Linux System and Performance Monitoring

Linux System and Performance Monitoring(CPU 篇)

08 月 10th, 2009 Posted in Linux, Monitor

作者:tonnyom
原载: http://www.sanotes.net/html/y2009/370.html

Linux System and Performance Monitoring(CPU 篇)
Date: 2009.07.21
Author: Darren Hoch
译: Tonnyom[AT]hotmail.com 2009.08.10

前言: 网上其实有很多关于这方面的文章,那为什么还会有此篇呢,有这么几个原因,是我翻译的动力,第一,概念和内容虽然老套,但都讲得很透彻,而且还很全面.第 二,理论结合实际,其中案例分析都不错.第三,不花哨,采用的工具及命令都是最基本的,有助于实际操作.但本人才疏学浅,译文大多数都是立足于自己对原文 的理解,大家也可以自己去 OSCAN 上找原文,如果有什么较大出入,还望留言回复,甚是感激!

1.0 性能监控介绍

性能优化就是找到系统处理中的瓶颈以及去除这些的过程,多数管理员相信看一些相关的"cook book"就可以实现性能优化,通常通过对内核的一些配置是可以简单的解决问题,但并不适合每个环境,性能优化其实是对 OS 各子系统达到一种平衡的定义,这些子系统包括了:

CPU
Memory
IO
Network

这些子系统之间关系是相互彼此依赖的,任何一个高负载都会导致其他子系统出现问题.比如:

大量的页调入请求导致内存队列的拥塞
网卡的大吞吐量可能导致更多的 CPU 开销
大量的 CPU 开销又会尝试更多的内存使用请求
大量来自内存的磁盘写请求可能导致更多的 CPU 以及 IO 问题

所以要对一个系统进行优化,查找瓶颈来自哪个方面是关键,虽然看似是某一个子系统出现问题,其实有可能是别的子系统导致的.

1.1 确定应用类型

基于需要理解该从什么地方来入手优化瓶颈,首先重要的一点,就是理解并分析当前系统的特点,多数系统所跑的应用类型,主要为 2 种:

IO Bound(译注:IO 范畴): 在这个范畴中的应用,一般都是高负荷的内存使用以及存储系统,这实际上表示 IO 范畴的应用,就是一个大量数据处理的过程.IO 范畴的应用不对 CPU 以及网络发起更多请求(除非类似 NAS 这样的网络存储硬件).IO 范畴的应用通常使用 CPU 资源都是为了产生 IO 请求以及进入到内核调度的 sleep 状态.通常数据库软件(译注:mysql,oracle 等)被认为是 IO 范畴的应用类型.

CPU Bound(译注:CPU 范畴): 在这个范畴中的应用,一般都是高负荷的 CPU 占用. CPU 范畴的应用,就是一个批量处理 CPU 请求以及数学计算的过程.通常 web server,mail server,以及其他类型服务被认为是 CPU 范畴的应用类型.

1.2 确定基准线统计

系统利用率情况,一般随管理员经验以及系统本身用途来决定.唯一要清楚的就是,系统优化希望达成什么效果,以及哪些方面是需要优化,还有参考值是什么?因此就建立一个基准线,这个统计数据必须是系统可用性能状态值,用来比较不可用性能状态值.

在以下例子中,1 个系统性能的基准线快照,用来比较当高负荷时的系统性能快照.

```
# vmstat 1
procs memory swap io system cpu
r b swpd free buff cache si so bi bo in cs us sy wa id
1 0 138592 17932 126272 214244 0 0 1 18 109 19 2 1 1 96
0 0 138592 17932 126272 214244 0 0 0 0 105 46 0 1 0 99
0 0 138592 17932 126272 214244 0 0 0 0 198 62 40 14 0 45
0 0 138592 17932 126272 214244 0 0 0 0 117 49 0 0 0 100
0 0 138592 17924 126272 214244 0 0 0 176 220 938 3 4 13 80
0 0 138592 17924 126272 214244 0 0 0 0 358 1522 8 17 0 75
1 0 138592 17924 126272 214244 0 0 0 0 368 1447 4 24 0 72
0 0 138592 17924 126272 214244 0 0 0 0 352 1277 9 12 0 79
```

```
# vmstat 1
procs memory swap io system cpu
r b swpd free buff cache si so bi bo in cs us sy wa id
2 0 145940 17752 118600 215592 0 1 1 18 109 19 2 1 1 96
2 0 145940 15856 118604 215652 0 0 0 468 789 108 86 14 0 0
3 0 146208 13884 118600 214640 0 360 0 360 498 71 91 9 0 0
2 0 146388 13764 118600 213788 0 340 0 340 672 41 87 13 0 0
2 0 147092 13788 118600 212452 0 740 0 1324 620 61 92 8 0 0
2 0 147360 13848 118600 211580 0 720 0 720 690 41 96 4 0 0
2 0 147912 13744 118192 210592 0 720 0 720 605 44 95 5 0 0
2 0 148452 13900 118192 209260 0 372 0 372 639 45 81 19 0 0
2 0 149132 13692 117824 208412 0 372 0 372 457 47 90 10 0 0
```

从上面第一个结果可看到,最后一列(id) 表示的是空闲时间,我们可以看到,在基准线统计时,CPU 的空闲时间在 79% - 100%.在第二个结果可看到,系统处于 100%的占用率以及没有空闲时间.从这个比较中,我们就可以确定是否是 CPU 使用率应该被优化.

2.0 安装监控工具

多数 *nix 系统都有一堆标准的监控命令.这些命令从一开始就是*nix 的一部分.Linux 则通过基本安装包以及额外包提供了其他监控工具,这些安装包多数都存在各个 Linux 发布版本中.尽管还有其他更多的开源以及第三方监控软件,但本文档只讨论基于 Linux 发布版本的监控工具.

本章将讨论哪些工具怎样来监控系统性能.

