性能分析学习笔记

Author:Geekwolf
Blog: http://www.simlinux.com
Weibo: http://weibo.com/geekwolf
注:本文只是大概总结了下性能分析牵涉部分问题,以后会逐渐完善丰富,也算最近大半个月的学习成果吧~

一、单位换算

1K=10^3	Bytes	1Ki=2^10	Bytes
1M=10^6	Bytes	1Mi=2^20	Bytes
1G=10^9	Bytes	1Gi=2^30	Bytes
1T=10^12	Bytes	1Ti=2^40	Bytes
df -H		df -h	

1B(ytes)=8b(it)
具体到某个命令的显示单位是什么最好通过 man 手册查看确认

二、性能分析工具

1、分析工具

vmstat :内存使用情况分析　(procps 软件包还有 top free 命令)
iostat :磁盘 IO 使用情况
mpstat :CPU 相关统计信息
sar :综合系统报告工具　　|
awk :对数据进行格式化
主要的性能因素 cpu、mem、磁盘 io、网络相关

CPU 信息:

LANG=C　（指定此关键变量，统计数据的时间 24 小时计）
mpstat　N M (N:统计间隔 N M:统计 M 次)
-P :指定哪个核
cat /proc/cpuinfo
lscpu 查看 cpu 的架构(util-linux-ng 包)

内存信息:

free -m(m 以 1000 为单位，查看帮助)
[root@cloudadmin ~]# free -m

	total	used	free	shared	buffers	cached
Mem:	7862	7172	689	0	333	1534
-/+ buffers/cache:		5303	2558			
Swap:	9536	134	9402			

实际使用的内存是 5303,7172-5303=buffers+cached

cat /proc/meminfo
Inactive : 表示不经常被访问的内存空间

vmstat
b: 不可中断睡眠一般是 IO 事件的进程数量
磁盘 IO 信息：
iostat 1 10
man iostat 查看相关参数的单位

网络信息：
netstat –ntlp 显示 tcp 的监听端口
netstat –ntlp 显示 tcp udp 的监听端口
netstat –r 显示路由表
netstat –rn 显示路由表不做名称解析（较快）
netstat –an 显示本地的连接

综合命令 sar：

脚本命令说明
/usr/lib64/sa/sa1 负责收集并存储每天系统动态信息到一个二进制的文件中。它是通过计划任务工具 cron 来运行，是为 sadc 所设计的程序前端程序
/usr/lib64/sa/sa2 负责把每天的系统活跃性息写入总结性的报告中。它是为 sar 所设计的前端 ，要通过 cron 来调用,参数-A 指定了从二进制文件中提取哪些数据存储到文本文件中
/usr/lib64/sa/sadc 是系统动态数据收集工具，收集的数据被写一个二进制的文件中，它被用作 sar 工具的后端
/usr/bin/sadf 显示被 sar 通过多种格式收集的数据
/usr/bin/sar 负责收集、报告并存储系统活跃的信息

详细信息参考 man
sar 的命令详解 http://lovesoo.org/linux-sar-command-detailed.html

影响性能因素	评判标准		
	好	坏	糟糕
CPU	user% + sys%< 70%	user% + sys%= 85%	user% + sys% >=90%
内存	Swap In（si）=0 Swap Out（so）=0	Per CPU with 10 page/s	More Swap In & Swap Out
磁盘	iowait % < 20%	iowait % =35%	iowait % >= 50%

1.当前状态的统计信息
sar 1 100
sar –f /var/log/sa/sa22　　查看 22 号 sa 信息文件

2.计划任务记录历史统计信息
默认安装好 sysstat 之后会生成/etc/cron.d/sysstat

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
# 0 * * * * root /usr/lib64/sa/sa1 600 6 &
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib64/sa/sa2 -A
```

更改性能搜集策略 修改文件/usr/lib64/sa/sa1 （42 44 行）
exec ${ENDIR}/sadc -F -L ${SADC_OPTIONS} 1 1 –
exec ${ENDIR}/sadc -F -L ${SADC_OPTIONS} $* -
可以添加 sar 的性能参数比如搜集电源相关的信息
exec ${ENDIR}/sadc -S POWE -F -L ${SADC_OPTIONS} 1 1 –
exec ${ENDIR}/sadc -S POWER -F -L ${SADC_OPTIONS} $* -

sar 配置文件/etc/sysconfig/sysstat
HISTORY=28 保存多少天
service sysstat start
chkconfig sysstat on

AWK 命令：

awk -F : '{print $1}' /etc/passwd　　查看 passwd 的第一列
awk -F : 'NR<=10{print NR,$1}' /etc/passwd 打印出前十行记录的行号和第一列
awk -F : 'NR>=10 && NR <=15{print NR,$1}' /etc/passwd
awk -F : '!NR==1{print NR,$1}' /etc/passwd

awk -F : '{print $0,NF}' /etc/passwd　　$0 表示所有，NF 表示以:为间隔符时有多少个字段
awk -F : '{print $0,$NF}' /etc/passwd　　$NF 表示以:为间隔符时最后一个字段值 $(NF-1)最后一列的前一列
awk '/^22/{print $0}' /etc/passwd
awk -F : '$1 ~ /^r/{print $0}' /etc/passwd
awk '/^[^a-zA-Z]+$/{print $0}' 1.txt　　表示匹配行首到行尾没有大小写字母的记录,+表示匹配前一个字符一次或者以上,$行尾

grep 'r..k' /etc/passwd　　点点表示中间任意两个字符
grep 'r.k' /etc/passwd　　点表示中间任意单个字符
grep '^r[abde]' /etc/passwd　表示开头匹配 ra 或 rb 或 rd 或 re
grep '^r[a-z]' /etc/passwd
grep '^r[a-z][A-Z]' /etc/passwd　表示 r 后面有两个字符，一个小写，一个大写
grep '^r[a-zA-Z]' /etc/passwd　[ ]表示一个字符
grep '^r[a-zA-Z0-9]' /etc/passwd
grep '^r[^a-z]' /etc/passwd　　[]中的^表示不是,非
grep '^ar*' /etc/passwd　　*表示匹配*前面字符的任意个
grep '^r.*bash$' /etc/passwd 表四匹配以 r 开始以 bash 结尾的任意字符

简单正则:
awk 使用正则时要放进/ /
^abc　　abc 开头
bash$　　bash 结尾
*　　匹配前面的子表达式零次或多次(大于等于 0 次)
+　　匹配前面的子表达式一次或多次(大于等于 1 次）
?　　匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。?等价于{0,1}。
{n}　　n 是一个非负整数。匹配确定的 n 次
{n,}　　n 是一个非负整数。至少匹配 n 次
{n,m}　　m 和 n 均为非负整数，其中 n<=m。最少匹配 n 次且最多匹配 m 次
.点　　匹配除“\n”之外的任何单个字符
x|y　　匹配 x 或 y

^$　　空行
^ $　　有一个空格的行
^ +$　　表示至少有一个空格
\bgeekwolf\b 精确匹配 geekwolf　\b 匹配一个单词边界
\B　　匹配非单词边界，“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”
\d　　匹配一位数字[0-9],如：0\d\d-\d\d\d\d\d\d\d\d 以 0 开头,然后 2 个数字,然后一个连字号”-”,最后是 8 个数字,简写：0\d{2}-\d{8}
\D　　非数字[^0-9]
\s　　匹配任何空白字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]

\S	匹配任何非空白字符。等价于[^ \f\n\r\t\v]
\w	匹配字母或数字或下划线或汉字。等价于“[A-Za-z0-9_]”
\W	匹配任何非单词字符。等价于“[^A-Za-z0-9_]”

http://www.cnblogs.com/deerchao/archive/2006/08/24/zhengzhe30fengzhongjiaocheng.html#mission

2、绘图工具

gnuplot 绘图工具

实例：sar –q 采集数据获取 1m 5m 15m 负载数据并用 gnuplot 绘图
sar –q 1 20 > file.txt
awk '/^[^a-zA-Z]+$/{print $1,$(NF-2),$(NF-1),$NF}' file.txt >format.txt
yum –y install gnuplot
vim load.gnuplot
set xdata time 设置 x 轴的格式是时间格式
set timefmt “%H:%M:%S” 时间格式是%H:%M%S，根据具体时间格式而定
set format x “%H:%M:%S” 设置横着的时间格式
set xlabel “Time” 设置 x 轴的名字是 Time
set ylabel “load average” 设置 y 轴的名字是 load average
set terminal png size 1024,768 设置输出的图片格式是 png 分辨率为 1024x768
set output “/tmp/file.png” 设置输出图片的位置
plot “/root/format.txt” using 1:2 title “1-load” with lines,”/root/format.txt” using 1:3 titile “5-load” with lines,”/root/format.txt” using 1:4 title “15-load” with lines
使用 format.txt 文件的第 1 列和第 2 列,名字是 1-load，线绘制

gnuplot load.gnuplot 会在/tmp 下生成 file.png 图片，使用 eog 命令可以打开图片
或者 gnuplot –persist load.gnuplot(如果不设置 output,会直接打开)
其他 gnuplot 参数参考 http://www.gnuplot.info/docs_4.6/gnuplot.pdf

rrdtool 绘图工具

RRDtool 是指 Round Robin Database 工具，Round robin 是一种处理定量数据、以及当前元素指针的技术。想象一个周边标有点的圆环－－这些点就是时间存储的位置。从圆心画一条到圆周的某个点的箭头－－这就是指针。就像我们在一个圆环上一样，没有起点和终点，你可以一直往下走下去。过来一段时间，所有可用的位置都会被用过，该循环过程会自动重用原来的位置。这样，数据集不会增大，并且不需要维护。RRDtool 处理 RRD 数据库。它用向 RRD 数据库存储数据、从 RRD 数据库中提取数据(来源百度问答)。为了方便理解其工作原理，本人做了一个 rrdtool 存储结构图：

1、 **DS** 用于定义 Data Soure 。也就是用于存放脚本的结果的变量名(DSN)。如 eth0_in ,eth0_out, lo_in , lo_out 。DSN 从 1-19 个字符，必须是 0-9,a-z,A-Z 。

2、 **DST** 的选择是十分重要的,如果选错了 DST ,即使你的脚本取的数据是对的,放入 RRDtool 后也是错误的,更不用提画出来的图是否有意义了。

DST 描述:

A) COUNTER:必须是递增的,除非是计数器溢出(overflows)。在这种情况下,RRDtool 会自动修改收到的值。例如网络接口流量、收到的 packets 数量都属于这一类型。

B) DERIVE: 和 COUNTER 类似。但可以是递增,也可以递减,或者一会增加一会儿减少。

C) ABSOLUTE:ABSOLUTE 比较特殊,它每次都假定前一个 interval 的值是 0,再计算平均值。

D) GAUGE:GAGUE 和上面三种不同,它没有"平均"的概念,RRDtool 收到值之后字节存入 RRA 中。

E) COMPUTE:COMPUTE 比较特殊,它并不接受输入,它的定义是一个表达式,能够引用其他 DS 并自动计算出某个值。

这五种类型有什么区别？不防测试一下便知。

3、 **CDP**:Consolidation Data Point 。RRDtool 使用多个 PDP 合并为(计算出)一个 CDP。也就是执行上面的 CF 操作后的结果。这个值就是存入 RRA 文件的数据，绘图时使用的也是这些数据。

4、 **CF** 就是 Consolidation Function 的缩写。也就是合并(统计)功能。有 AVERAGE、MAX、MIN、LAST 四种分别表示对多个 PDP 进行取平均、取最大值、取最小值、取当前值四种类型。具体作用等到 update 操作时再说。

5、 **PDP**:Primary Data Point。正常情况下每个 interval RRDtool 都会收到一个值;RRDtool 在收到脚本给来的值后会计算出另外一个值(例如平均值),这个值就是 PDP;这个值代表的一般是"xxx/秒"的含义。注意,该值不一定等于 RRDtool 收到的那个值。除非是 GAUGE,可以看下面的例子就知道了。

6、 **DST** 就是 Data Source Type 的意思。有 COUNTER、GUAGE、DERIVE、ABSOLUTE、COMPUTE5 种。由于网卡流量属于计数器型，所以这里应该为 COUNTER。

7、 **RRD** 用于指定数据如何存放。我们可以把一个 RRA 看成一个表，各保存不同 interval 的统计结果

8、 **resolution** 就是每个 RRA 中两个 CDP 相隔的时间

思路：create →update 数据到 rrd 文件 →graph 绘图

yum –y install rrdtool
rrdtool -h 查看帮助
例子：绘制系统 1 分钟负载图

