ID 号码，对应的16进制数为：**  **CPU ID:      7      6      5      4      3      2      1      0  
 对应的10数为:        128  64     32   16      8      4      2      1**

**当前, 我的系统中cpu ID 的为（0,1,2,3）**  **pid 2030's current affinity mask: f的值为cpu ID 16进制的值的和（1+2+4+8=f),转换成二进制为：1111**  **这个说明了（pid=2030)的这个sshd进程工作在cpu ID 分别为0,1,2,3这个四个cpu上面的切换。**

**注： 我们的CPU是4核心，所以taskset -c后可以跟： 0,1,2,3**

**例：指定vim c.txt 程序运行在第2和第4个CPU上**

**[root@xuegod63 ~]# taskset -c 1,3 vim b.txt**

**[root@xuegod63 ~]# ps -axu | grep vim**

**Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ**

**root 6314 1.5 0.2 143612 3280 pts/1 S+ 14:41 0:00 vim b.txt**

**root 6317 0.0 0.0 103300 848 pts/2 S+ 14:41 0:00 grep vim**

**[root@xuegod63 ~]# taskset -p 6314**

**pid 6314's current affinity mask: a**

**# a为十进制的10=2+8**

**注：在哪个CPU上运行，那一位就赋为1 。**

**12.1.5 CPU 性能监控**

**理解运行队列，利用率，上下文切换对怎样CPU 性能最优化之间的关系，早期提及到性能是相对于基准线数据的，在一些系统中，通常预期所达到的性能包括:**

**Run Queues ­ 每个处理器应该运行队列不超过1­3 个线程.**

**例如： 一个双核处理器应该运行队列不要超过6 个**

**注：有两个特殊的进程永远在运行队列中待着：当前进程和空进程idle。**

****

**12.1.6 CPU 利用率比例分配：**

**如果一个CPU 被充分使用，利用率分类之间均衡的比例应该是**

**65% ­ 70% User Time #用户态**

**30% ­ 35% System Time #内核态**

**0% ­ 5% Idle Time #空闲**

**Context Switches ­ 上下文切换的数目直接关系到CPU 的使用率，如果CPU 利用率保持在上述均衡状态时，有大量的上下文切换是正常的。**

**实例1：持续的CPU 利用率**

**在这个例子中，这个系统的CPU被充分利用**

**[root@xuegod63 ~]# vmstat 1 10 # 本机为单核CPU，执行vmstat显示以下内容**

**procs --------memory---------- ---swap-- -----io---- --system-- -----cpu-----**

**r b swpd free buff cache si so bi bo in cs us sy id wa st**

**3 0 0 130644 86244 609860 0 0 4 1 531 25 0 0 20 0 0**

**4 0 0 130620 86244 609860 0 0 0 0 638 62 0 0 14 0 0**

**2 0 0 130620 86244 609860 0 0 0 0 658 62 0 0 13 0 0**

**4 0 0 130620 86244 609860 0 0 0 0 688 62 0 0 11 0 0**

**注：**

**根据观察值，我们可以得到以下结论:**

**1、有大量的中断(in) 和较少的上下文切换(cs)。这意味着一个单一的进程正在大量使用cpu**

**2、进一步显示某单个应用，user time(us) 经常在86%或者更多。**

**执行top -》按P-》查看使用CPU最多的进程**

**3、运行队列还在可接受的性能范围内，其中有2个地方，是超出了允许限制.**

**实例2：超负荷调度**

**在这个例子中，内核调度中的上下文切换处于饱和**

**# vmstat 1 #通过查看vmstat输出结果，分析当前系统中出现的问题**

**procs memory swap io system cpu**

**r b swpd  free  buff  cache si so bi  bo  in cs  us sy  wa id**

**2 1 207740 98476 81344 180972 0 0 2496 0 900 2883  4 12  57  27**

**0 1 207740 96448 83304 180984 0 0 1968 328  810 2559  8 9  83  0**

**0 1 207740 94404 85348 180984 0 0 2044 0  829 2879  9 6  78  7**

**0 1 207740 92576 87176 180984 0 0 1828 0  689 2088  3 9  78  10**

**2 0 207740 91300 88452 180984 0 0 1276 0  565 1282  7 6  83  4**

**3 1 207740 90124 89628 180984 0 0 1176 0  551 1229  2 7  91  0**

**根据观察值,我们可以得到以下结论:**

**1、上下文切换数目高于中断数目，说明kernel中相当数量的时间都开销在：上下文切换线程.**

**2、大量的上下文切换将导致CPU 利用率不均衡，很明显实际上等待io 请求的百分比(wa)非常高,以及user time百分比非常低(us). 说明磁盘比较慢，磁盘是瓶颈**

**3、因为CPU 都阻塞在IO请求上，所以运行队列里也有相当数个的可运行状态线程在等待执行.