Tool	Description	Base	Repository
vmstat	all purpose performance tool	yes	yes
mpstat	provides statistics per CPU	no	yes
sar	all purpose performance monitoring tool	no	yes
iostat	provides disk statistics	no	yes
netstat	provides network statistics	yes	yes
dstat	monitoring statistics aggregator	no	in most distributions
iptraf	traffic monitoring dashboard	no	yes

netperf Network bandwidth tool no In some distributions
ethtool reports on Ethernet interface configuration yes yes
iperf Network bandwidth tool no yes
tcptrace Packet analysis tool no yes

3.0 CPU 介绍

CPU 利用率主要依赖于是什么资源在试图存取.内核调度器将负责调度 2 种资源种类:线程(单一或者多路)和中断.调度器去定义不同资源的不同优先权.以下列表从优先级高到低排列:

Interrupts(译注:中断) - 设备通知内核,他们完成一次数据处理的过程.例子,当一块网卡设备递送网络数据包或者一块硬件提供了一次 IO 请求.

Kernel(System) Processes(译注:内核处理过程) - 所有内核处理过程就是控制优先级别.

User Processes(译注:用户进程) - 这块涉及"userland".所有软件程序都运行在这个 user space.这块在内核调度机制中处于低优先级.

从上面,我们可以看出内核是怎样管理不同资源的.还有几个关键内容需要介绍,以下部分就将介绍 context(译注:上下文切换),run queues(译注:运行队列)以及 utilization(译注:利用率).

3.1 上下文切换

多数现代处理器都能够运行一个进程(单一线程)或者线程.多路超线程处理器有能力运行多个线程.然而,Linux 内核还是把每个处理器核心的双核心芯片作为独立的处理器.比如,以 Linux 内核的系统在一个双核心处理器上,是报告显示为两个独立的处理器.

一个标准的 Linux 内核可以运行 50 至 50,000 的处理线程.在只有一个 CPU 时,内核将调度并均衡每个进程线程.每个线程都分配一个在处理器中被开销的时间额度.一个线程要么就是获得时间额度或已抢先 获得一些具有较高优先级(比如硬件中断),其中较高优先级的线程将从区域重新放置回处理器的队列中.这种线程的转换关系就是我们提到的上下文切换.

每次内核的上下文切换,资源被用于关闭在 CPU 寄存器中的线程和放置在队列中.系统中越多的上下文切换,在处理器的调度管理下,内核将得到更多的工作.

3.2 运行队列

每个 CPU 都维护一个线程的运行队列.理论上,调度器应该不断的运行和执行线程.进程线程不是在 sleep 状态中(译注:阻塞中和等待 IO 中)或就是在可运行状态中.如果 CPU 子系统处于高负荷下,那就意味着内核调度将无法及时响应系统请求.导致结果,可运行状态进程拥塞在运行队列里.当运行队列越来越巨大,进程线程将花费更多 的时间获取被执行.

比较流行的术语就是"load",它提供当前运行队列的详细状态.系统 load 就是指在 CPU 队列中有多少数目的线程,以及其中当前有多少进程线程数目被执行的组合.如果一个双核系统执行了 2 个线程,还有 4 个在运行队列中,则 load 应该为 6. top 这个程序里显示的 load averages 是指 1,5,15 分钟以内的 load 情况.

3.3 CPU 利用率

CPU 利用率就是定义 CPU 使用的百分比.评估系统最重要的一个度量方式就是 CPU 的利用率.多数性能监控工具关于 CPU 利用率的分类有以下几种:

User Time(译注:用户进程时间) - 关于在 user space 中被执行进程在 CPU 开销时间百分比.

System Time(译注:内核线程以及中断时间) - 关于在 kernel space 中线程和中断在 CPU 开销时间百分比.

Wait IO(译注:IO 请求等待时间) - 所有进程线程被阻塞等待完成一次 IO 请求所占 CPU 开销 idle 的时间百

分比.

Idle(译注:空闲) - 一个完整空闲状态的进程在 CPU 处理器中开销的时间百分比.

4.0 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 1-3 个线程.例子,一个双核处理器应该运行队列不要超过 6 个线程.

CPU Utiliation - 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
30% - 35% System Time
0% - 5% Idle Time

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,大量的上下文切换是正常的.

很多 Linux 上的工具可以得到这些状态值,首先就是 vmstat 和 top 这 2 个工具.

4.1 vmstat 工具的使用

vmstat 工具提供了一种低开销的系统性能观察方式.因为 vmstat 本身就是低开销工具,在非常高负荷的服务器上,你需要查看并监控系统的健康情况,在控制窗口还是能够使用 vmstat 输出结果.这个工具运行在 2 种模式下:average 和 sample 模式.sample 模式通过指定间隔时间测量状态值.这个模式对于理解在持续负荷下的性能表现,很有帮助.下面就是

vmstat 运行 1 秒间隔的示例:

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
0 0 104300 16800 95328 72200 0 0 5 26 7 14 4 1 95 0
0 0 104300 16800 95328 72200 0 0 0 24 1021 64 1 1 98 0
0 0 104300 16800 95328 72200 0 0 0 0 1009 59 1 1 98 0
```

Table 1: The vmstat CPU statistics
Field Description
r The amount of threads in the run queue. These are threads that are runnable, but the CPU is not available to execute them.
当前运行队列中线程的数目.代表线程处于可运行状态,但 CPU 还未能执行.
b This is the number of processes blocked and waiting on IO requests to finish.
当前进程阻塞并等待 IO 请求完成的数目
in This is the number of interrupts being processed.
当前中断被处理的数目
cs This is the number of context switches currently happening on the system.
当前 kernel system 中,发生上下文切换的数目
us This is the percentage of user CPU utilization.
CPU 利用率的百分比
sys This is the percentage of kernel and interrupts utilization.
内核和中断利用率的百分比
wa This is the percentage of idle processor time due to the fact that ALL runnable threads

are blocked waiting on IO.
所有可运行状态线程被阻塞在等待 IO 请求的百分比
id This is the percentage of time that the CPU is completely idle.
CPU 空闲时间的百分比

4.2 案例学习:持续的 CPU 利用率

在这个例子中,这个系统被充分利用