```
 rrdtool create /tmp/loadave.rrd \
 --step=10\
 DS:1_min_load_average:GAUGE:30:0:U \
 RRA:AVERAGE:0.5:2:60\

 rrdtool update /tmp/loadave.rrd \
$(date +%s):$(uptime |awk '{print $(NF-2) $(NF-1)}'|sed 's/,//g')\

 rrdtool graph /var/www/html/load_average_hour.png\
 -X 0 --start=$(date --date=-1hour +%s) --end=$(date +%s) \
DEF:1load=/tmp/loadave.rrd:1_min_load_average:AVERAGE\
 LINE1:1load#000000:"1-min-load"\
```

练习题：绘制系统 1 5 15 分钟负载图
要求：

1. rrd 数据库存储过去 60m 的每 1m 的平均值，每分钟一个 PDP 数据点（1m 一个 CDP，共 60 个合并数据点[CDP]）[RRD 1:60]
2. 30m 一个 CDP(平均值)，存储过去一周的数据[RRD 30:336]
3. 写脚本能按照 1 2 的要求将数据每 1m 写入 rrd
4. 绘制过去一小时和过去一周的图创建 2 个 crontab，一个每 10m 绘制过去一小时的图，一个每小时绘制过去一天的图
5. 并将图放在/var/www/html 名称为:load_avg_hourly.png loadavg_daily.png,可以通过 http 访问

操作步骤：
1.创建 rrd
rrdtool create /tmp/loadavg.rrd --step=60 --start=$(date +%s) DS:loadavg1:GAUGE:60:0:U DS:loadavg5:GAUGE:60:0:U DS:loadavg15:GAUGE:60:0:U RRA:AVERAGE:0.5:1:60 RRA:AVERAGE:0.5:30:336
2.更新数据
vim /root/update_loadavg.sh