**

**12.2 有关内存的调优**

**12.2.1内存调优相关内容：**

**关于内存，一般情况不用调优，我们在这里分析一些情况：**

**BUFFER inode节点索引缓存 缓存 写时用，先写入到内存**

**CACHE block块/页缓存 快取 读时用，先读入到内存**

**buffers #缓存从磁盘读出的内容 ，这种理解是片面的**

**cached #缓存需要写入磁盘的内容 ，这种理解是片面的**

**例1：**

**终端1： free -h**

**或：vmstat**

**终端2：find /**

**终端1:free -m #查看 buffer 增长**

**例2：**

**终端1： free -m**

**或：vmstat**

**终端2： grep aaaa / -R**

**终端1:free -m #查看 CAHCE 增长**

**CACHE：页缓存，内存页 一页尺寸 4KB**

**对象文件系统 块block 1kB 2kB 4kB**

**扇区sectors 512b**

**12.2.2 手动清空buffer+cache ：**

**[root@xuegod63 ~]# cat /proc/sys/vm/drop\_caches #默认是 0**

**0**

**[root@xuegod63 ~]# free -m**

**[root@xuegod64 ~]# free**

**total used free shared buff/cache available**

**Mem: 999720 290728 367972 7396 341020 495424**

**Swap: 2097148 0 2097148**

**[root@apache ~]# sync # 把内存中的数据写入磁盘**

**[root@xuegod63 ~]# echo 1 > /proc/sys/vm/drop\_caches**

**[root@xuegod63 ~]# free -m**

**total used free shared buff/cache available**

**Mem: 976 283 484 7 208 495**

**Swap: 2047 0 2047**

**12.3 I/O调优 相关内容**

**12.3.1 资源限制**

**限制用户资源配置文件：/etc/security/limits.conf**

**[root@xuegod63 ~]# vim /etc/security/limits.conf**

**每行的格式：  
 用户名/@用户组名    类型(软限制/硬限制)   选项     值**

**通常我们在服务器上需要开放的内容：**

**实例1： 永久修改一个进程可以打开的最大文件数**

**vim /etc/security/limits.conf #在最添加：  
 \* soft nofile 1024000**

**\* hard nofile 1024000**

**注：soft是一个警告值，而hard则是一个真正意义的阀值，超过就会报错。soft一定要比hard小。**

**最大打开的文件数(以文件描叙符，file descripter计数)**

**2、启动系统： reboot #永久生效的缺点，必须重启系统**

**3、检查：**

**root@xuegod63 ~]# ulimit -n**

**1024000**

**[root@xuegod63 ~]# useradd kill #以普通用户登录，测试**

**[root@xuegod63 ~]# su - kill**

**[mkkk@xuegod63 ~]$ ulimit -n**

**1024000**

**方法二：#临时修改**

**[root@xuegod63 ~]# ulimit -n 10000**

**[root@xuegod63 ~]# ulimit -n**

**10000**

**说明：/etc/security/limits.conf 是模块pam\_limits.so的配置文件。**

**pam相关配置文件：**

**/lib64/security/ #pam模块所在目录**

**/etc/security/ #pam每个模块的配置文件**

**/etc/pam.d/ #使用pam功能的服务和应用程序的配置文件**

**说明：查看系统中哪些应用程序和服务使用了：pam\_limits.so 模块**

**[root@xuegod63 ~]# grep pam\_limits.so /etc/pam.d/ -R**

**。。。**

**/etc/pam.d/system-auth:session required pam\_limits.so**

**例2：nproc #用户可以打开的最大进程数**

**[root@xuegod63 ~]# vim /etc/security/limits.d/90-nproc.conf #RHEL6 必须这个文件中配置**

**改：**

**\* soft nproc 10240**

**为：**

**\* soft nproc 66666**

**\* hard nproc 66666**

**[root@xuegod63 ~]# reboot #最好重启一下**

**[root@xuegod63 ~]# ulimit -u**

**66666**

**或：**

**再打一个终端，直接查看**

**[root@xuegod63 ~]# ulimit -u**

**66666**

**临时：**

**[root@xuegod63 ~]# ulimit -u 60000**

**[root@xuegod63 ~]# ulimit -u**

**60000**

**注:默认用户可用的最大进程数量1024.这样以apache用户启动的进程就数就不能大于1024了。