```
# vmstat 1
procs memory swap io system cpu
r b swpd free buff cache si so bi bo in cs us sy wa id
3 0 206564 15092 80336 176080 0 0 0 0 718 26 81 19 0 0
2 0 206564 14772 80336 176120 0 0 0 0 758 23 96 4 0 0
1 0 206564 14208 80336 176136 0 0 0 0 820 20 96 4 0 0
1 0 206956 13884 79180 175964 0 412 0 2680 1008 80 93 7 0 0
2 0 207348 14448 78800 175576 0 412 0 412 763 70 84 16 0 0
2 0 207348 15756 78800 175424 0 0 0 0 874 25 89 11 0 0
1 0 207348 16368 78800 175596 0 0 0 0 940 24 86 14 0 0
1 0 207348 16600 78800 175604 0 0 0 0 929 27 95 3 0 2
3 0 207348 16976 78548 175876 0 0 0 2508 969 35 93 7 0 0
4 0 207348 16216 78548 175704 0 0 0 0 874 36 93 6 0 1
4 0 207348 16424 78548 175776 0 0 0 0 850 26 77 23 0 0
2 0 207348 17496 78556 175840 0 0 0 0 736 23 83 17 0 0
0 0 207348 17680 78556 175868 0 0 0 0 861 21 91 8 0 1
```

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在产生对硬件设备的请求.

2,进一步显示某单个应用,user time(us) 经常在 85%或者更多.考虑到较少的上下文切换,这个应用应该还在处理器中被处理.

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.

4.3 案例学习: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
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
4 2 207740 89240 90512 180984 0 0 880 520 443 907 22 10 67 0
5 3 207740 88056 91680 180984 0 0 1168 0 628 1248 12 11 77 0
4 2 207740 86852 92880 180984 0 0 1200 0 654 1505 6 7 87 0
6 1 207740 85736 93996 180984 0 0 1116 0 526 1512 5 10 85 0
```

0 1 207740 84844 94888 180984 0 0 892 0 438 1556 6 4 90 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 利用率分类不均衡.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).

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

4.4 mpstat 工具的使用

如果你的系统运行在多处理器芯片上,你可以使用 mpstat 命令来监控每个独立的芯片.Linux 内核视双核处理器为 2 CPU's,因此一个双核处理器的双内核就报告有 4 CPU's 可用.

mpstat 命令给出的 CPU 利用率统计值大致和 vmstat 一致,但是 mpstat 可以给出基于单个处理器的统计值.

mpstat –P ALL 1
Linux 2.4.21-20.ELsmp (localhost.localdomain) 05/23/2006

05:17:31 PM CPU %user %nice %system %idle intr/s
05:17:32 PM all 0.00 0.00 3.19 96.53 13.27
05:17:32 PM 0 0.00 0.00 0.00 100.00 0.00
05:17:32 PM 1 1.12 0.00 12.73 86.15 13.27
05:17:32 PM 2 0.00 0.00 0.00 100.00 0.00
05:17:32 PM 3 0.00 0.00 0.00 100.00 0.00

4.5 案例学习: 未充分使用的处理量

在这个例子中,为 4 CPU 核心可用.其中 2 个 CPU 主要处理进程运行(CPU 0 和 1).第 3 个核心处理所有内核和其他系统功能(CPU 3).第 4 个核心处于 idle(CPU 2).

使用 top 命令可以看到有 3 个进程差不多完全占用了整个 CPU 核心.

top -d 1
top - 23:08:53 up 8:34, 3 users, load average: 0.91, 0.37, 0.13
Tasks: 190 total, 4 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 75.2% us, 0.2% sy, 0.0% ni, 24.5% id, 0.0% wa, 0.0% hi, 0.0%
si
Mem: 2074736k total, 448684k used, 1626052k free, 73756k buffers
Swap: 4192956k total, 0k used, 4192956k free, 259044k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
15957 nobody 25 0 2776 280 224 R 100 20.5 0:25.48 php
15959 mysql 25 0 2256 280 224 R 100 38.2 0:17.78 mysqld
15960 apache 25 0 2416 280 224 R 100 15.7 0:11.20 httpd
15901 root 16 0 2780 1092 800 R 1 0.1 0:01.59 top
1 root 16 0 1780 660 572 S 0 0.0 0:00.64 init

mpstat –P ALL 1
Linux 2.4.21-20.ELsmp (localhost.localdomain) 05/23/2006

05:17:31 PM CPU %user %nice %system %idle intr/s
05:17:32 PM all 81.52 0.00 18.48 21.17 130.58
05:17:32 PM 0 83.67 0.00 17.35 0.00 115.31

05:17:32 PM 1 80.61 0.00 19.39 0.00 13.27
05:17:32 PM 2 0.00 0.00 16.33 84.66 2.01
05:17:32 PM 3 79.59 0.00 21.43 0.00 0.00

05:17:32 PM CPU %user %nice %system %idle intr/s
05:17:33 PM all 85.86 0.00 14.14 25.00 116.49
05:17:33 PM 0 88.66 0.00 12.37 0.00 116.49
05:17:33 PM 1 80.41 0.00 19.59 0.00 0.00
05:17:33 PM 2 0.00 0.00 0.00 100.00 0.00
05:17:33 PM 3 83.51 0.00 16.49 0.00 0.00

05:17:33 PM CPU %user %nice %system %idle intr/s
05:17:34 PM all 82.74 0.00 17.26 25.00 115.31
05:17:34 PM 0 85.71 0.00 13.27 0.00 115.31
05:17:34 PM 1 78.57 0.00 21.43 0.00 0.00
05:17:34 PM 2 0.00 0.00 0.00 100.00 0.00
05:17:34 PM 3 92.86 0.00 9.18 0.00 0.00

05:17:34 PM CPU %user %nice %system %idle intr/s
05:17:35 PM all 87.50 0.00 12.50 25.00 115.31
05:17:35 PM 0 91.84 0.00 8.16 0.00 114.29
05:17:35 PM 1 90.82 0.00 10.20 0.00 1.02
05:17:35 PM 2 0.00 0.00 0.00 100.00 0.00
05:17:35 PM 3 81.63 0.00 15.31 0.00 0.00

你也可以使用 ps 命令通过查看 PSR 这列,检查哪个进程在占用了哪个 CPU.

while :; do ps -eo pid,ni,pri,pcpu,psr,comm | grep 'mysqld'; sleep 1;
done
PID NI PRI %CPU PSR COMMAND
15775 0 15 86.0 3 mysqld
PID NI PRI %CPU PSR COMMAND
15775 0 14 94.0 3 mysqld
PID NI PRI %CPU PSR COMMAND
15775 0 14 96.6 3 mysqld
PID NI PRI %CPU PSR COMMAND
15775 0 14 98.0 3 mysqld
PID NI PRI %CPU PSR COMMAND
15775 0 14 98.8 3 mysqld
PID NI PRI %CPU PSR COMMAND
15775 0 14 99.3 3 mysqld

4.6 结论

监控 CPU 性能由以下几个部分组成:

1,检查 system 的运行队列,以及确定不要超出每个处理器 3 个可运行状态线程的限制.

