Linux 云计算集群架构师课程

北京学神科技有限公司

学神 IT 教育：从零基础到实战，从入门到精通！

联系方式：

学神 IT 教育官方网站: http://www.xuegod.cn

学神 IT 教育-PHP 技术交流 QQ 群: 196954821

学神 IT 教育-Linux 技术交流 QQ 群: 57873866

==========

咨询 QQ: 1514460659

学习一个服务的过程：

1、 此服务器的概述：名字，功能，特点，端口号
2、 安装
3、 配置文件的位置
4、 服务启动关闭脚本，查看端口
5、 此服务的使用方法
6、 修改配置文件，实战举例
7、 排错（从下到上，从内到外）

本节所讲内容：

- 有关 CPU 的调优
- 有关内存的调优
- 有关 I/O 的调优
- 有关网络的调优
- 有关内核参数的调优

关于 CPU 中央处理器调优

CPU 处理数据的方式：

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

查看内核一秒钟中断 CPU 次数：

```
[root@xuegod63 ~]# grep HZ /boot/config-2.6.32-220.el6.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-2.6.32-220.el6.x86_64 是编译内核的参数文件

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 b.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
24219 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
24219 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 24219

24219: old priority -20, new priority -20

[root@xuegod63 ~]# renice -n 20 24219

24219: old priority -20, new priority 19

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-ng-2.17.2-12.4.el6.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

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：

8 核 CPU ID:	7	6	5	4	3	2	1	0
对应的 10 十进制数为:	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 。 二进制表示为： 0101=2+8=10
```