**

**[root@apache ~]# ulimit -a**

**core file size (blocks, -c) 0 kdump转储功能打开后产生的core file大小限制**

**data seg size (kbytes, -d) unlimited 数据段大小限制**

**scheduling priority (-e) 0**

**file size (blocks, -f) unlimited 文件大小限制**

**pending signals (-i) 27955**

**max locked memory (kbytes, -l) 32**

**max memory size (kbytes, -m) unlimited**

**open files (-n) 1024 打开的文件个数限制**

**pipe size (512 bytes, -p) 8 管道大小的限制**

**POSIX message queues (bytes, -q) 819200 消息队列大小**

**real-time priority (-r) 0**

**stack size (kbytes, -s) 10240 栈大小**

**cpu time (seconds, -t) unlimited CPU时间使用限制**

**max user processes (-u) 27955 最大的用户进程数限制**

**virtual memory (kbytes, -v) unlimited 虚拟内存限制**

**file locks (-x) unlimited**

**测试硬盘速度：**

**安装：**

**[root@xuegod64 ~]# yum -y install hdparm**

**[root@xuegod64 ~]# hdparm -T -t /dev/sda**

**/dev/sda:**

**Timing cached reads: 3850 MB in 2.00 seconds = 1926.60 MB/sec**

**#2秒中直接从内存的 cache读取数据的速度读 3850 MB。 平均1926.60 MB/sec**

**Timing buffered disk reads: 50 MB in seconds = 13.17 MB/sec**

**#3.80秒中从硬盘缓存中读 50 MB。 seconds = 13.17 MB/sec**

**参数：**

**-t perform device read timings #不使用预先的数据缓冲, 标示了Linux下没有任何文件系统开销时磁盘可以支持多快的连续数据读取.**

**-T perform cache read timings #直接从内存的 cache读取数据的速度。实际上显示出被测系统的处理器缓存和内存的吞吐量.**

**测试硬盘写命令： dd**

**在使用前首先了解两个特殊设备**

**/dev/null 伪设备，回收站.写该文件不会产生IO开销**

**/dev/zero 伪设备，会产生空字符流，读该文件不会产生IO开销**

**测试方法：**

**a.测试磁盘的IO写速度**

**[root@xuegod64 ~]# dd if=/dev/zero of=/test.dbf bs=8k count=3000**

**3000+0 records in**

**3000+0 records out**

**24576000 bytes (25 MB) copied, 1.04913 s, 23.4 MB/s**

**可以看到，在1.1秒的时间里，生成25M的一个文件，IO写的速度约为122.6MB/sec；**

**当然这个速度可以多测试几遍取一个平均值，符合概率统计.**

**time 命令： 执行命令并计时**

**[root@xuegod64 ~]# time dd if=/dev/zero of=/test.dbf bs=8k count=3000**

**3000+0 records in**

**3000+0 records out**

**24576000 bytes (25 MB) copied, 1.04913 s, 23.4 MB/s**

**real 0m1.061s 12:00 出去吃饭**

**user 0m0.002s 路上 20分**

**sys 0m0.770s 吃10分钟**

**注释：**

**1)实际时间(real time): 从command命令行开始执行到运行终止的消逝时间；**

**2)用户CPU时间(user CPU time): 命令执行完成花费的用户CPU时间，即命令在用户态中执行时间总和；**

**3)系统CPU时间(system CPU time): 命令执行完成花费的系统CPU时间，即命令在核心态中执行时间总和。**

**其中，用户CPU时间和系统CPU时间之和为CPU时间，即命令占用CPU执行的时间总和。实际时间要大于CPU时间，因为Linux是多任务操作系统，往往在执行一条命令时，系统还要处理其它任务。排队时间没有算在里面。**

**另一个需要注意的问题是即使每次执行相同命令，但所花费的时间也是不一样，其花费时间是与系统运行相关的。**

**12.4 网络相关调优**

**12.4.1 网卡绑定技术**

**/双线冗余**

**功能**

**\带宽增备**

**100M／8 ＝ 12.5MByte/s**

**Bonding技术**

**什么是网卡绑定及简单原理**

**网卡绑定也称作"网卡捆绑"，就是使用多块物理网卡虚拟成为一块网卡，以提供负载均衡或者冗余，增加带宽的作用。当一个网卡坏掉时，不会影响业务。这个聚合起来的设备看起来是一个单独的以太网接口设备，也就是这几块网卡具有相同的IP地址而并行链接聚合成一个逻辑链路工作。这种技术在Cisco等网络公司中，被称为Trunking和Etherchannel 技术，在Linux的内核中把这种技术称为bonding。**

**12.4.2 技术分类**

**1. 负载均衡**

**对于bonding的网络负载均衡是我们在文件服务器中常用到的，比如把三块网卡，当做一块来用，解决一个IP地址，流量过大，服务器网络压力过大的问题。为了解决同一个IP地址，突破流量的限制，毕竟网线和网卡对数据的吞吐量是有限制的。如果在有限的资源的情况下，实现网络负载均衡，最好的办法就是 bonding。**

**2. 网络冗余**

**对于服务器来说，网络设备的稳定也是比较重要的，特别是网卡。在生产型的系统中，网卡的可靠性就更为重要了。在生产型的系统中，大多通过硬件设备的冗余来提供服务器的可靠性和安全性，比如电源。bonding 也能为网卡提供冗余的支持。把多块网卡绑定到一个IP地址，当一块网卡发生物理性损坏的情况下，另一块网卡自动启用，并提供正常的服务，即：默认情况下只有一块网卡工作，其它网卡做备份。**