2,确定 CPU 利用率中 user/system 比例维持在 70/30

3,当 CPU 开销更多的时间在 system mode,那就说明已经超负荷并且应该尝试重新调度优先级

4,当 I/O 处理得到增长,CPU 范畴的应用处理将受到影响

Linux System and Performance Monitoring(Memory 篇)

08 月 11th, 2009 Posted in Linux, Monitor

作者:tonnyom
原载: http://www.sanotes.net/html/y2009/376.html

Linux System and Performance Monitoring(Memory 篇)
Date: 2009.07.21
Author: Darren Hoch
译: Tonnyom[AT]hotmail.com

5.0 Virtual Memory 介绍

虚拟内存就是采用硬盘对物理内存进行扩展,所以对可用内存的增加是要相对在一个有效范围内的.内核会写当前未使用内存块的内容到硬盘上,此时这部分 内存被用于其它用途.当再一次需要原始内容时,此时再读回到内存中.这对于用户来说,是完全透明的;在 Linux 下运行的程序能够看到,也仅仅是大量的可用内存,同时也不会留意到,偶尔还有部分是驻留在磁盘上的.当然,在硬盘上进行读和写,都是很慢的(大约会慢上千 倍),相对于使用真实内存的话,因此程序无法运行的更快.用硬盘的一部分作为 Virtual Memory,这就被称为"swap space"(译注:交换空间).

5.1 Virtual Memory Pages

虚拟内存被分为很多 pages(译注:页),在 X86 架构中,每个虚拟内存页为 4KB.当内核写内存到磁盘或者读磁盘到内存,这就是一次写内存到页的过程.内核通常是在 swap 分区和文件系统之间进行这样的操作.

5.2 Kernel Memory Paging

内存分页在正常情况下总是活跃的,与 memory swapping(译注:内存交换)之间不要搞错了.内存分页是指内核会定期将内存中的数据同步到硬盘,这个过程就是 Memory Paging.日复一日,应用最终将会消耗掉所有的内存空间.考虑到这点,内核就必须经常扫描内存空间并且收回其中未被使用的内存页,然后再重新分配内存 空间给其他应用使用.

5.3 The Page Frame Reclaim Algorithm(PFRA)(译注:页框回收算法)

PFRA 就是 OS 内核用来回收并释放内存空间的算法.PFRA 选择哪个内存页被释放是基于内存页类型的.页类型有以下几种:

Unreclaimable –锁定的，内核保留的页面
Swappable –匿名的内存页
Syncable –通过硬盘文件备份的内存页
Discardable –静态页和被丢弃的页

除了第一种（Unreclaimable）之外其余的都可以被 PFRA 进行回收.

与 PFRA 相关的,还包括 kswapd 内核线程以及 Low On Memory Reclaiming(LMR 算法) 这 2 种进程和实现.

5.4 kswapd

kswapd 进程负责确保内存空间总是在被释放中.它监控内核中的 pages_high 和 pages_low 阀值.如果空闲内存的数值低于 pages_low,则每次 kswapd 进程启动扫描并尝试释放 32 个 free pages.并一直重复这个过程,直到空闲内存的数值高于 pages_high.

kswapd 进程完成以下几个操作:

1,如果该页处于未修改状态,则将该页放置回空闲列表中.
2,如果该页处于已修改状态并可备份回文件系统,则将页内容写入到磁盘.
3,如果该页处于已修改状态但没有任何磁盘备份,则将页内容写入到 swap device.

ps -ef | grep kswapd
root 30 1 0 23:01 ? 00:00:00 [kswapd0]

5.5 Kernel Paging with pdflush

pdflush 进程负责将内存中的内容和文件系统进行同步操作.也就是说,当一个文件在内存中进行修改后,pdflush 将负责写回到磁盘上.

ps -ef | grep pdflush
root 28 3 0 23:01 ? 00:00:00 [pdflush]
root 29 3 0 23:01 ? 00:00:00 [pdflush]

当内存中存在 10% 的脏页,pdflush 将被启动同步脏页回文件系统里.这个参数值可以通过 vm.dirty_background_ratio 来进行调整.

(译注:
Q:什么是脏页?
A:由于内存中页缓存的缓存作用,写操作实际上都是延迟的.当页缓存中的数据比磁盘存储的数据还要更新时,那么该数据就被称做脏页.)

sysctl -n vm.dirty_background_ratio
10

在多数环境下,Pdflush 与 PFRA 是独立运行的,当内核调用 LMR 时,LMR 就触发 pdflush 将脏页写入到磁盘里.

++
在 2.4 内核下,一个高负荷的内存环境中,系统将遇到交换过程中不断的崩溃.这是因为 PFRA 从一个运行进程中,偷取其中一个内存页并尝试使用.导致结果就是,这个进程如果要回收那个页时,要是没有就会尝试再去偷取这个页,这样一来,就越来越糟糕 了.在 2.6 内核下,使用"Swap token"修复了这个 BUG,用来防止 PFRA 不断从一个进程获取同一个页.
++

5.6 案例学习:大量的入口 I/O

vmstat 工具报告里除了 CPU 使用情况,还包括了虚拟内存.以下就是 vmstat 输出中关于虚拟内存的部分:

Table 2: The vmstat Memory Statistics
Field Description
Swapd The amount of virtual memory in KB currently in use. As free memory reaches low thresholds, more data is paged to the swap device.
当前虚拟内存使用的总额(单位:KB).空闲内存达到最低的阀值时,更多的数据被转换成页到交换设备中.
Free The amount of physical RAM in kilobytes currently available to running applications.
当前内存中可用空间字节数.
Buff The amount of physical memory in kilobytes in the buffer cache as a result of read() and write() operations.
当前内存中用于 read()和 write()操作的缓冲区中缓存字节数
Cache The amount of physical memory in kilobytes mapped into process address space.
当前内存中映射到进程地址空间字节数
So The amount of data in kilobytes written to the swap disk.