```
#!/bin/bash
rrdtool update /tmp/loadavg.rrd $(date +%s):$(uptime|awk '{print $(NF-2),$(NF-1),$NF}'|sed 's/,
```

/:/g')

chmod +x /root/update_loadavg.sh
crontab –e
*/1 * * * * /root/update_loadavg.sh

加压：
#!/bin/bash
while :
do
 :
done

查询 rrd 是有数据
rrdtool lastupdate /tmp/loadavg.rrd
rrdtool info /tmp/loadavg.rrd

3.绘制图形

绘制过去一小时的图,每隔 10m 刷新一次：
vim /root/plot_loadavg_hourly.sh
rrdtool graph /var/www/html/load_avg_hourly.png -X 0\
--start=$(date --date=-1hour +%s) –end=$(date +%s) \
DEF:ldavg1=/tmp/loadavg.rrd:loadavg1:AVERAGE\
DEF:ldavg2=/tmp/loadavg.rrd:loadavg5:AVERAGE \
DEF:ldavg3=/tmp/loadavg.rrd:loadavg15:AVERAGE\
LINE1:ldavg1#FF0000A0:”1 Min Load”\
LINE1:ldavg5#00FF00A0:”5 Min Load” \
LINE1:ldavg15#0000FFA0:”15 Min Load”\

crontab –e
*/10 * * * * /root/plot_loadavg_hourly.sh

绘制过去一天的图,每隔 1h 刷新一次：

vim /root/plot_loadavg_daily.sh
rrdtool graph /var/www/html/load_avg_daily.png -X 0\
--start=$(date --date=-1day +%s) --end=$(date +%s) \
DEF:ldavg1=/tmp/loadavg.rrd:loadavg1:AVERAGE\
DEF:ldavg5=/tmp/loadavg.rrd:loadavg5:AVERAGE \
DEF:ldavg15=/tmp/loadavg.rrd:loadavg15:AVERAGE\
LINE1:ldavg1#FF0000A0:”1 Min Load”\
LINE1:ldavg2#00FF00A0:”5 Min Load” \
LINE1:ldavg3#0000FFA0:”15 Min Load”\

crontab –e
0 */1 * * * /root/plot_loadavg_hourly.sh

rrdtool create 查看帮助
http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html
http://my.oschina.net/u/1458120/blog/208857
man rrdgraph_examples 查看举例

三、性能调优

1、对列理论(Queueing Theory)

利特尔法则：
对列长度=平均到达率 x 平均等待时间

$L = \lambda x W$

最佳情况是：到达率和完成率相等，即对列长度为 0

等待时间：
waitime = service time + queue time

W=S+Q

service time:服务处理时间
queue time:等待时间

综上所述:

$L = \lambda x (S+Q)$

限流法则:
系统吞吐量(单位时间完成的任务数)=某资源的吞吐量/某资源的单位访问量
Xsystem = Xresource / Vresource
整个系统的性能取决于系统中最慢的环节（例如 disk 可以考虑 raid 等分流提升吞吐量）

2、渐进复杂度(Asymptotic Complexity)

算法的渐近的复杂度是衡量其资源需求增长速度随着输入规模趋于无穷;
较好的算法是随着输入的无限增长时，所消耗的资源是否增长非常厉害;如果增长较低表示渐进复杂度较低，效率较高
参考资料: http://www.cs.cornell.edu/courses/cs312/2006fa/lectures/lec11.html

3、查看和配置内核模块

lsmod 查看内核已装载的模块
modinfo –p usb_storage 查看 usb_storage 模块的详细信息

cat /sys/module/usb_storage/parameters/delay_use

rmmod usb_storage 卸载模块
modprobe usb_storage delay_use=5 手动加载模块并设置参数，但不会永久生效
cat /sys/module/usb_storage/parameters/delay_use 查看 usb_storage 模块参数值

设置模块参数永久生效的方法：
A.vim /etc/modprobe.d/usb_storage.conf
options usb_storage delay_use=5
然后 modprobe usb_storage 会加载配置文件的参数配置
depmod -a 把当前已加载的模块信息写入/boot/ System.map-2.6.32-220.el6.x86_64 配置文件，下次重启会重新加载(`depmod -e -F` /boot/ System.map-2.6.32-220.el6.x86_64 分析目前已经加载但不可执行的模块，具体参数查看帮助 depmod --help)
B. grep -n modules /etc/rc.sysinit 可以看到脚本会执行/etc/sysconfig/modules/*.modules 中的文件
vim /etc/sysconfig/modules/usb_storage.modules
modprobe usb_storage delay_use=5
chmod +x /etc/sysconfig/modules/usb_storage.modules

两种方法的不同点：A 是系统启动之后加载 B 是系统启动时加载

4、安装和使用调优工具 tuned

yum –y install tuned
service ktune start
service tuned start
chkconfig ktune on
chkcofnig tuned on

tuned-adm list 查看性能优化模式都有哪些
man tuned-adm 可以查看每个工作模式的介绍
tuned-adm active 查看当前状态
tuned-adm off 关闭所有 tuning
tuned-adm profile throughput-performance 设置成最大吞吐量模式,此时 tuned 会调整/etc/tune-profiles/throughput-performance 里面的文件参数

ktune.sh 被 ktuned 服务调用的一个 init 风格的脚本，在系统启动时可以运行特定的命令来对系统进行调优
ktune.sysconfig 启用和关闭 ktune，并且设置不同的磁盘调度策略
sysctl.ktune 设置一些内核参数,类似/etc/sysctl.conf 文件
sysctl.s390x.ktune
tuned.conf 启用和关闭各种监控和调优插件，全局配置/etc/tuned.conf

5、磁盘碎片

为了避免产生碎片，文件系统上就必须有足够的连续空闲块，工具包：e2fsprogs
filefrag[-v]　filename　　查看某个文件占用的 extents 数量
e2freefrag　/dev/vda　e2freefrag 命令来确认有多少连续的空闲块
e4defrag -v　/root/file1　消除碎片

6、Raid Layout

raid5 最少三块盘（一个校验盘，两个数据盘），raid6 最少五块盘（二个校验盘，三个数据盘）
对 raid0,5,6 存在两个特别重要的概念：（chunk size），（Stride size）
block size :系统块大小
chunksize :RAID 中每个成员盘一次写入的数据量，大于 chunk size 才到下一个盘读写
Stride-size：步长，Stride size = (chunk size)/(filesystem blocksize)，在一个盘中一次性写入量（多少个 block）
stripe-width= stride-size * N：条带大小 在 raid 组的一个条带中写入数据盘的 block 数量，N 代表 raid 组中的数据盘个数

查看系统 block 大小：
tune2fs -l /dev/sda1

例子：
mkfs –t ext4 –E stride=16,stripe-width=64 /dev/san/lun1

7、资源限制(pam & cgroup)

A.ulimit 命令/PAM 模块(资源限制能力比较弱，目前 RSS IO 无法限制)：
是一个内建命令，通过 ulimit 设置只对当前 bash 生效
ulimit -a　　查看当前用户所有限制
help ulimit　查看帮助
永久生效方法：
使用 pam_limits.so 模块可以在登录后打开会话时设置资源控制,在/etc/pam.d/system-auth 文件会调用 pam_limits.so 模块，然后就会应用 /etc/security/limits.conf 和 /etc/security/limits.d/*.conf 配置
cat /etc/pam.d/system-auth|grep pam_limts
session　　required　　pam_limits.so
限制的资源可以通过查看默认的/etc/security/limits.conf 文件注释或者 help ulimit 看到帮助

B.利用 cgroup 进行资源限制
cgroup 会将资源细分成 controllers，比如 cpu memory blkio 等
controllers：
bklio　　这个子系统限制块设备的输入输出控制（disk cdrom usb…）
cpu　　限制可用的 cpu 时间
cpuset　　限制 cpu 核数(多核系统)和在 NUMA 中的内存区域

cpuacct　　这个子系统会自动生成关于所使用的 CPU 资源的报告
devices　　允许或据对访问设备
freezer　　暂停和恢复 cgroup 任务
memory　　这个子系统设置限制内存,并生成自动报告这些任务所使用的内存资源
net_cls　　这个子系统用 classid 标识网络数据包，允许 cgroup 任务使用 TC(traffic controller)来识别数据包
net_prio　　这个子系统提供了一种方法来动态地设置每个网络接口网络流量的优先级
ns　　命名空间子系统
这些子系统也被称做资源控制器或者控制器

```
yum -y install libcgroup
vim /etc/cgconfig.conf
mount {
        cpuset  = /cgroup/cpuset;
        cpu     = /cgroup/cpu;
        cpuacct = /cgroup/cpuacct;
        memory  = /cgroup/memory;
        devices = /cgroup/devices;
        freezer = /cgroup/freezer;
        net_cls = /cgroup/net_cls;
        blkio   = /cgroup/blkio;
}
```

mount 选项可以自定义控制器，控制器的相关定义在/cgroup/目录

/etc/cgrules.conf　分配资源到相应的限制组内

chkconfig cgconfig on
service cgconfig start (启动后/cgroup 会生成相应的目录)
man cgconfig.conf 查看帮助和使用实例
lssubsys -am　　查看已经 mount 的子系统
cat /proc/cgroup　　查看 cgroup 当前相关信息

实例：
A.定义资源组
vim /etc/cgconfig.conf

```
mount {
        cpuset  = /cgroup/cpuset;
        cpu     = /cgroup/cpu;
        cpuacct = /cgroup/cpuacct;
        memory  = /cgroup/memory;
        devices = /cgroup/devices;
        freezer = /cgroup/freezer;
```

```
        net_cls = /cgroup/net_cls;
        blkio     = /cgroup/blkio;
}
group    cpulimit_hig {
                    cpu {
                                cpu.shares = 20;
        }
}
group cpulimit_low{
                cpu{
                                cpu.shares = 10;
        memory{
         memory.max_usage_in_bytes=100000;
        }
        }
}
service cgconfig restart
ll /cgroup/cpu/cpulimit_hig/
ll /cgroup/cpu/cpulimit_low/
```

B.将进程或者脚本分配到相应的资源限制组

```
vim /etc/cgrules.conf
*:a.sh      cpu                    cpulimit_hig/
*:b.sh      cpu,memory             cpulimit_low/
service cgred restart
```

详细参数可以参考 man cgrules.conf

C.测试:

```
a.sh b.sh
 #!/bin/bash
while :
do
        :
done
```

a.sh 和 b.sh 同时执行会发现所有 a.sh 消耗的 cpu 的总百分比和所有 b.sh 消耗的总的 cpu 百分比总是 2：1 的关系，如果只有 a.sh 或者 b.sh 执行时将会独占 CPU

练习题：创建一个资源控制组为 bigload，限制 dd 进程在主磁盘上读速率不超过 1MiB/s
blkio.throttle.read_bps_device 语法:MAJOR:MINOR Bytes/s

MAJOR:MINOR 表示设备主次设备号
cat /proc/partitions 或者 ll /dev/sda* 查询