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 # 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快速运行

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 #本机为单核 CPU，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 2182    7 6    83   4
3 1 207740 90124 89628 180984 0 0 1176 0       551 2219    2 7    91   0
```

1,上下文切换数目高于中断数目,说明当前系统中运行着大量的线程 ,kernel 中相当数量的时间都开销在线程的”上下文切换“。

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量的可运行状态线程在等待执行.

总结：说明

内存调优相关内容：

实例 1：安装完系统后，测试内存

[root@xuegod63 ~]# rpm -ivh /mnt/Packages/memtest86+-4.10-2.el6.x86_64.rpm

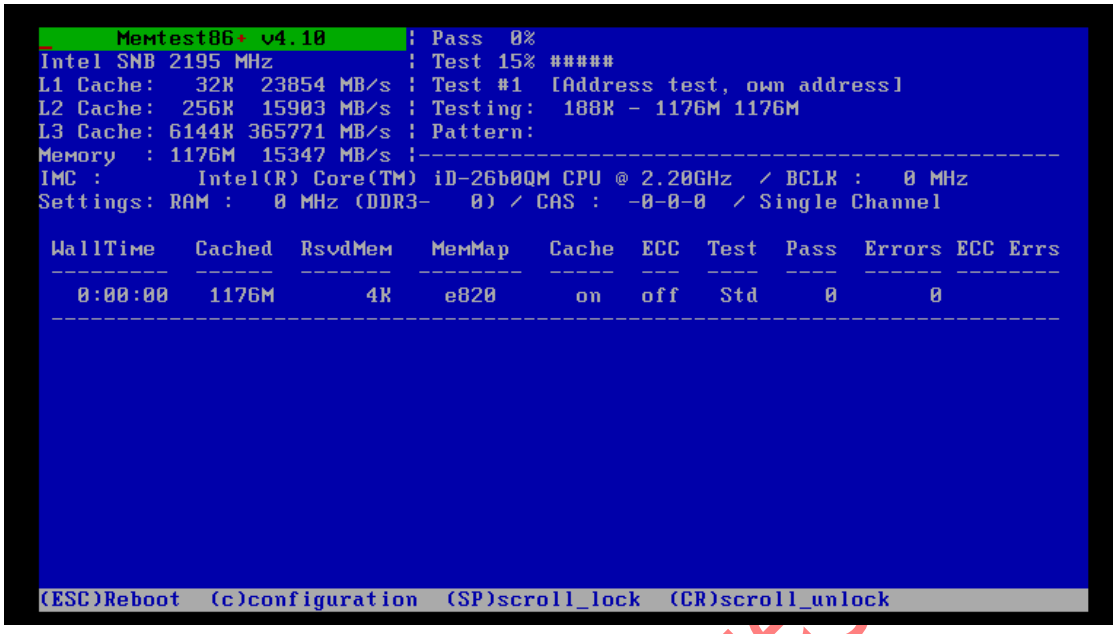
[root@xuegod63 ~]# memtest-setup

Setup complete.

[root@xuegod63 ~]# vim /etc/grub.conf #多了一个启动项

重启后，在 grub 启动项中选择：

Memtest86+ (4.10)　这个项就可以了



4. 关于缓存

BUFFER　索引缓存　缓存 写时用，先写入到内存

CACHE　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 -m

终端 2：find /

终端 1:free　-m　#查看　buffer 增长

CACHE：页缓存，　内存页是内存中的最小存储单位，一页尺寸 4kB

对象文件系统 块 block 1kB 2kB 4kB

　　扇区 sectors 512b

手动清空 buffer+cache ：

[root@xuegod63 ~]# cat /proc/sys/vm/drop_caches　#默认是 0

0

[root@xuegod63 ~]# free -m

	total	used	free	shared	buffers	cached
Mem:	1137	783	353	0	67	411
-/+ buffers/cache:		303	833			
Swap:	999	0	999			

[root@apache ~]# sync　# 把内存中的数据写入磁盘

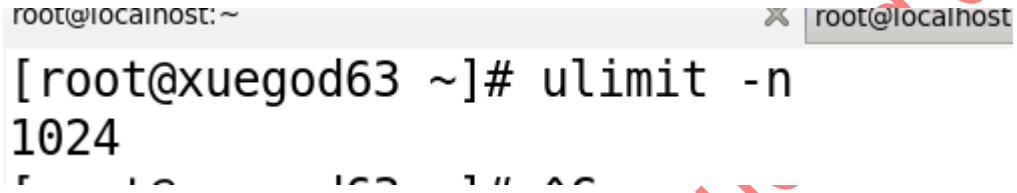
[root@xuegod63 ~]# echo 1 > /proc/sys/vm/drop_caches