写入交换空间的字节数总额
Si The amount of data in kilobytes written from the swap disk back into RAM.
从交换空间写回内存的字节数总额
Bo The amount of disk blocks paged out from the RAM to the filesystem or swap device.
磁盘块页面从内存到文件或交换设备的总额
Bi The amount of disk blocks paged into RAM from the filesystem or swap device.
磁盘块页面从文件或交换设备到内存的总额

以下 vmstat 的输出结果,就是演示一个在 I/O 应用中,虚拟内存在高负荷情况下的环境

```
# vmstat 3
procs memory swap io system cpu
r b swpd free buff cache si so bi bo in cs us sy id wa
3 2 809192 261556 79760 886880 416 0 8244 751 426 863 17 3 6 75
0 3 809188 194916 79820 952900 307 0 21745 1005 1189 2590 34 6 12 48
0 3 809188 162212 79840 988920 95 0 12107 0 1801 2633 2 2 3 94
1 3 809268 88756 79924 1061424 260 28 18377 113 1142 1694 3 5 3 88
1 2 826284 17608 71240 1144180 100 6140 25839 16380 1528 1179 19 9 12 61
2 1 854780 17688 34140 1208980 1 9535 25557 30967 1764 2238 43 13 16 28
0 8 867528 17588 32332 1226392 31 4384 16524 27808 1490 1634 41 10 7 43
4 2 877372 17596 32372 1227532 213 3281 10912 3337 678 932 33 7 3 57
1 2 885980 17800 32408 1239160 204 2892 12347 12681 1033 982 40 12 2 46
5 2 900472 17980 32440 1253884 24 4851 17521 4856 934 1730 48 12 13 26
1 1 904404 17620 32492 1258928 15 1316 7647 15804 919 978 49 9 17 25
4 1 911192 17944 32540 1266724 37 2263 12907 3547 834 1421 47 14 20 20
1 1 919292 17876 31824 1275832 1 2745 16327 2747 617 1421 52 11 23 14
5 0 925216 17812 25008 1289320 12 1975 12760 3181 772 1254 50 10 21 19
0 5 932860 17736 21760 1300280 8 2556 15469 3873 825 1258 49 13 24 15
```

根据观察值,我们可以得到以下结论:

1,大量的 disk pages(bi)被写入内存,很明显在进程地址空间里,数据缓存(cache)也在不断的增长.

2,在这个时间点上,空闲内存(free) 始终保持在 17MB,即使数据从硬盘读入而在消耗 RAM.

3,为了维护空闲列表, kswapd 从读/写缓存区(buff)中获取内存并分配到空闲列表里.很明显可以看到 buffer cache(buff) 在逐渐的减少中.

4, 同时 kswapd 进程不断的写脏页到 swap device(so)时,很明显虚拟内存的利用率是在逐渐的增加中(swpd).

5.7 结论

监控虚拟内存性能由以下几个部分组成:

1,当系统中出现较少的页错误,获得最好的响应时间,是因为 memory caches(译注:内存高速缓存)比 disk caches 更快(译注:磁盘高速缓存).

2,较少的空闲内存,是件好事情,那意味着缓存的使用更有效率.除非在不断的写入 swap device 和 disk.

3,如果系统不断报告,swap device 总是繁忙中,那就意味着内存已经不足,需要升级了.

Linux System and Performance Monitoring(I/O 篇)

08 月 12th, 2009 Posted in Linux, Monitor

作者:tonnyom
原载: http://www.sanotes.net/html/y2009/381.html

Linux System and Performance Monitoring(I/O 篇)
Date: 2009.07.21
Author: Darren Hoch
译: Tonnyom[AT]hotmail.com

6.0 I/O 监控介绍

磁盘 I/O 子系统是 Linux 系统中最慢的部分.这个主要是归于 CPU 到物理操作磁盘之间距离(译注:盘片旋转以及寻道).如果拿读取磁盘和内存的时间作比较就是分钟级到秒级,这就像 7 天和 7 分钟的区别.因此本质上,Linux 内核就是要最低程度的降低 I/O 数.本章将诉述内核在磁盘和内存之间处理数据的这个过程中,哪些地方会产生 I/O.

6.1 读和写数据 - 内存页

Linux 内核将硬盘 I/O 进行分页,多数 Linux 系统的默认页大小为 4K.读和写磁盘块进出到内存都为 4K 页大小.你可以使用 time 这个命令加-v 参数,来检查你系统中设置的页大小:

```
# /usr/bin/time -v date
<snip>
Page size (bytes): 4096
<snip>
```

6.2 Major and Minor Page Faults(译注:主要页错误和次要页错误)

Linux,类似多数的 UNIX 系统,使用一个虚拟内存层来映射硬件地址空间.当一个进程被启动,内核先扫描 CPU caches 和物理内存.如果进程需要的数据在这 2 个地方都没找到,就需要从磁盘上读取,此时内核过程就是 major page fault(MPF).MPF 要求磁盘子系统检索页并缓存进 RAM.

一旦内存页被映射进内存的 buffer cache(buff)中,内核将尝试从内存中读取或写入,此时内核过程就是 minor page fault(MnPF).与在磁盘上操作相比,MnPF 通过反复使用内存中的内存页就大大的缩短了内核时间.

以下的例子,使用 time 命令验证了,当进程启动后,MPF 和 MnPF 的变化情况.第一次运行进程,MPF 会更多:

```
# /usr/bin/time -v evolution
<snip>
Major (requiring I/O) page faults: 163
Minor (reclaiming a frame) page faults: 5918
<snip>
```

第二次再运行时,内核已经不需要进行 MPF 了,因为进程所需的数据已经在内存中:

```
# /usr/bin/time -v evolution
<snip>
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 5581
<snip>
```

6.3 The File Buffer Cache(译注:文件缓存区)

文件缓存区就是指,内核将 MPF 过程最小化,MnPF 过程最大化.随着系统不断的产生 I/O,buffer cache 也将不断的增加.直到内存不够,以及系统需要释放老的内存页去给其他用户进程使用时,系统就会丢弃这些内存页.结果是,很多 sa(译注:系统管 理员)对系统中过少的 free memory(译注:空闲内存)表示担心,实际上这是系统更高效的在使用 caches.

以下例子,是查看/proc/meminfo 文件:

```
# cat /proc/meminfo
MemTotal: 2075672 kB
MemFree: 52528 kB
Buffers: 24596 kB
Cached: 1766844 kB
<snip>
```

可以看出,这个系统总计有 2GB (Memtotal)的可用内存.当前的空闲内存为 52MB (MemFree),有 24 MB 内存被分配磁盘写操作(Buffers),还有 1.7 GB 页用于读磁盘(Cached).