```
vim /etc/cgconfig.conf
mount {
```

```
        cpuset    = /cgroup/cpuset;
        cpu       = /cgroup/cpu;
        cpuacct = /cgroup/cpuacct;
        memory    = /cgroup/memory;
        devices = /cgroup/devices;
        freezer = /cgroup/freezer;
        net_cls = /cgroup/net_cls;
        blkio     = /cgroup/blkio;
}
group bigload{
                blkio{
                        blkio.throttle.read_bps_device="8:0 1048576";
}
}
service cgconfig reload

vim /etc/cgrules.conf
*:dd        blkio                bigload/
service cgred reload
```

测试可使用 iotop 软件进行观察
dd if=/dev/zero of=/dev/null

ps -eLo pid,cgroup,cmd|grep -i dd　　查看是否启动 cgroup 正常

参考资料：
Linux Ggroups 相关 http://www.aikaiyuan.com/5904.html

四、硬件配置相关

1、搜集硬件信息

FSB/QPI/DMI http://www.365pcbuy.com/article-373.html

dmesg 命令

/var/log/dmesg
在系统引导时，内核将与硬件和模块初始化相关的信息会填充到 ring buffer 里面,开机后保存在/var/log/dmesg

分析/var/log/dmesg 中的重要内容：
1) 内核版本及内核启动的参数
   Initializing cgroup subsys cpuset
   Initializing cgroup subsys cpu

Linux version 2.6.32-358.11.1.el6.x86_64 (mockbuild@c6b7.bsys.dev.centos.org) (gcc version 4.4.7 20120313 (Red Hat 4.4.7-3) (GCC)) #1 SMP Wed Jun 12 03:34:52 UTC 2013

Command line: ro root=UUID=97693d73-443f-438a-90a3-208855faff19 rd_NO_LUKS KEYBOARDTYPE=pc KEYTABLE=us rd_NO_MD crashkernel=auto LANG=zh_CN.UTF-8 rd_NO_LVM rd_NO_DM rhgb quiet

2) 组织和分配内存到 NUMA 节点和分配区域

On node 0 totalpages: 4182667

DMA zone: 56 pages used for memmap

DMA zone: 105 pages reserved

DMA zone: 3834 pages, LIFO batch:0

DMA32 zone: 14280 pages used for memmap

DMA32 zone: 822056 pages, LIFO batch:31

Normal zone: 45696 pages used for memmap

Normal zone: 3296640 pages, LIFO batch:31

On node 1 totalpages: 4194304

Normal zone: 57344 pages used for memmap

Normal zone: 4136960 pages, LIFO batch:31

3) 探测到的物理 CPU 上，单个 CPU 激活的核数

Booting Node 1, Processors #1 Ok.

Booting Node 0, Processors #2 Ok.

Booting Node 1, Processors #3 Ok.

Booting Node 0, Processors #4 Ok.

Booting Node 1, Processors #5 Ok.

Booting Node 0, Processors #6 Ok.

Booting Node 1, Processors #7 Ok.

Booting Node 0, Processors #8 Ok.

Booting Node 1, Processors #9 Ok.

Booting Node 0, Processors #10 Ok.

Booting Node 1, Processors #11 Ok.

Booting Node 0, Processors #12 Ok.

Booting Node 1, Processors #13 Ok.

Booting Node 0, Processors #14 Ok.

Booting Node 1, Processors #15 Ok.

Booting Node 0, Processors #16 Ok.

Booting Node 1, Processors #17 Ok.

Booting Node 0, Processors #18 Ok.

Booting Node 1, Processors #19 Ok.

Booting Node 0, Processors #20 Ok.

Booting Node 1, Processors #21 Ok.

Booting Node 0, Processors #22 Ok.

Booting Node 1, Processors #23

Brought up 24 CPUs

Total of 24 processors activated (91193.83 BogoMIPS)

4) 分配大内存页面初始化机制

HugeTLB registered 2 MB page size, pre-allocated 0 pages

5) 存储初始化,I/O 策略

io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)

6) 网卡信息

tg3 0000:02:00.0: eth0: Tigon3 [partno(BCM95720) rev 5720000] (PCI Express) MAC address 90:b1:1c:59:7c:2d
tg3 0000:02:00.0: eth0: attached PHY is 5720C (10/100/1000Base-T Ethernet) (WireSpeed[1], EEE[1])
tg3 0000:02:00.0: eth0: RXcsums[1] LinkChgREG[0] MIirq[0] ASF[1] TSOcap[1]
tg3 0000:02:00.0: eth0: dma_rwctrl[00000001] dma_mask[64-bit]
alloc irq_desc for 17 on node 0
alloc kstat_irqs on node 0
alloc irq_2_iommu on node 0
tg3 0000:02:00.1: PCI INT B -> GSI 17 (level, low) -> IRQ 17
tg3 0000:02:00.1: setting latency timer to 64
tg3 0000:02:00.1: eth1: Tigon3 [partno(BCM95720) rev 5720000] (PCI Express) MAC address 90:b1:1c:59:7c:2e
tg3 0000:02:00.1: eth1: attached PHY is 5720C (10/100/1000Base-T Ethernet) (WireSpeed[1], EEE[1])
tg3 0000:02:00.1: eth1: RXcsums[1] LinkChgREG[0] MIirq[0] ASF[1] TSOcap[1]
tg3 0000:02:00.1: eth1: dma_rwctrl[00000001] dma_mask[64-bit]

lscpu、getconf 和 dmidecode

A．lscpu