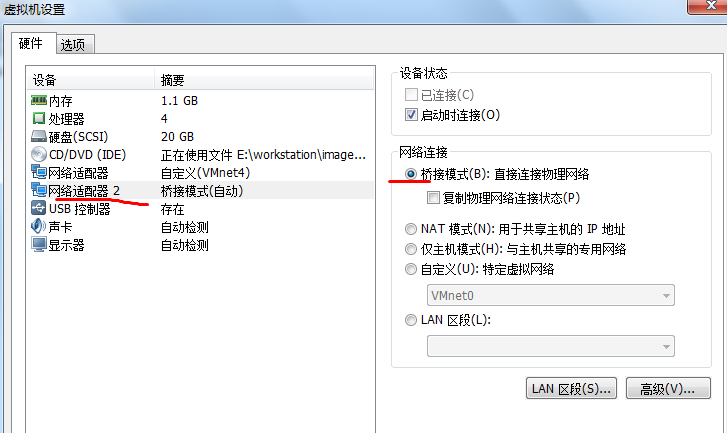
**12.4.3 实战：配置多网卡绑定技术**

**RHEL 7**

**配置环境：**

**xuegod63 配置两双网卡，网卡ens33和ens37都桥接**

**添加网卡：**

 **[root@xuegod63 ~]# cd /etc/sysconfig/network-scripts/**

**[root@xuegod63 network-scripts]# ls ifcfg-ens3\***

**ifcfg-ens33 ifcfg-ens37**

**12.4.3 双网卡绑定目标**

**网卡绑定模式：active-backup - 主备模式**

**一个网卡处于活跃状态，另一个处于备份状态，所有流量都在主链路上处理，当活跃网卡down掉时，启用备份网卡。**

**绑定网卡：ens33+ens37=bond0**

**设置网卡ens33为主网卡(优先处于活跃状态)，ens37为辅网卡(备份状态，主网卡链路正常时，辅网卡处于备份状态)。**

**12.4.5 双网卡绑定步骤：**

**查看物理网卡信息:**

**[root@xuegod63 ~]# nmcli device**

**设备 类型 状态 连接**

**virbr0 bridge 连接的 virbr0**

**ens33 ethernet 连接的 ens33**

**ens37 ethernet 连接的 ens37**

**[root@xuegod63 ~]# nmcli connection show**

**名称 UUID 类型 设备**

**ens33 7cd11800-7199-4cae-8927-ec49303cfe52 802-3-ethernet ens33**

**ens37 605551ec-6f72-368f-a5f5-cf2ae3fedd45 802-3-ethernet ens37**

**12.4.6 删除网卡连接信息：**

**本次Network bonding配置中，需要将ens33和ens37绑定为bond0，并且设置ens33为主网卡，首先需要这两块网卡现有的配置信息，否则team0创建完成后，未删除的网卡配置信息会影响bond0的正常工作。**

**如果nmcli connection show命令输出中无将要进行配置的网卡连接信息，则无需进行删除操作。**

**[root@xuegod63 network-scripts]# nmcli connection delete ens33**

**成功删除连接 'ens33'（7cd11800-7199-4cae-8927-ec49303cfe52）。**

**[root@xuegod63 network-scripts]# nmcli connection delete ens37**

**成功删除连接 'ens37'（605551ec-6f72-368f-a5f5-cf2ae3fedd45）。**

**[root@xuegod63 network-scripts]# nmcli connection show**

**名称 UUID 类型 设备**

**virbr0 0c6bda20-636e-4555-a5eb-10ebffc43c38 bridge virbr0**

**有线连接 1 33136afd-2e13-310d-9c9d-cf4858545cf0 802-3-ethernet ens33**

**有线连接 2 3b9c673a-53eb-3f0c-91ae-1b63015ec734 802-3-ethernet ens37**

**网卡连接信息删除成功。**

**说明：**

**这里删除的其实就是/etc/sysconfig/network-scripts目录下两块网卡的配置文件**

**[root@xuegod63 network-scripts]# pwd**

**/etc/sysconfig/network-scripts**

**[root@xuegod63 network-scripts]# ls ifcfg-\***

**ifcfg-lo**

**[root@xuegod63 network-scripts]# nmcli connection add type bond ifname bond0 con-name bond0 miimon 100 mode active-backup primary ens33 ip4 192.168.1.63/24**

**连接“bond0”(d558d571-3171-4a8d-b71c-2ff04aca68c1) 已成功添加。**

**注解：**

**primary ens33：指定主网卡为ens33**

**mode active-backup：指定bonding模式为active-backup（主动备份）**

**miimon 100：以毫秒为单位指定 MII 链接监控的频率(默认为0，即关闭此功能；配置miimon参数时，最好从100开始)。**

**bonding内核模块必须在ifcfg-bondN中的BONDING\_OPTS="bonding parameters"中指定，各参数间使用空格分隔，不要在/etc/modprobe.d/bonding.conf和已经弃用的/etc/modprobe.conf文件中指定bonding配置选项。**