内核这样是通过 MnPF 机制,而不代表所有的页都是来自磁盘.通过以上部分,我们不可能确认系统是否处于瓶颈中.

6.4 Type of Memory Pages

在 Linux 内核中,memory pages 有 3 种,分别是:

1,Read Pages - 这些页通过 MPF 从磁盘中读入,而且是只读.这些页存在于 Buffer Cache 中以及包括不能够修改的静态文件,二进制文件,还有库文件.当内核需要它们时,将读取到内存中.如果内存不足,内核将释放它们回空闲列表中.程 序再次请求时,则通过 MPF 再次读回内存.

2,Dirty Pages - 这些页是内核在内存中已经被修改过的数据页.当这些页需要同步回磁盘上,由 pdflush 负责写回磁盘.如果内存不足,kswapd (与 pdflush 一起)将这些页写回到磁盘上并释放更多的内存.

3,Anonymous Pages - 这些页属于某个进程,但是没有任何磁盘文件和它们有关.他们不能和同步回磁盘.如果内存不足,kswapd 将他们写入 swap 分区上并释放更多的内存("swapping" pages).

6.5 Writing Data Pages Back to Disk

应用程序有很多选择可以写脏页回磁盘上,可通过 I/O 调度器使用 fsync() 或 sync() 这样的系统函数来实现立即写回.如果应用程序没有调用以上函数,pdflush 进程会定期与磁盘进行同步.

```
# ps -ef | grep pdflush
root 186 6 0 18:04 ? 00:00:00 [pdflush]
```

7.0 监控 I/O

当觉得系统中出现了 I/O 瓶颈时,可以使用标准的监控软件来查找原因.这些工具包括了 top,vmstat,iostat,sar.它们的输出结果一小部分是很相似,不过每个也都提供了各自对于性能不同方面的解释.以下章节就将讨论哪些情况会导致 I/O 瓶颈的出现.

7.1 Calculating IO's Per Second(译注:IOPS 的计算)

每个 I/O 请求到磁盘都需要若干时间.主要是因为磁盘的盘边必须旋转,机头必须寻道.磁盘的旋转常常被称为"rotational delay"(RD),机头的移动称为"disk seek"(DS).一个 I/O 请求所需的时间计算就是 DS 加上 RD.磁盘的 RD 基于设备自身 RPM 单位值(译注:RPM 是 Revolutions Perminute 的缩写,是转/每分钟,代表了硬盘的转速).一个 RD 就是一个盘片旋转的

半圆.如何计算一个 10K RPM 设备的 RD 值呢:

1, 10000 RPM / 60 seconds (10000/60 = 166 RPS)
2, 转换为 166 分之 1 的值(1/166 = 0.006 seconds/Rotation)
3, 单位转换为毫秒(6 MS/Rotation)
4, 旋转半圆的时间(6/2 = 3MS) 也就是 RD
5, 加上平均 3 MS 的寻道时间 (3MS + 3MS = 6MS)
6, 加上 2MS 的延迟(6MS + 2MS = 8MS)
7, 1000 MS / 8 MS (1000/8 = 125 IOPS)

每次应用程序产生一个 I/O,在 10K RPM 磁盘上都要花费平均 8MS.在这个固定时间里,磁盘将尽可能且有效率在进行读写磁盘.IOPS 可以计算出大致的 I/O 请求数,10K RPM 磁盘有能力提供 120-150 次 IOPS.评估 IOPS 的效能,可用每秒读写 I/O 字节数除以每秒读写 IOPS 数得出.

7.2 Random vs Sequential I/O(译注:随机/顺序 I/O)

per I/O 产生的 KB 字节数是与系统本身 workload 相关的,有 2 种不同 workload 的类型,它们是 sequential 和 random.

7.2.1 Sequential I/O(译注:顺序 IO)

iostat 命令提供信息包括 IOPS 和每个 I/O 数据处理的总额.可使用 iostat -x 查看.顺序的 workload 是同时读顺序请求大量的数据.这包括的应用,比如有商业数据库(database)在执行大量的查询和流媒体服务.在这个 workload 中,KB per I/O 的比率应该是很高的.Sequential workload 是可以同时很快的移动大量数据.如果每个 I/O 都节省了时间,那就意味了能带来更多的数据处理.

iostat -x 1

avg-cpu: %user %nice %sys %idle
0.00 0.00 57.1 4 42.86

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 12891.43 0.00 105.71 0.00 1 06080.00 0.00 53040.00 1003.46 1099.43 3442.43 26.49 280.00
/dev/sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda2 0.00 12857.14 0.00 5.71 0.00 105782.86 0.00 52891.43 18512.00 559.14 780.00 490.00 280.00
/dev/sda3 0.00 34.29 0.00 100.00 0.00 297.14 0.00 148.57 2.97 540.29 594.57 24.00 240.00

avg-cpu: %user %nice %sys %idle
0.00 0.00 23.53 76.47

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 17320.59 0.00 102.94 0.00 142305.88 0.00 71152.94 1382.40 6975.29 952.29 28.57 294.12
/dev/sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda2 0.00 16844.12 0.00 102.94 0.00 138352.94 0.00 69176.47 1344.00 6809.71 952.29 28.57 294.12
/dev/sda3 0.00 476.47 0.00 0.00 0.00 952.94 0.00 1976.47 0.00 165.59 0.00 0.00 276.47

评估 IOPS 的效能,可用每秒读写 I/O 字节数除以每秒读写 IOPS 数得出,比如
rkB/s 除以 r/s
wkB/s 除以 w/s

53040/105 = 505KB per I/O

71152/102 = 697KB per I/O
在上面例子可看出,每次循环下,/dev/sda 的 per I/O 都在增加.