```
[root@xuegod63 ~]# free -m
             total       used       free     shared    buffers     cached
Mem:          1137        367        770          0          0         67
-/+ buffers/cache:        299        838
Swap:          999          0        999
```

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

I/O调优 相关内容

1、设置一个进程可以打开的文件数



测试：

```
[root@xuegod63 ~]# service httpd restart
Stopping httpd:                                            [FAILED]
Starting httpd:                                            [  OK  ]
[root@xuegod63 ~]# echo 111 > /var/www/html/index.html
[root@xuegod63 ~]# ab -n 2000  -c 2000 http://192.168.1.63/index.html
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.63 (be patient)
socket: Too many open files (24)
```

解决：

限制用户资源配置文件：/etc/security/limits.conf

```
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。

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 mk    #以普通用户登录，测试
[root@xuegod63 ~]# su - mk
[mkkk@xuegod63 ~]$ ulimit -n
```

1024000
方法二：#临时修改
[root@xuegod63 ~]# ulimit -n 10000
[root@xuegod63 ~]# ulimit -n
10000

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：

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

[root@xuegod63 ~]# dd if=/dev/zero of=/test.dbf bs=8K count=30000
3000+0 records in
3000+0 records out
24576000 bytes (25 MB) copied, 5.13755 s, 4.8 MB/s
生成 25M 的一个文件，IO 写的速度约为 4.8 MB/s
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
例1： 测试dd 命令使用时间和开销
[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
user 0m0.002s
sys 0m0.770s
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。
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。

测试硬盘速度：
测试硬盘速度：
[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.

四、网相关调优：

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。

Trunking（链路聚集）

二、技术分类

1. 负载均衡

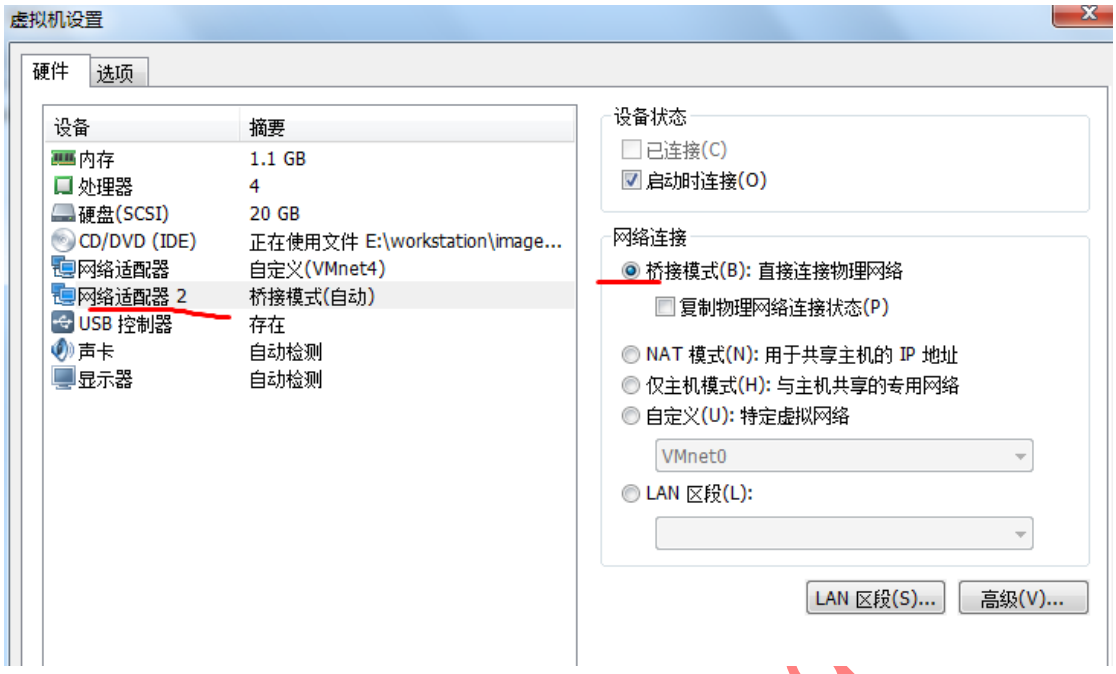
2. 网络冗余

实战：配置多网卡绑定技术

配置环境：

xuegod63 配置两双网卡，网卡eth0和eth1都桥接

添加网卡：



配置：

查看是否加载了bond模块：

查看是否加载了bond模块：

[root@xuegod63 network-scripts]# modinfo bonding| grep .ko

filename:

/lib/modules/2.6.32-220.el6.x86_64/kernel/drivers/net/bonding/bonding.ko

[root@xuegod63 network-scripts]# modprobe bonding　#加载模块

[root@xuegod63 ~]# lsmod | grep bond

bonding　125610　0

ipv6　322029　158 bonding,ip6t_REJECT,nf_conntrack_ipv6,nf_defrag_ipv6

开机自动加载模块到内核：

[root@xuegod63 ~]# echo 'modprobe bonding &> /dev/null' >> /etc/rc.local

[root@xuegod63 ~]# vim　/etc/modprobe.conf　#创建此配置文件，并写入以下内容

编辑模块载入配置文件，让系统支持bonding

alias bond0 bonding

options bonding miimon=100 mode=balance-rr

说明：

a. mode=balance-rr 或mode=0 这里我采用这种rr轮循模式， 此模式提供负载平衡和容错。 此模式下所以网卡共用一个MAC地址，后面结果会有验证！

b. miimon是用来进行链路监测的，如:miimon=100，那么系统每100ms监测一次链路连接状态，如果有一条线路不通就转入另一条线路；

c. bonding只能提供链路监测，即从主机到交换机的链路是否接通。如果只是交换机对外的链路down掉

了，而交换机本身并没有故障，那么bonding会认为链路没有问题而继续使用。

创建bond0设置配置文件