**配置完成后，此时会在/etc/sysconfig/network-scripts目录下生成ifcfg-bond0的配置文件：**

**[root@xuegod63 network-scripts]# cat ifcfg-bond0**

**DEVICE=bond0**

**BONDING\_OPTS="miimon=100 mode=active-backup primary=ens33"**

**TYPE=Bond**

**BONDING\_MASTER=yes**

**PROXY\_METHOD=none**

**BROWSER\_ONLY=no**

**BOOTPROTO=none**

**IPADDR=192.168.0.63**

**PREFIX=24**

**DEFROUTE=yes**

**IPV4\_FAILURE\_FATAL=no**

**IPV6INIT=yes**

**IPV6\_AUTOCONF=yes**

**IPV6\_DEFROUTE=yes**

**IPV6\_FAILURE\_FATAL=no**

**IPV6\_ADDR\_GEN\_MODE=stable-privacy**

**NAME=bond0**

**UUID=d558d571-3171-4a8d-b71c-2ff04aca68c1**

**ONBOOT=yes**

**12.4.7 创建bond0子接口：**

**将网卡ens33和ens37创建为bond0的子接口：**

**添加网卡ens33：设备类型：bond-slave；连接名称：bond0-p1；master：bond0**

**[root@xuegod63 network-scripts]# nmcli connection add type bond-slave ifname ens33 con-name bond0-p1 master bond0**

**连接“bond0-p1”(fd76468d-a644-4f5f-855f-fbbc859f440a) 已成功添加。**

**添加网卡ens37：设备类型：bond-slave；连接名称：bond0-p2；master：bond0**

**[root@xuegod63 network-scripts]# nmcli connection add type bond-slave ifname ens37 con-name bond0-p2 master bond0**

**连接“bond0-p2”(39d983d3-1549-4fc0-bc4b-392d52a8e24e) 已成功添加。**

**[root@xuegod63 network-scripts]# cat ifcfg-bond0-p1**

**TYPE=Ethernet**

**NAME=bond0-p1**

**UUID=fd76468d-a644-4f5f-855f-fbbc859f440a**

**DEVICE=ens33**

**ONBOOT=yes**

**MASTER=bond0**

**SLAVE=yes**

**[root@xuegod63 network-scripts]# cat ifcfg-bond0-p2**

**TYPE=Ethernet**

**NAME=bond0-p2**

**UUID=39d983d3-1549-4fc0-bc4b-392d52a8e24e**

**DEVICE=ens37**

**ONBOOT=yes**

**MASTER=bond0**

**SLAVE=yes**

**12.4.8 激活bond0及其子接口：**

**查看当前已激活的网络接口：**

**[root@xuegod63 network-scripts]# nmcli connection show --active**

**名称 UUID 类型 设备**

**bond0 d558d571-3171-4a8d-b71c-2ff04aca68c1 bond bond0**

**virbr0 0c6bda20-636e-4555-a5eb-10ebffc43c38 bridge virbr0**

**有线连接 1 33136afd-2e13-310d-9c9d-cf4858545cf0 802-3-ethernet ens33**

**有线连接 2 3b9c673a-53eb-3f0c-91ae-1b63015ec734 802-3-ethernet ens37**

**如果bond0-p1、bond0-p2、bond0没有激活，可使用下面命令进行激活：**

**[root@xuegod63 network-scripts]# nmcli connection up bond0-p1**

**连接已成功激活（D-Bus 活动路径：/org/freedesktop/NetworkManager/ActiveConnection/9）**

**[root@xuegod63 network-scripts]# nmcli connection up bond0-p2**

**连接已成功激活（D-Bus 活动路径：/org/freedesktop/NetworkManager/ActiveConnection/10）**

**[root@xuegod63 network-scripts]# nmcli connection up bond0**

**成功激活（主服务器等待从服务器）连接（D-Bus 