```
[root@localhost ~]# lscpu
Architecture:          x86_64                                  ❶
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24                                      ❷
On-line CPU(s) list:   0-23
Thread(s) per core:    2                                       ❸
Core(s) per socket:    6                                       ❹
Socket(s):             2                                       ❺
NUMA node(s):          2                                       ❻
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Stepping:              7
CPU MHz:               1899.994
BogoMIPS:              3799.49
Virtualization:        VT-x                                    ❼
L1d cache:             32K                                     ❽
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22          ❾
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23
```

说明：

❶ CPU 架构

❷ 逻辑可用的 CPU 个数（核数）

❸ 每核超线程数

❹ 每个插槽支持的 cpu 核数

❺ CPU 物理插槽的数量

❻ 非均匀内存访问节点数量

❼ 是否支持虚拟化 VT-x(Intel,支持全虚拟化) AMD-v(AMD)

❽ 一级数据缓存(L1i cache:一级指令缓存)

❾ 映射到一个 numa 总线的逻辑 cpu

B．getconf 获取系统信息

例如

获取页大小：getconf　PAGESIZE

获取所有系统信息:getconf -a

C. dmidecode(DMI 桌面管理接口，RHEL5 环境可使用)

dmidecod -t processor　查看 CPU 信息

dmidecod -t cache　查看 CPU 缓存信息

cat /proc/meminfo

dmidecode -t　列出可查询的分组

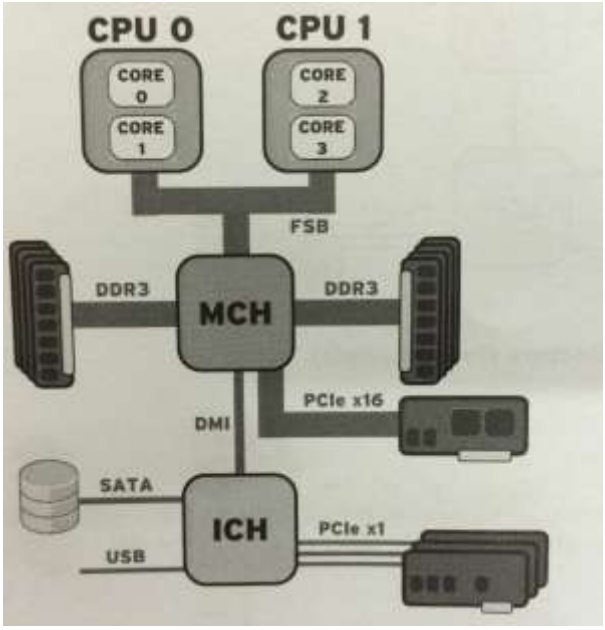
dmidecode -t memory　查看内存信息

dmidecode -t cache　查看 CPU L1 L2 L3 缓存，计算方法参考 http://debugo.com/ls-cpu-dmidecode/ （可直接使用 lscpu 命令）

其他：

lshw、hwinfo	显示硬件信息列表
lspci	显示 pci 总线相关信息
lsscsi	列出 scsi 设备信息
lsusb	列出 usb 总线信息及设备信息
lsblk	列出块设备信息
lshal	查看所有注册到 hal 的硬件信息
sosreport	搜集关于系统硬件和系统的详细信息

SMP 和 NUMA 架构的区别



相关术语：

SMP(Symmetric Multi-Processor)：多处理器架构

FSB (Front-Side Bus)：前端总线

MCH(Memory Controller Hub):常说的北桥（控制着到声卡、内存通信包括到 ICH 访问 PCie、disk 等）

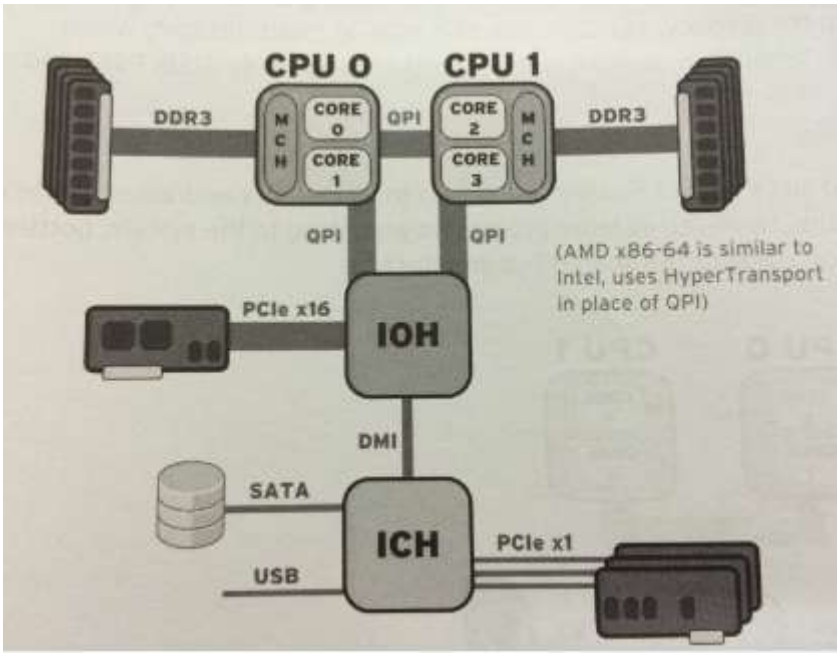
ICH(I/O Controller Hub):常说的南桥

DMI(Direct Media Interface):直连媒体接口,连接南北桥总线

描述：

每一个 CPU 通过共享的 FSB 总线和 MCH 通信，MCH 上的内存控制器然后处理访问内存；当 CPU 请求较慢的设备如 disk、usb 时,MCH 会通过 DMI 与 ICH 通信，ICH 来处理 disks、ps/2、USB 等；

由上图的 SMP 架构可知，所有 CPU 争用一个总线来访问所有内存，优点是资源共享了，而缺点是无论是 CPU 和内存的增加，总线的争用会越加严重



相关术语：

NUMA(Non-Uniform Memory Access):非一致性内存访问

QPI(Quick Path Interconnect)：快速直连总线

IOH(Input Output Hub): 由于 NUMA 架构内存控制器等功能集成到了 CPU，IOH 这里起到传统意义上的南桥的功能

描述：

每个 CPU 在访问内存时通过 QPI(快速直连通道,在 Interl CPU 中是 HyperTransport）替代 FSB 直接访问内存，取消 MCH 由 IOH 替代

NUMA 解决了 SMP 架构扩展的问题，内存就近访问原则，访问本地内存的速度将远远高于访问远地内存（为了更好地发挥系统性能，开发应用程序时需要尽量减少不同 CPU 模块之间的信息交互）

单实例 Mysql 服务器要关闭 NUMA，多实例 Mysql 可以启用 NUMA 结合 CGroup 来灵活进行资源控制

NUMA 两个重要的管理命令：

numactl 是设定进程 NUMA 策略的命令行工具。对于那些无法修改和重新编译的程序，它可以进行非常有效的策略设定。Numactl 使管理员可以通过简单的命令行调用来设定进程的策略， 并可以集成到管理脚本中。

numactl(详细例子可参考 man numactl)的主要功能包括：

1). 设定进程的内存分配基本策略

2). 限定内存分配范围，如某一特定节点或部分节点集合

3). 对进程进行节点或节点集合的绑定

4). 修改命名共享内存，tmpfs 或 hugetblfs 等的内存策略

5). 获取当前策略信息及状态

6). 获取 NUMA 硬件拓扑

下面是使用 numactl 设定进程策略的实例：

numactl --cpubind=0 --membind=0,1 program

其意义为：在节点 0 上的 CPU 运行名为 program 的程序，并且只在节点 0,1 上分配内存。Cpubind 的参数是节点编号，而不是 cpu 编号。在每个节点上有多个 CPU 的系统上，编号的定义顺序可能会不同

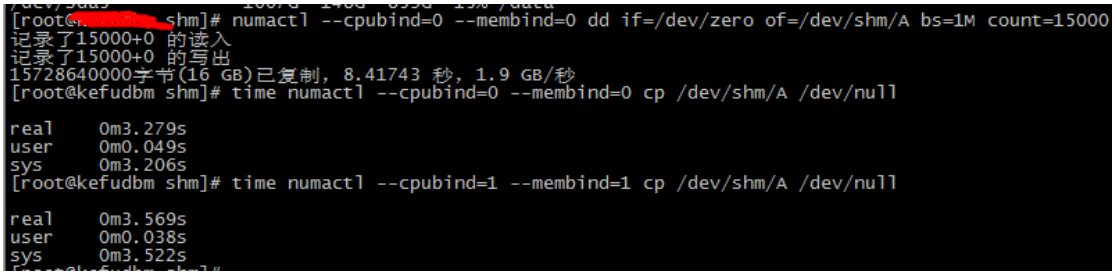
下面是使用 numactl 更改共享内存段的分配策略的实例：

numactl --length=1G --file=/dev/shm/interleaved --interleave=all

其意义为： 对命名共享内存 interleaved 进行设置，其策略为全节点交织分配，大小为 1G。

numastat 是获取 NUMA 内存访问统计信息的命令行工具。对于系统中的每个节点，内核维护了一些有关 NUMA 分配状态的统计数据。numastat 命令会基于节点对内存的申请，分配，转移，失败等等做出统计，也会报告 NUMA 策略的执行状况。这些信息对于测试 NUMA 策略的有效性是非常有用的。

测试 numa 跨 node 读：



关闭 NUMA 方法：

1) BIOS 设置
2) 可以直接在/etc/grub.conf 的 kernel 行最后添加 numa=off

numactl --hardware 查看详细的节点数

numstat

local_node 在本地申请内存的次数

other_node 在远端申请内存的次数(越少越好)

可以结合 cgroup 灵活分配资源

参考文章：

玩转 CPU Topology

http://www.searchtb.com/2012/12/%E7%8E%A9%E8%BD%ACcpu-topology.html

NUMA 取舍

http://www.cnblogs.com/yjf512/archive/2012/12/10/2811823.html

Mysql 单机多实例情况

http://www.hellodb.net/tag/numa

2、I/O Scheduling

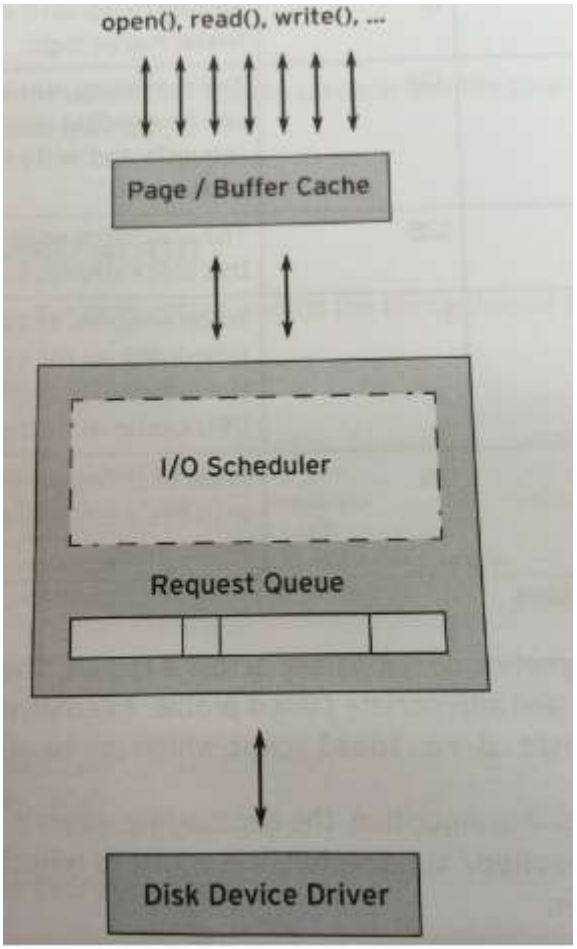
通常磁盘的读写影响是由磁头到柱面移动造成了延迟，解决这种延迟内核主要采用两种策略：缓存和 IO 调度算法来进行弥补

Caching：IO 请求被缓存在大页和 buffer caches 里面，读请求会预先从缓存读取，写请求会先写进缓存，然后在保存到磁盘

四种 IO 调度算法：

cat /sys/block/sda/queue/scheduler

noop anticipatory deadline [cfq] (当前是 cfq)



noop：noop 调度算法不会对 I/O 请求排序操作，除了合并外也不会做任何其他优化，直接以类似 FIFO 的顺序提交 I/O 请求；对于 SSD、虚拟机或者存储设备可能会更加高效

anticipatory(as)：基于预测的 IO 算法，类似 DeadLine，也维护了三个请求对列；区别在于当它处理完一个 I/O 请求后并不会直接返回处理下一个请求，而是等待 6ms(默认),如果这时候有新来的针对当前扇区相邻扇区的请求，那么会直接处理它，当等待时间结束后，调度器才返回处理下一个对列请求

试想一下，如果系统有频繁的针对邻近扇区的 I/O 请求，那么这种预测算法必然大幅提高整体的吞吐量，毕竟节约了那么多寻道时间

deadline：DEADLINE 在 CFQ 的基础上，解决了 IO 请求饿死的极端情况。除了 CFQ 本身具有的 IO 排序队列之外，DEADLINE 额外分别为读 IO 和写 IO 提供了 FIFO 队 列。读 FIFO 队列的最大等待时间为 500ms，写 FIFO 队列的最大等待时间为 5s。FIFO 队列内的 IO 请求优先级要比 CFQ 队列中的高，，而读 FIFO 队列的优先级又比写 FIFO 队列的优先级高。优先级可以表示如下：

FIFO(Read) > FIFO(Write) > CFQ

deadline 算法保证对于既定的 IO 请求以最小的延迟时间，从这一点理解，对于 DSS 应用

应该会是很适合的

cfq(2.6.18+内核默认 CFQ)：该算法的特点是按照 IO 请求的地址进行排序，而不是按照先来后到的顺序来进行响应。在传统的 SAS 盘上，磁盘寻道花去了绝大多数的 IO 响应时间。CFQ 的出发点是对 IO 地址进行排序，以尽量少的磁盘旋转次数来满足尽可能多的 IO 请求。在 CFQ 算法下，SAS 盘的吞吐量大大提高了。但是相比于 NOOP 的缺点是，先来的 IO 请求并不一定能被满足，可能会出现饿死的情况；

调度算法适用场合：

在传统的 SAS 盘上，CFQ、DEADLINE、ANTICIPATORY 都是不错的选择；对于专属的数据库服务器和文件服务器，DEADLINE 的吞吐量和响应时间都表现良好，适用于大量 IO 操作的环境

在 SSD、Fusion IO 上，最简单的 NOOP 反而可能是最好的算法，因为其他三个算法的优化是基于缩短寻道时间的，而固态硬盘没有所谓的寻道时间且 IO 响应时间非常短。

ANTICIPATORY 通常更适用于大量持续读的环境，并不适用于 DB Server

CFQ 适用于有大量来自不同进程的小的并发读写的环境如桌面环境等

手动临时更改调度算法：

echo deadline > /sys/block/sda/queue/scheduler

永久更改：

A.使用 tuned 来修改调度算法

比如：vim /etc/tuned-profiles/throughtput-performance/ktune.sysconfig

ELEVATOR="deadline"

ELEVATOR_TUNE_DEVS="/sys/block/{sd,cciss,dm-,vd}*/queue/scheduler"

tuned-admin profile throughtput-performance

chkconfig tuned on

chkconfig ktune on

更改调度算法之后/sys/block/sda/quue/iosched/会生成对应的参数文件

B.通过修改 grub.conf 来修改调度算法

kernel /vmlinuz-2.6.32-358.11.1.el6.x86_64 ro root=UUID=97693d73-443f-438a-90a3-208855faff19 rd_NO_LUKS KEYBOARDTYPE=pc KEYTABLE=us rd_NO_MD crashkernel=auto LANG=zh_CN.UTF-8 rd_NO_LVM rd_NO_DM **elevator=deadline** rhgb quiet