7.2.2 Random I/O(译注:随机 IO)

Random 的 worklaod 环境下,不依赖于数据大小的多少,更多依赖的是磁盘的 IOPS 数.Web 和 Mail 服务就是典型的 Random workload.I/O 请求内容都很小.Random workload 是同时每秒会有更多的请求数产生.所以,磁盘的 IOPS 数是关键.

iostat -x 1

avg-cpu: %user %nice %sys %idle
2.04 0.00 97.96 0.00

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 633.67 3.06 102.31 24.49 5281.63 12.24 2640.82 288.89 73.67 113.89 27.22 50.00
/dev/sda1 0.00 5.10 0.00 2.04 0.00 57.14 0.00 28.57 28.00 1.12 55.00 55.00 11.22
/dev/sda2 0.00 628.57 3.06 100.27 24.49 5224.49 12.24 2612.24 321.50 72.55 121.25 30.63 50.00
/dev/sda3 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

avg-cpu: %user %nice %sys %idle
2.15 0.00 97.85 0.00

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 41.94 6.45 130.98 51.61 352.69 25.81 3176.34 19.79 2.90 286.32 7.37 15.05
/dev/sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda2 0.00 41.94 4.30 130.98 34.41 352.69 17.20 3176.34 21.18 2.90 320.00 8.24 15.05
/dev/sda3 0.00 0.00 2.15 0.00 17.20 0.00 8.60 0.00 8.00 0.00 0.00 0.00 0.00

计算方式和之前的公式一致:

2640/102 = 23KB per I/O
3176/130 = 24KB per I/O

(译注:对于顺序 I/O 来说,主要是考虑读取大量数据的能力即 KB per request.对于随机 I/O 系统,更需要考虑的是 IOPS 值)

7.3 When Virtual Memory Kills I/O

如果系统没有足够的 RAM 响应所有的请求,就会使用到 SWAP device.就像使用文件系统 I/O,使用 SWAP device 代价也很大.如果系统已经没有物理内存可用,那就都在 SWAP disk 上创建很多很多的内存分页,如果同一文件系统的数据都在尝试访问 SWAP device,那系统将遇到 I/O 瓶颈.最终导致系统性能的全面崩溃.如果内存页不能够及时读或写磁盘,它们就一直保留在 RAM 中.如果保留时间太久,内核又必须释放内存空间.问题来 了,I/O 操作都被阻塞住了,什么都没做就被结束了,不可避免地就出现 kernel panic 和 system crash.

下面的 vmstat 示范了一个内存不足情况下的系统:

procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
17 0 1250 3248 45820 1488472 30 132 992 0 2437 7657 23 50 0 23
11 0 1376 3256 45820 1488888 57 245 416 0 2391 7173 10 90 0 0
12 0 1582 1688 45828 1490228 63 131 1348 76 2432 7315 10 90 0 10
12 2 3981 1848 45468 1489824 185 56 2300 68 2478 9149 15 12 0 73
14 2 10385 2400 44484 1489732 0 87 1112 20 2515 11620 0 12 0 88
14 2 12671 2280 43644 1488816 76 51 1812 204 2546 11407 20 45 0 35

这个结果可看出,大量的读请求回内存(bi),导致了空闲内存在不断的减少(free).这就使得系统写入 swap device 的块数目(so)和 swap 空间(swpd)在不断增加.同时看到 CPU WIO time(wa)百分比很大.这表明 I/O 请求已经导致 CPU 开始效率低下.

要看 swaping 对磁盘的影响,可使用 iostat 检查 swap 分区

iostat -x 1

avg-cpu: %user %nice %sys %idle
0.00 0.00 100.00 0.00

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 1766.67 4866.67 1700.00 38933.33 31200.00 19466.67 15600.00 10.68 6526.67 100.56 5.08 3333.33
/dev/sda1 0.00 933.33 0.00 0.00 0.00 7733.33 0.00 3866.67 0.00 20.00 2145.07 7.37 200.00
/dev/sda2 0.00 0.00 4833.33 0.00 38666.67 533.33 19333.33 266.67 8.11 373.33 8.07 6.90 87.00
/dev/sda3 0.00 833.33 33.33 1700.00 266.67 22933.33 133.33 11466.67 13.38 6133.33 358.46 11.35 1966.67

在这个例子中,swap device(/dev/sda1) 和 file system device(/dev/sda3)在互相作用于 I/O. 其中任一个会有很高写请求(w/s),也会有很高 wait time(await),或者较低的服务时间比率(svctm).这表明 2 个分区之间互有联系,互有影响.

7.4 结论

I/O 性能监控包含了以下几点:

1,当 CPU 有等待 I/O 情况时,那说明磁盘处于超负荷状态.
2,计算你的磁盘能够承受多大的 IOPS 数.
3,确定你的应用是属于随机或者顺序读取磁盘.
4,监控磁盘慢需要比较 wait time(await) 和 service time(svctm).
5,监控 swap 和系统分区,要确保 virtual memory 不是文件系统 I/O 的瓶颈.

Linux System and Performance Monitoring(Network 篇)

08 月 13th, 2009 Posted in Linux, Monitor

作者:tonnyom
原载: http://www.sanotes.net/html/y2009/390.html

Linux System and Performance Monitoring(Network 篇)
Date: 2009.07.21
Author: Darren Hoch
译: Tonnyom[AT]hotmail.com

8.0 Network 监控介绍

在所有的子系统监控中,网络是最困难的.这主要是由于网络概念很抽象.当监控系统上的网络性能,这有太多因素.这些因素包括了延迟,冲突,拥挤和数据包丢失.

这个章节讨论怎么样检查 Ethernet(译注:网卡),IP,TCP 的性能.

8.1 Ethernet Configuration Settings(译注:网卡配置的设置)

除非很明确的指定,几乎所有的网卡都是自适应网络速度.当一个网络中有很多不同的网络设备时,会各自采用不同的速率和工作模式.

多数商业网络都运行在 100 或 1000BaseTX.使用 ethtool 可以确定这个系统是处于那种速率.