激活路径：/org/freedesktop/NetworkManager/ActiveConnection/11）**

**12.4.9 查看bond0当前状态：**

**[root@xuegod63 network-scripts]# cd /proc/net/bonding/**

**[root@xuegod63 bonding]# ls**

**bond0**

**[root@xuegod63 bonding]# cat bond0**

**Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)**

**Bonding Mode: fault-tolerance (active-backup)**

**Primary Slave: ens33 (primary\_reselect always)**

**Currently Active Slave: ens33 //当前所使用的接口**

**MII Status: up**

**MII Polling Interval (ms): 100**

**Up Delay (ms): 0**

**Down Delay (ms): 0**

**Slave Interface: ens33 //从接口**

**MII Status: up //状态为开启**

**Speed: 1000 Mbps**

**Duplex: full**

**Link Failure Count: 0**

**Permanent HW addr: 00:0c:29:72:33:db**

**Slave queue ID: 0**

**Slave Interface: ens37 //从接口**

**MII Status: up**

**Speed: 1000 Mbps**

**Duplex: full**

**Link Failure Count: 0**

**Permanent HW addr: 00:0c:29:72:33:e5**

**Slave queue ID: 0**

**12.4.10 bonding切换测试：**

**[root@xuegod63 bonding]# ping 192.168.1.1**

**PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.**

**64 bytes from 192.168.0.1: icmp\_seq=1 ttl=64 time=2.35 ms**

**64 bytes from 192.168.0.1: icmp\_seq=2 ttl=64 time=1.41 ms**

**64 bytes from 192.168.0.1: icmp\_seq=3 ttl=64 time=1.01 ms**

**[root@xuegod63 bonding]# nmcli connection down bond0-p1**

**成功取消激活连接 'bond0-p1'（D-Bus 活动路径：/org/freedesktop/NetworkManager/ActiveConnection/12）**

**[root@xuegod63 bonding]# ping 192.168.1.1 //停止了bond0-p1，仍然可以Ping通过**

**PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.**

**64 bytes from 192.168.0.1: icmp\_seq=1 ttl=64 time=0.922 ms**

**64 bytes from 192.168.0.1: icmp\_seq=2 ttl=64 time=0.604 ms**

**64 bytes from 192.168.0.1: icmp\_seq=3 ttl=64 time=0.551 ms**

**停止bond0-p1网卡，再次查看bond0状态：**

**[root@xuegod63 bonding]# cat bond0**

**Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)**

**Bonding Mode: fault-tolerance (active-backup)**

**Primary Slave: None**

**Currently Active Slave: ens37 //原来的接口为ens33，目前变为ens37**

**MII Status: up**

**MII Polling Interval (ms): 100**

**Up Delay (ms): 0**

**Down Delay (ms): 0**

**Slave Interface: ens37**

**MII Status: up**

**Speed: 1000 Mbps**

**Duplex: full**

**Link Failure Count: 0**

**Permanent HW addr: 00:0c:29:72:33:e5**

**Slave queue ID: 0**

**说明：**

**以上是直接停止bond0-p1接口，如果是直接在虚拟机中断开网卡的连接，那么在bond0信息中则会看到bond0-p1为down的信息**

**[root@xuegod63 bonding]# nmcli connection