查看调度算法参数的含义：

yum -y install kernel-doc

比如:/usr/share/doc/kernel-doc-2.6.32/Documentation/block/deadline-iosched.txt

3、CPU Scheduling

nice 调整范围-20~19,对于实时进程优先级并不重要

chrt 设置实时进程优先级

chrt -p 2339 查看 pid 为 2339 进程的调度策略和进程优先级
chrt -m 查看各种调度策略和其优先级范围
chrt -f 20 md5sum /dev/zero & 表示使用 fifo 调度算法指定实时进程优先级为 20 的方式运行 md5sum /dev/zero 命令
chrt -f -p 98 3 设置 pid 为 3 的进程调度算法为 fifo 优先级为 98
ps ax -o pid,cmd,class,rtprio,pri,nice,policy 查看系统所有进程的优先级等
非实时进程优先级：top 查看 PR 值,为 RT 时表示实时进程

(动态)非实时优先级进程调度策略: SCHED_BATCH,SCHED_OTHER,SCHED_IDLE
(静态)实时优先级进程调度策略： SCHED_RR,SCHED_FIFO 优先级高于所有非实时优先级进程,但在 RHEL6.x 版本不完全依赖优先级,基于虚拟时间,尽可能公平的调度进程

优先级范围对应关系：
System Priority
Real-time Priority
Nice Level
top[PR]

关于详细的完全公平对列调度算法(CFS)可参考帮助：
/usr/share/doc/kernel-doc-2.6.32/Documentation/scheduler/sched-design-CFS.txt

4、进程管理

5、strace、ltrace、valgrind

strace 常用来跟踪进程执行时的系统调用
strace ls 查看 ls 命令的系统调用
strace -c ls 统计 ls 命令系统调用
strace -fc ls 统计 ls 命令系统调用,并对子进程也进行统计-f(fork)
strace -c -S calls ls 统计 ls 命令系统调用并按调用 call 次数排序
strace -e open iptables -L 只查看 open 系统调用
strace -p 3192 跟踪 pid 为 3192 进程的系统调用
kill -l 查看所有信号列表
详细参数参考：
http://www.cnblogs.com/ggjucheng/archive/2012/01/08/2316692.html

ltrace 常用来跟踪库函数调用情况和系统调用
ltrace -cf grep -r test /etc 统计命令 grep -r test /etc 都使用了哪些库函数,包含子进程-f(fork)
ltrace -Scf grep -r test /etc -S 表示包含系统调用
ltrace -p 3192 跟踪 pid 为 3192 进程的库函数调用

valgrind 用于分析缓存使用情况

yum -y install valgrind cache-lab
cache-lab 实验环境才有
测试两个功能一样的脚本哪个 CPU 缓存命中率高：
/usr/local/bin/cache1
/usr/local/bin/cache2

valgrind--tool=cachegrind cache1
valgrind--tool=cachegrind cache2
执行后查看哪个 miss rate 高，命中率就低,软件性能就较差
另外可以使用 perf 同样可以做到
perf 是一款很好的 linux 性能分析工具，可以对硬件层和软件层都可以进行分析优化
可参考 http://www.ibm.com/developerworks/cn/linux/l-cn-perf1/
yum --y install perf cache-lab
perf stat --e cache-misses cache1
perf stat --e cache-misses cache2
执行后查看 cache-misses 值越高代表命中率越低

6、systemtap

systemtap 是内核开发者必须要掌握的一个工具，利用 Kprobe 提供的 API 来实现动态地监控和跟踪运行中的 Linux 内核的工具，相比 Kprobe，systemtap 更加简单，提供给用户简单的命令行接口，以及编写内核指令的脚本语言
参考文档:
http://www.cnblogs.com/hazir/p/systemtap_introduction.html
https://sourceware.org/systemtap/tutorial/
http://www.redbooks.ibm.com/abstracts/redp4469.html
http://www.ibm.com/developerworks/cn/linux/l-systemtap/

测试环境需要安装:kernel-debuginfo kernel-debuginfo-common kernel-devel systemtap-runtime gcc
调试好之后可以把生成的模块放到生产环境只需安装 systemtap-runtime 软件包
实例：
分为两步，在开发测试库编写 xx.tap 相关功能脚本，使用 stap –v xx.tap 测试成功后,使用 stap -p4 –m xx.ko 编译成 xx.ko 模块，拷贝到生产环境,在生产环境 staprun xx.ko 运行模块使用其功能

1、 在开发环境编写调试脚本
yum –y install kernel-devel systemtap-* gcc
wget
http://debuginfo.centos.org/6/x86_64/kernel-debuginfo-2.6.32-431.el6.x86_64.rpm
wget
http://debuginfo.centos.org/6/x86_64/kernel-debuginfo-common-x86_64-2.6.32-431.el6.x86_64.rpm

yum -y localinstall kernel-debuginfo*

安装好 systemtap 之后查看默认的例子和帮助:
/usr/share/doc/systemtap-client/examples
stap -v topsys.stp 测试 topsys.stp 脚本是否正常
stap -v -p4 -m /tmp/systop.ko topsys.stp 编译到第 4 步生成 ko 模块为 systop.ko
将 systop.ko 拷贝到生产环境

2、在生产环境使用

yum -y install systemtap-runtime
对于普通用户运行:
useradd cc
usermod -aG stapusr cc
usermod -aG stapdev cc
su -cc
staprun /tmp/systop.ko

注意：测试机和生产机器内核版本和相关软件版本都要一致

五、邮件服务器优化

Disk 建议调度算法适用 CFQ/Deadline，自行测试
Memory
CPU
Network

MTA：sendmail qmail postfix exchange
SMTP/IMAP 如果加密了 ssl，可以适用 aes-ni 来对加解密加速工作
IMAP/SMTP 使用集群
ext4/ext3 文件系统去掉 atime,mount -o remount,noatime /

六、大内存场景优化 Tips

虚拟内存：最小单位是 pages
物理内存：最小单位是 page frames
1 个 page frame 等于 1 个 page 的数据

ps -aux or top
VIRT or VSZ:虚拟内存大小
RES or RSS:实际分配的物理内存大小
pmap $$ 查看当前进程的虚拟内存大小 VSZ

ps -aux|grep $$

pages→page frames 映射的对应关系存在表中叫页表，为了高效率查询页表引入了TLB(Translation Look-aside Buffer,TLB 缓存是在 CPU 里面)
x86info -c 可以查看 TLB 信息

查找内存泄漏
相关工具：ps、top、free、sar -r、sar -R、memcheck、valgrind

valgrind --tool=memcheck bigmem 256 检查 bigmem 程序分配 256M 内存时是否产生了内存泄漏
valgrind --leak-check=full bingmem 256 检查程序 bigmem 执行过程中哪里产生了内存泄漏
grep Committed_AS /proc/meminfo 可以查看分配的虚拟内存大小
两种不同类型内存泄漏：
A、 虚拟内存增长，物理内存几乎不变
B、 虚拟内存和物理内存都增加

SWAP 优化：
进程和 cache 都需要内存，cache 分 buffers 和 page cached

	total	used	free	shared	buffers	cached
Mem:	1877	1769	108	0	81	336
-/+ buffers/cache:		1350	526			
Swap:	0	0	0			

-/+ buffers/cache: free：526=108+81+336

buffers：索引信息(从磁盘索引文件时的索引信息)
cached：缓存文件、数据内容

在内存不够用时如果清除 cache 会造成 IO 低效,会从磁盘读取相关内容,此时使用 swap 会更好些，使内存做尽可能有意义的事情，但如果 swap 换入换出很频繁对性能影响也很大

swap_tendency = mapped_ration/2 +distress+vm.swappiness
mapped_ration:实际已经分配的物理内存的百分比如 1769 / 1877
distress：表示内核释放内存有多难,越难值越大(0~100)
cat /proc/sys/vm/swapiness :

swap_tendency<100 内核回收 page cache 以释放更多内存空间
swap_tendency>=100 优先去使用 swap

sysctl -w vm.swappiness=70

优化 swap:
对于传统的机械磁盘，尽量将交换分区划分到外面的磁道；亦可以适用 SSD(SLC)来作为 swap

分区
多个 swap 分区时，可以设置优先级 优先适用指定 swap 分区

内存回收机制：
内存四种状态：
free :可以立即被分配出去的物理 page frames(页帧)
Inactive clean :读缓存空间，可以被回收
Inactive Dirty ：被修改过的数据未写回磁盘所占用的内存空间
Active ：正在适用无法释放的内存空间
grep -i acvive /proc/meminfo 其中换出到 swap 的是 Inactive(file)
其他可参考：http://blog.sina.com.cn/s/blog_7106477c0100qj9d.html

flush 进程作用：释放存储在缓存区中的数据
ll /proc/sys/vm/
vm.dirty_expire_centisecs: 脏数据在内存中未写会磁盘的有效生命周期，在此时间未写回数据就丢失(单位:1/100 s)
vm.dirty_writeback_centisecs(单位:1/100 s):表示多久写回一次磁盘
vm.dirty_background_ratio:系统中所有进程产生的脏数据达到了总物理内存的百分比立即写入磁盘(单位:百分比)
vm.dirty_ratio:如果系统某个进程产生的脏数据占用总物理内存的百分比时立即写入磁盘(单位:百分比)

OOM-killer

OOM killer（Out-Of-Memory killer），该机制会监控那些占用内存过大，尤其是瞬间很快消耗大量内存的进程，为了防止内存耗尽而内核会把该进程杀掉
防止重要的系统进程触发(OOM)机制而被杀死：

可以设置参数/proc/PID/oom_adj 为-17，可临时关闭 linux 内核的 OOM 机制。内核会通过特定的算法给每个进程计算一个分数来决定杀哪个进程，每个进程的 oom 分数可以/proc/PID/oom_score 中找到；oom_adj 范围是[-17, 15]，值越大越容易被 oom kill 掉，设为 OOM_DISABLE（-17）的进程不会被 oom,内核进程不受此参数控制

注：

默认/proc/sys/vm/panic_on_oom 值为 0 表示启用 OOM，如改为 1 表示暂时关闭 OOM 机制，但当内存紧张时，内核可能会出现重大故障

有时 free 查看还有充足的内存，但还是会触发 OOM，是因为该进程可能占用了特殊的内存地址空间

OOM-killer 策略：

参考文档：http://blog.sae.sina.com.cn/archives/2259

七、CPU 密集型优化 Tips

通过 Cgroup 控制 CPU 利用率

默认：/cgroup/cpu/cpu.shares 为 1024，可认为 100%的 CPU