```
[root@xuegod63 ~]# vim /etc/sysconfig/network-scripts/ifcfg-bond0    #写入以下内容
DEVICE=bond0
IPADDR=192.168.1.63
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.1
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

创建eth0配置文件：

```
[root@xuegod63 ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0   #写入以下内容
DEVICE=eth0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

创建eth1配置文件：

```
[root@xuegod63 ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth1     #写入以下内容
DEVICE=eth1
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

重启网卡：

```
[root@xuegod63 ~]# service network restart
Shutting down interface eth0:                              [  OK  ]
Shutting down loopback interface:                          [  OK  ]
Bringing up loopback interface:                            [  OK  ]
Bringing up interface bond0:                               [  OK  ]
```

查看：

```
[root@xuegod63 ~]# ifconfig     #注此时MAC地址都一样
bond0     Link encap:Ethernet  HWaddr 00:0C:29:12:EC:1E
          inet addr:192.168.1.63  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe12:ec1e/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:751 errors:0 dropped:0 overruns:0 frame:0
```

TX packets:445 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:73353 (71.6 KiB) TX bytes:59783 (58.3 KiB)

eth0 Link encap:Ethernet HWaddr 00:0C:29:12:EC:1E
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
RX packets:743 errors:0 dropped:0 overruns:0 frame:0
TX packets:432 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:72050 (70.3 KiB) TX bytes:57101 (55.7 KiB)

eth1 Link encap:Ethernet HWaddr 00:0C:29:12:EC:1E
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1

测试：

```
[root@xuegod63 ~]#  ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=2.89 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=1.82 ms
```

[root@xuegod63 network-scripts]# route -n #查看网关

```
[root@xuegod63 ~]#  route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 bond0
169.254.0.0     0.0.0.0         255.255.0.0     U     1004   0        0 bond0
0.0.0.0         192.168.1.1     0.0.0.0         UG    0      0        0 bond0
```

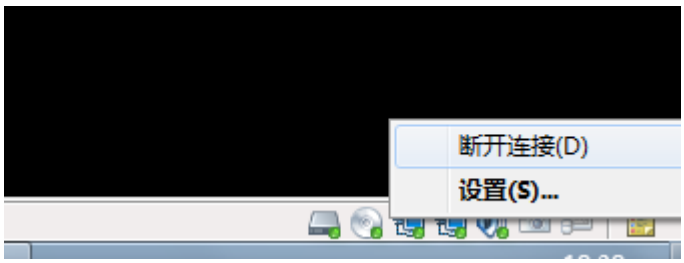
[root@xuegod63 network-scripts]# cat /etc/resolv.conf #查看DNS

```
# Generated by NetworkManager
nameserver 192.168.1.1
search localhost
```

测试高可用：

1、[root@xuegod63 ~]# ping 192.168.1.1 #打开终端后，持续ping 网关

2、此时可以断开虚拟机eth1网络连接，查看ping是否有丢包：



3、断开后，发现通信正常，没有问题。说明多网卡绑定成功

mode=0：可实现网络负载均衡和网络冗余，采用平衡轮循策略(balance-rr)。
此模式的特点：
a. 所以网卡都工作，传输数据包顺序是依次传输，也就是说第1个包经过eth0，下一个包就经过eth1，一直循环下去，直到最后一个包传输完毕。
b. 此模式对于同一连接从不同的接口发出的包，中途传输过程中再经过不同的链接，在客户端很有可能会出现数据包无序到达的问题，而无序到达的数据包需要重新要求被发送，这样网络的吞吐量就会下降。
c. 与网卡相连的交换必须做特殊配置（ 这两个端口应该采取聚合方式），因为做bonding的这两块网卡是使用同一个MAC地址