up bond0-p1 //再次启动bond0-p1接口**

**连接已成功激活（D-Bus 活动路径：/org/freedesktop/NetworkManager/ActiveConnection/15）**

**[root@xuegod63 bonding]# cat bond0**

**Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)**

**Bonding Mode: fault-tolerance (active-backup)**

**Primary Slave: ens33 (primary\_reselect always)**

**Currently Active Slave: ens33 //此时的ens33成为了当前使用的接口**

**MII Status: up**

**MII Polling Interval (ms): 100**

**Up Delay (ms): 0**

**Down Delay (ms): 0**

**Slave Interface: ens37**

**MII Status: up**

**Speed: 1000 Mbps**

**Duplex: full**

**Link Failure Count: 0**

**Permanent HW addr: 00:0c:29:72:33:e5**

**Slave queue ID: 0**

**Slave Interface: ens33**

**MII Status: up**

**Speed: 1000 Mbps**

**Duplex: full**

**Link Failure Count: 0**

**Permanent HW addr: 00:0c:29:72:33:db**

**Slave queue ID: 0**

**12.4.11 添加网关：**

**[root@xuegod63 network-scripts]# vim ifcfg-bond0**

**GATEWAY=192.168.0.1**

**[root@xuegod63 network-scripts]# service network restart**

**Restarting network (via systemctl): [ 确定 ]**

**[root@xuegod63 network-scripts]# route -n**

**Kernel IP routing table**

**Destination Gateway Genmask Flags Metric Ref Use Iface**

**0.0.0.0 192.168.0.1 0.0.0.0 UG 300 0 0 bond0**

**192.168.0.0 0.0.0.0 255.255.255.0 U 300 0 0 bond0**

**192.168.122.0 0.0.0.0 255.255.255.0 U 0 0 0 virbr0**

**12.4.12 添加DNS：**

**[root@xuegod63 bonding]# vim /etc/resolv.conf**

**nameserver 114.114.114.114**

**[root@xuegod63 network-scripts]# curl -I www.baidu.com**

**HTTP/1.1 200 OK**

**Server: bfe/1.0.8.18**

**Date: Thu, 18 Jan 2018 17:00:30 GMT**

**Content-Type: text/html**

**Content-Length: 277**

**Last-Modified: Mon, 13 Jun 2016 02:50:04 GMT**

**Connection: Keep-Alive**

**ETag: "575e1f5c-115"**

**Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform**

**Pragma: no-cache**

**Accept-Ranges: bytes**

**12.5 网络内核相关参数调优**

**12.5.1 TCP 连接三次握手相关**

**Client ------------------------- Server**

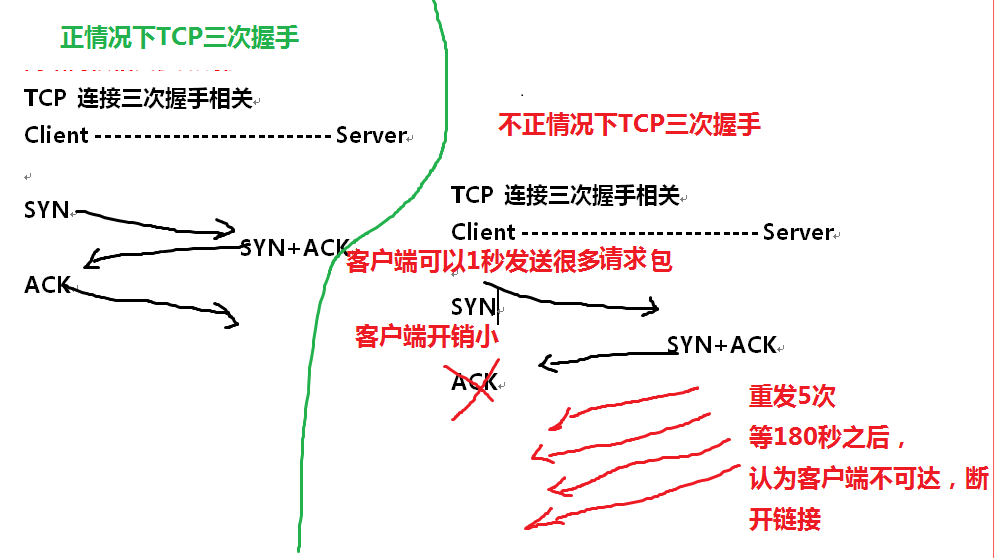
**SYN**

**SYN+ACK**

**ACK**

1. **抵御SYN  
    SYN攻击是利用TCP/IP协议3次握手的原理，发送大量的建立连接的网络包，但不实际建立连接，最终导致被攻击服务器的网络队列被占满，无法被正常用户访问。**

**原理图：**



**SYN Flood是当前最流行的DoS（拒绝服务攻击）与DDoS（分布式拒绝服务攻击）的方式之一，这是一种利用TCP协议缺陷，发送大量伪造的TCP连接请求，常用假冒的IP或IP段发来海量的请求连接的第一个握手包（SYN包），被攻击服务器回应第二个握手包（SYN+ACK包），因为对方是假冒IP，对方永远收不到包且不会回应第三个握手包。导致被攻击服务器保持大量SYN\_RECV状态的“半连接”，并且会重试默认5次回应第二个握手包，塞满TCP等待连接队列，资源耗尽（CPU满负荷或内存不足），让正常的业务请求连接不进来。**