```
vi /etc/cgconfig
group limitcpu1{
        cpu{
                cpu.share=256;
        }
}
group limitcpu2{
        cpu{
                cpu.share=512;
        }
}
```

简单测试(测试机共有 2 核)：

```
yum -y install cpuload
service cgconfig restart
```

手动将某个程序在 limitcpu1 组执行

```
cgexec   -g cpu:limitcpu1 cpuload
cgexec   -g cpu:limitcpu2 cpuload
```

观察可发现二者 CPU 占用率是 1:2 关系

然后在另外一个终端直接执行 cpuload，默认是在父组(1024)

如果只有在 limitcpu1 limitcpu2 组运行程序，测试机没有其他负载程序时,分别占用的 cpu 比例为:256/(256+512)=1/3 512/(256+512)=2/3

如有其他负载程序在同时执行，别占用的 cpu 比例为:256/(256+512+1024)=1/7 512/(256+512+1024)=2/7

CPU 中断

irqbalance 服务:默认每隔 10s 调整 smp_affinity 参数以便中断更合理均衡的被处理,单核、或者 L2 缓存是共享的,irqbalance 作用并不明显;调整过于频繁对于 CPU 缓存效果不利

```
cat /proc/interrupts                          查看中断情况
watch -n 1 'cat /proc/interrupts'|grep eth0   观察网卡中断情况
cat /proc/irq/18/smp_affinity                 查看中断号为 18 在哪 CPU 上工作

service irqbalance start

ONESHOT=yes           开启表示每个 1min 中断平衡一次
IRQ_AFFINITY_MASK     指定均衡的 cpu 列表
```

指定进程在特定 cpu 执行：