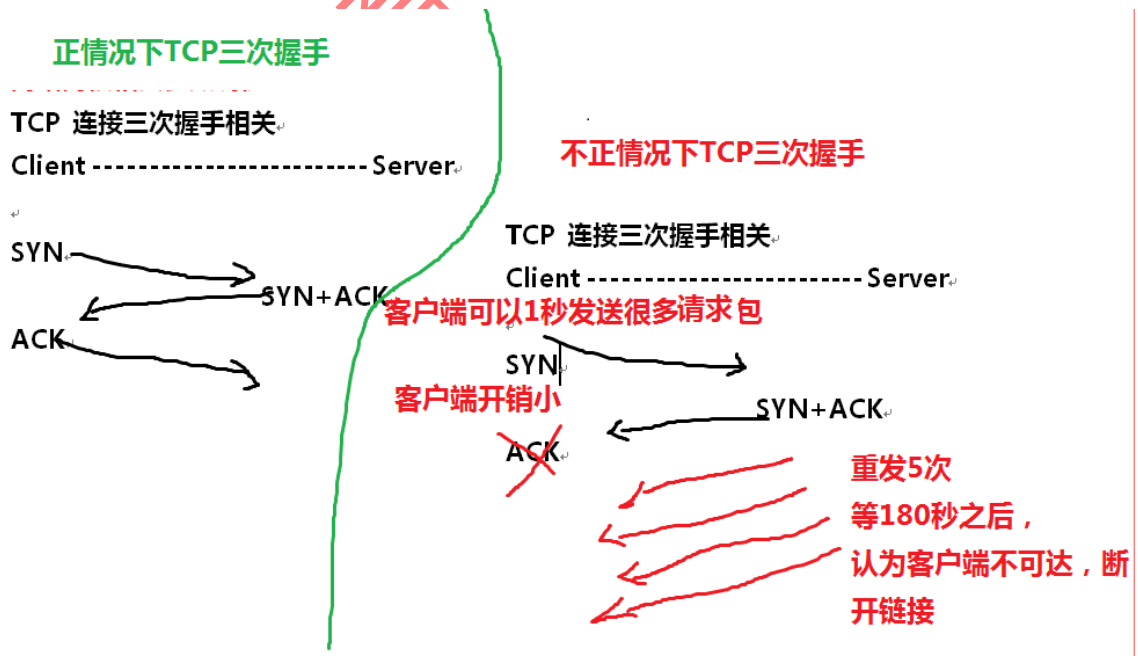
网络内核相关参数调优
TCP 连接三次握手相关
Client ---------- Server

SYN
SYN+ACK
ACK

1. 抵御SYN 洪水攻击

1. 抵御SYN洪水攻击
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SYN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号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 = 0
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 = 16777216
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

1

#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
0
#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。

[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 数据发送缓冲（字节）
net.core.wmem_max = 16777216
#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，还需改大另外两个参数：
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：

意，以下参数面对外网时，不要打开。因为副作用很明显。

```
[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接收溢出的SYN连接，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tcp链接重用。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
30
#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。
```

内网提速之巨帧Jumbo Frame

```
[root@xuegod63 ~]# ifconfig eth0
eth0        Link encap:Ethernet  HWaddr 00:0C:29:12:EC:1E
            inet addr:192.168.1.63  Bcast:192.168.1.255  Mask:255.255.255.0
            inet6 addr: fe80::20c:29ff:fe12:ec1e/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

实例：调大MTU

```
[root@xuegod63 ~]# ifconfig eth0 mtu 9000
[root@xuegod63 ~]# ifconfig eth0
eth0        Link encap:Ethernet  HWaddr 00:0C:29:12:EC:1E
            inet addr:192.168.1.63  Bcast:192.168.1.255  Mask:255.255.255.0
            inet6 addr: fe80::20c:29ff:fe12:ec1e/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
```

注：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。