**解决：**

**[root@xuegod63 ~]# vim /etc/sysctl.conf #在文件最后添加以下内容**

**net.ipv4.tcp\_synack\_retries = 0**

**net.ipv4.tcp\_syn\_retries = 1**

**net.ipv4.tcp\_max\_syn\_backlog = 20480**

**net.ipv4.tcp\_syncookies = 1**

**net.ipv4.tcp\_tw\_reuse = 1**

**net.ipv4.tcp\_tw\_recycle = 1**

**net.ipv4.tcp\_fin\_timeout = 10**

**fs.file-max = 819200**

**net.core.somaxconn = 65536**

**net.core.rmem\_max = 1024123000**

**net.core.wmem\_max = 16777126**

**net.core.netdev\_max\_backlog = 165536**

**net.ipv4.ip\_local\_port\_range = 10000 65535**

**注：每台服务器上线之前，都应该配置以上内核参数。**

**最重要参数：**

**注释：**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_synack\_retries #最关键参数，默认为5，修改为0 表示不要重发**

**net.ipv4.tcp\_synack\_retries = 0**

**#表示回应第二个握手包（SYN+ACK包）给客户端IP后，如果收不到第三次握手包（ACK包）后，不进行重试，加快回收“半连接”，不要耗光资源。**

**#作为服务端。回应时，如果连接失败，达到对应的失败数后，停止发送synack包**

**第一个参数tcp\_synack\_retries = 0是关键，表示回应第二个握手包（SYN+ACK包）给客户端IP后，如果收不到第三次握手包（ACK包）后，不进行重试，加快回收“半连接”，不要耗光资源。**

**不修改这个参数，模拟攻击，10秒后被攻击的80端口即无法服务，机器难以ssh登录； 用命令netstat -na |grep SYN\_RECV检测“半连接”hold住180秒；**

**修改这个参数为0，再模拟攻击，持续10分钟后被攻击的80端口都可以服务，响应稍慢些而已，只是ssh有时也登录不上；检测“半连接”只hold住3秒即释放掉。**

**修改这个参数为0的副作用：网络状况很差时，如果对方没收到第二个握手包，可能连接服务器失败，但对于一般网站，用户刷新一次页面即可。这些可以在高峰期或网络状况不好时tcpdump抓包验证下。**

**根据以前的抓包经验，这种情况很少，但为了保险起见，可以只在被tcp洪水攻击时临时启用这个参数。**

**tcp\_synack\_retries默认为5，表示重发5次，每次等待30~40秒，即“半连接”默认hold住大约180秒。**

**我们之所以可以把tcp\_synack\_retries改为0，因为客户端还有tcp\_syn\_retries参数，默认是5，即使服务器端没有重发SYN+ACK包，客户端也会重发SYN握手包。**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_syn\_retries**

**1**

**#tcp\_syn\_retries参数，默认是5，当没有收到服务器端的SYN+ACK包时，客户端重发SYN握手包的次数，注意：在rhel 7中，此项不能调为0**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_max\_syn\_backlog**

**20480**

**##半连接队列长度，增加SYN队列长度到20480：加大SYN队列长度可以容纳更多等待连接的网络连接数，具体多少数值受限于内存。**

**接下来辅助参数：**

**#系统允许的文件句柄的最大数目，因为连接需要占用文件句柄**

**fs.file-max = 819200**

**#用来应对突发的大并发connect 请求**

**net.core.somaxconn = 65536**

**#最大的TCP 数据接收缓冲（字节）**

**net.core.rmem\_max = 1024123000**

**#最大的TCP 数据发送缓冲（字节）**

**net.core.wmem\_max = 16777126**

**#网络设备接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目**

**net.core.netdev\_max\_backlog = 165536**

**#本机主动连接其他机器时的端口分配范围，比如说，在vsftpd主动模式会用到**

**net.ipv4.ip\_local\_port\_range = 10000 65535**

**注：如果只是开启22端口，是不会使用到ip\_local\_port\_range这个功能**

**[root@xuegod63 ~]# netstat -antup | grep :22**

**tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1993/sshd**

**tcp 0 0 192.168.1.63:22 192.168.1.23:51855 ESTABLISHED 9316/sshd**

**tcp 0 0 192.168.1.63:22 192.168.1.23:51861 ESTABLISHED 10878/sshd**

**为了处理大量连接，还需改到另外两个参数：  
限制用户资源配置文件：/etc/security/limits.conf**

**[root@xuegod63 ~]#vim /etc/security/limits.conf #在最添加：  
\* soft nofile 1024000**

**\* hard nofile 1024000**

**例2：nproc #用户可以打开的最大进程数**

**[root@xuegod63 ~]# vim /etc/security/limits.d/90-nproc.conf #RHEL6 必须这个文件中配置**

**改：**

**\* soft nproc 10240**

**为：**

**\* soft nproc 66666**

**\* hard nproc 66666**

**[root@xuegod63 ~]# reboot #最好重启一下**

**次要辅助参数，以上还无法解决syn洪水攻击，把以下内核参数关闭：**

**注意，以下参数面对外网时，不要打开。因为副作用很明显。**

**#当出现 半连接 队列溢出时向对方发送syncookies，调大 半连接 队列后没必要**

**net.ipv4.tcp\_syncookies = 0**

**#TIME\_WAIT状态的连接重用功能**

**net.ipv4.tcp\_tw\_reuse = 0**

**#时间戳选项，与前面net.ipv4.tcp\_tw\_reuse参数配合**

**net.ipv4.tcp\_timestamps = 0**

**#TIME\_WAIT状态的连接回收功能**

**net.ipv4.tcp\_tw\_recycle = 0**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_syncookies**

**1**

**#表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；**

**[root@xuegod63 ~]# cat /proc/sys/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，现在开启，改为1**

**[root@xuegod63 ~]# cat /proc/sys/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。现在改为1，表示开启**

**[root@xuegod63 ~]# cat /proc/sys/net/ipv4/tcp\_fin\_timeout**

**10**

**#默认值是 60，对于本端断开的socket连接，TCP保持在FIN\_WAIT\_2状态的时间。**

**调整MTU最大传输单元：**

**[root@xuegod63 ~]# ifconfig ens33 mtu 9000**

**MTU，即Maximum Transmission Unit(最大传输单元)，此值设定TCP/IP协议传输数据报时的最大传输单元。**

**系统与ISP之间MTU的不符就会直接导致数据在网络传输过程中不断地进行分包、组包，浪费了宝贵的传输时间，也严重影响了宽带的工作效率。**

**总结：**

**12.1 有关CPU的调优**

**12.2 有关内存的调优**

**12.3 有关I/O的调优**

**12.4 有关网络的调优**

**12.5 有关内核参数的调优**