```
vi /etc/cgconfig.conf
        group limitcpu{
                cpu{
```

```
                cpu.share=600;
        }
                cpuset{
                        cpuset.mems=0,1;
                        cpuset.cpus=0-17;
        }
}
```

vi /etc/cgrules.conf
*:cpuload cpu limitcpu

cpuset.mems 表示可以使用 node0 node1 的内存
cpuset.cpus 表示可以在 0-17 核上执行此进程

实时进程调度（内核级的进程）

查看是否有实时进程：
top 命令 PR 项显示 RT
chrt -p 6 查看 pid 6 是否为实时进程
chrt -p $$ 查看当前进程是否为实时进程
chrt -m 查看系统都有哪些调度策略
详细可参考 CPU Scheduling 章节

cat /cgroup/cpu/cpu.rt_period_us
1000000 (us)
cat /cgroup/cpu/cpu.rt_runtime_us
950000(us)
表示在 1000000us 周期内有 950000us 时间运行实时进程,一般不需要人为干涉

八、文件服务调优 Tips

文件系统 journal 模式：
ordered(默认) 元数据先写如日志空间，然后在将元数据和数据一起刷新到磁盘
writeback 略
journal 元数据和数据都会写到日志一份，最大程度保证数据安全性，但性能最差

journal 的目的：断电以后能够快速的进行文件系统检查，从日志中获取相应信息来对系统做一定程度的修复
磁盘本身有自己的写缓存
ext4 默认开启了 barries 了,ext3 默认关闭的

如果磁盘有自己的写缓存和备用电池可以不用开启 barries 来保证数据安全

例子：将日志空间独立存储到 sdd1 分区

mkfs -t ext4 -O journal_dev -b 4096 /dev/sdd1　　创建日志空间分区 sdd1(64M 以内就可以)，制作日志设备
mkfs -t ext4 -J device=/dev/sdd1 -b 4096 /dev/sdc1　格式化数据分区指定日志设备为 /dev/sdd1,block 大小为 4k

对现存的分区做修改：
mkfs -t ext4 -O journal_dev -b 4096 /dev/sdd1　　创建日志空间分区 sdd1(64M 以内就可以)，制作日志设备
tune2fs　-o '^has_journal' /dev/sdc1　　先取消/dev/sdc1 的日志空间
tune2fs -j -J device=/dev/sdd1 /dev/sdc1　　给/dev/sdc1 指定新的日志空间 /dev/sdd1
tune2fs -l /dev/sdd1　　查看 sdd1 的 UUID
vim /etc/fstab
UUID=xxxx /data ext4 data=journal 1 1

两个重要的优化 Tips：

disk：　tuned profiles
　　default　　默认适用 CFQ 调度算法，适合大量小的并发读写
　　enterprise-storage 默认除了/和/boot 以外都禁用 barries
　throughput-performance 默认是启用了 barries

network：
　　the network cards
　　the type of cabling
　　number of hops in a connection
　the size of the packets being sent

网络优化：
ethtool eth0　　查看网卡信息
ethtool　-s eth0 autoneg off speed 1000 duplex full　关闭自动协商功能速率调为 1000 全双工(默人自动协商是开启的)
ifcfg-eth0 ：ETHTOOL_OPS=" autoneg off speed 1000 duplex full" 永久生效，一般不需要手动更改,速率和对端连接的设备有关系

qperf 网络带宽测试：
A:
yum -y install qperf
qperf -oo msg_size:1:32k:*2　-uv 192.168.0.7 tcp_bw　　利用 1~32k 大小的消息,消息以指数大小增长来测试 tcp 带宽
B:
yum -y install qperf

qperf

网络 buffers 优化：

TCP/UDP 最大读写 buffer 值
net.core.rmem_max
net.core.wmem_max(in bytes)

TCP buffer 总限制不分读写 buffer
net.ipv4.tcp_mem
格式：
最小值 压力值 最大值

UDP buffer 总限制不分读写 buffer
net.ipv4.udp.mem(in pages , getconf PAGESIZE)
格式：
最小值 压力值 最大值

针对每一个 tcp socket 的读写 buffer 限制
net.ipv4.tcp_rmem net.ipv4.tcp_wmem(in bytes)
格式：
最小值 默认值 最大值
net.ipv4.tcp_rmem=4096 BDP/2 BDP

通常要对接收 buffer 进行优化
全局网络接收 buffer：
net.core.rmem_max=BDP(in bytes)
全局网络发送 buffer :
net.core.wmem_max=BDP(in bytes)

tcp 总的 buffer
net.ipv4.tcp_mem 换算成多少页比 net.ipv4.tcp_rmem 要大

每个 tcp socket 的接收 buffer：
net.ipv4.tcp_rmem=min default max
net.ipv4.tcp_rmem=4096 BDP/2 BDP(in bytes)
net.ipv4.tcp_wmem=4096 BDP/2 BDP(in bytes)

BDP 带宽延时积
RTT(round trip time): ping delay time
network speed

BDP=network speed * RTT
当 BDP 值超过 64KiB 时，需要打开窗口
net.ipv4.tcp_window_scaling=1 (0 是关闭) 默认打开的

Network Bonding:
Mode:
balance-rr(0) ：数据包传输将按顺序发送到 Bonding 中的各个网卡上,提供均衡和容错
active-backup(1) ：此模式只有一个网卡处于激活状态，其余网卡只有在处于工作状态的网卡出错才会被激活,为了不扰乱交换机，Bonding 的 MAC 只有一个是外部可见

balance-xor(2) ：基于 transmit-hash-policy 的传输，默认是简单策略。还有 xmit_hash_policy 可选。PS: Hash policy: 散列策略
broadcast(3) ：广播模式，此模式下对 Bonding 中所有网卡都进行数据包广播，提供容错能力
802.3ad(4) ： 802.3ad 动态链接汇聚模式，创建链接组后从网卡将有同样的速度和双工设置。根据 802.3ad 协议专有的 Balance-tlb 模式（提供负载均衡），Bonding 中的所有网卡将在链路中全部激活，同时信道绑定并不需要专用的交换设备支持。传出通信流量将根据此时的负载分散于此链接中的各个网卡上，传入通信流量则由当前网卡接收。若接收的网卡出错，则将另外的网卡取代之，并使用该出错网卡的 MAC 地址。

balance-tlb(5) ： 略
balance-alb(6) ：Balance-alb 模式提供自适应负载均衡，此模式在 IPV4 通信下包括 balance-tlb 模式外加接收负载均衡并且不需要任何专用交换设备的支持。RLB(Receive Load Balancing)是通过 ARP 协商完成的。bonding 驱动截获本机发送的 ARP 应答，并把源硬件地址改写为 bond 中某个 slave 的唯一硬件地址，从而使得不同的对端使用不同的硬件地址进行通信。这样就实现了网络负载均衡。当其中的一个 slave 失败，就会由其他的 slave 来接管，从而提高了网卡的容错能力

具体原理可参考：
http://blog.csdn.net/xrb66/article/details/7863285
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/sec-Using_Channel_Bonding.html

实例：bonding 网卡模式为 rr
前提 NetworkManager 是关闭的
查看帮助地址：
/usr/share/doc/kernel-doc-2.6.32/Documentation/networking/bonding.txt

vim /etc/modprobe.d/bonding.conf
alias bond0 bonding
options bond0 -o bond0 mode=balance-rr miimon=100

miimon=100 表示每隔 100ms 检查一下链路是否正常

vim /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
IPADDR=192.168.1.1
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no

vim /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none

vim /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none

手动设置网卡延时:
tc qd add dev eth0 root netem delay 2s

巨帧：

MTU:最大传输单元,一般为 1500Bytes,需要网卡的支持
其中 tcp 有 52Bytes 的开销，udp 有 28Bytes，可见增大 MTU 可以减少开销
应用场景一般是前端节点访问后端存储(iscsi nfs 等)
临时增大:
ifconfig eth0 MTU 9000
永久修改 ifcfg-eth0 文件
MTU=9000

九、数据库服务调优 Tips

1.可选用 tuned 选择 profile：enterprise-storage 或者 latency-performance
2.IO 调度算法 deadline
3.关闭文件系统 barriers[开启会影响性能](最好有后备的电池缓存,如 Raid BBU)
4.网络延时问题：
net.ipv4.tcp_low_latency 默认值为 0,改为 1 优化 tcp stack 来降低数据包延迟提高吞吐量

测试调整前后的变化：
systctl -w “net.ipv4.tcp_low_latency=1”
host1: qperf host2:qperf host1 tcp_lat
5.IPC(进程间通信)优化
ipcs -l
三种进程间通信方法：
semaphores(信号量)：
kernel.sem

message queues(消息对列):
kernel.msgmnb kernel.msgmni kernel.msgmax

shared memory regions(共享内存段):
kernel.shmmni kernel.shmall kernel.shmmax(in pages)

查看帮助：/usr/share/doc/kernel-doc-2.6.32/Documentation/sysctl

实验：优化测试共享内存段,使得最大共享内存是 512MiB

ipcs -l 查看三种进程通信方式的限制情况
ipcs -l -m 查看共享内存段限制情况
ipcs -m 查看当前打开的共享内存段情况

kernel.shmmni 共享内存段的最大数量
kernel.shmall 控制共享内存页数, Linux 共享内存页大小为 4KiB， 共享内存段的大小都是共享内存页大小的整数倍(pages)
kernel.shmmax 单个共享内存段的最大值(bytes)

答案：
sysctl -w “kernel.shmall=131072” (512 * 1024*1024 /4096 单位：页数量)
或者
sysctl -w “kernel.shmmni=8”
sysctl -w “kernel.shmmx= 67108864” (512 * 1024 * 1024/8 单位：bytes[每个共享内存段大小])

6. Huge Pages

huge pages 不能交换到交换分区
grep ^Huge /proc/meminfo
HugePages_Total 总共多少个 HugePages
HugePages_Free 空闲的多少个
HugePages_Rsvd 保留多少个
HugePages_Surp

Hugepagesize 每个 Huge page 大小

程序通过 mmap shmat shmget 调用才可以使用用 huge pages(sysctl -w "vm.nr_hugepages=20" 设置 Huge page 数量),其中 shmat shmget 调用设置了 huge pages 之后就能直接使用，而 mmap 调用需要做额外:
mkdir /hugepagedir
mount -t hugetlbfs none /hugepagedir
查看挂载参数:man mount

Transparent Huge Page(THP 透明巨页或者匿名巨页):内核会自动将小的页合并成大页,可以被交换到交换分区

透明巨页和标准巨页区别是透明巨页是被内核自动管理的
查看是否启用了透明巨页：
cat /sys/kernel/mmm/redhat_transparent_hugepage/enabled
[always]never

不使用 THP:
vim /boot/grub.conf kernel 后面增加
transparent_hugepage=never
cat /proc/meminfo |grep AnonHugePages

7. 优化 overcommit
/proc/sys/vm/overcommit_memory
0 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。
1 表示内核允许分配所有的物理内存，而不管当前的内存状态如何。
2 表示内核允许分配超过物理内存*(overcommit_ratio/100)+swap

/proc/sys/vm/overcommit_ratio

grep ^Commit /proc/meminfo

当前内核限制的可申请的最大内存[物理内存*(overcommit_ratio/100)+swap]
CommitLimit: 961312 kB

已经申请的内存大小
Committed_AS: 59068 kB

/proc/sys/vm/swappiness

值范围[0-100]

表示内存不够时内核是倾向于使用 swap 分区还是 buffer cache，值越小表示越倾向适用 buffer cache(内存)