以下的例子中,是一个有 100BaseTX 网卡的系统,自动协商适应至 10BaseTX 的情况.

```
# ethtool eth0
Settings for eth0:
Supported ports: [ TP MII ]
Supported link modes: 10baseT/Half 10baseT/Full
100baseT/Half 100baseT/Full
Supports auto-negotiation: Yes
Advertised link modes: 10baseT/Half 10baseT/Full
100baseT/Half 100baseT/Full
Advertised auto-negotiation: Yes
Speed: 10Mb/s
Duplex: Half
Port: MII
PHYAD: 32
Transceiver: internal
Auto-negotiation: on
Supports Wake-on: pumbg
Wake-on: d
Current message level: 0x00000007 (7)
Link detected: yes
```

以下示范例子中,如何强制网卡速率调整至 100BaseTX:

```
# ethtool -s eth0 speed 100 duplex full autoneg off

# ethtool eth0
```

Settings for eth0:
Supported ports: [TP MII]
Supported link modes: 10baseT/Half 10baseT/Full
100baseT/Half 100baseT/Full
Supports auto-negotiation: Yes
Advertised link modes: 10baseT/Half 10baseT/Full
100baseT/Half 100baseT/Full
Advertised auto-negotiation: No
Speed: 100Mb/s
Duplex: Full
Port: MII
PHYAD: 32
Transceiver: internal
Auto-negotiation: off
Supports Wake-on: pumbg
Wake-on: d
Current message level: 0x00000007 (7)
Link detected: yes

8.2 Monitoring Network Throughput(译注:网络吞吐量监控)

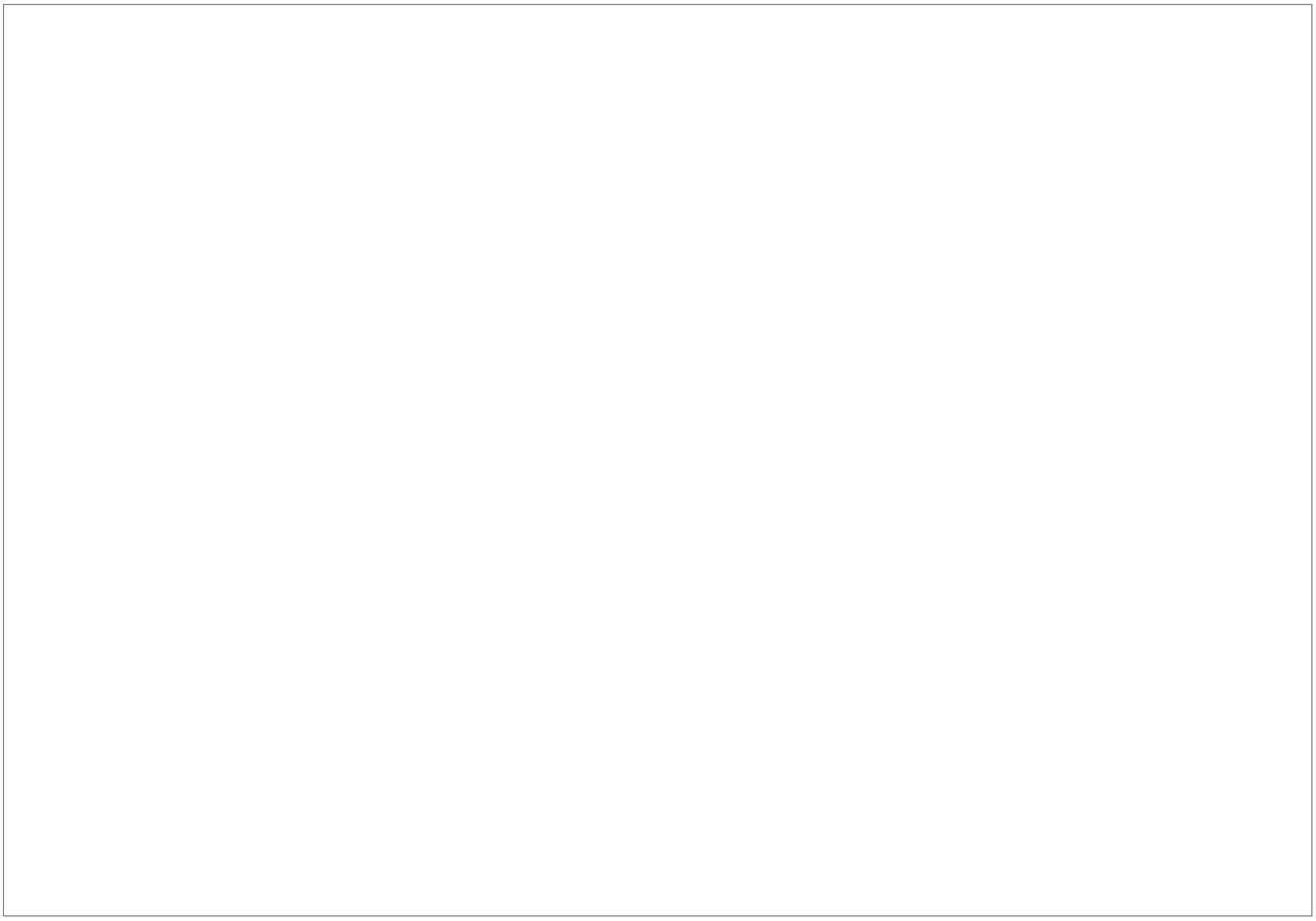
接口之间的同步并不意味着仅仅有带宽问题.重要的是,如何管理并优化,这 2 台主机之间的交换机,网线,或者路由器.测试网络吞吐量最好的方式就是,在这 2 个系统之间互相发送数据传输并统计下来,比如延迟和速度.

8.2.0 使用 iptraf 查看本地吞吐量

iptraf 工具(http://iptraf.seul.org),提供了每个网卡吞吐量的仪表盘.

#iptraf -d eth0

Figure 1: Monitoring for Network Throughput



从输出中可看到,该系统发送传输率(译注:Outgoing rates)为 61 mbps,这对于 100 mbps 网络来说,有点慢.

8.2.1 使用 netperf 查看终端吞吐量

不同于 iptraf 被动的在本地监控流量,netperf 工具可以让管理员,执行更加可控的吞吐量监控.对于确定从客户端工作站到一个高负荷的服务器端(比如 file 或 web server),它们之间有多少吞吐量是非常有帮助的.netperf 工具运行的是 client/server 模式.

完成一个基本可控吞吐量测试,首先 netperf server 必须运行在服务器端系统上:

server# netserver
Starting netserver at port 12865
Starting netserver at hostname 0.0.0.0 port 12865 and family AF_UNSPEC

netperf 工具可能需要进行多重采样.多数基本测试就是一次标准的吞吐量测试.以下例子就是,一个 LAN(译注:局域网) 环境下,从 client 上执行一次 30 秒的 TCP 吞吐量采样:

从输出可看出,该网络的吞吐量大致在 89 mbps 左右.server(192.168.1.215) 与 client 在同一 LAN 中.这对于 100 mbps 网络来说,性能非常好.

client# netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.230 (192.168.1.230) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput

bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 30.02 89.46

从 LAN 切换到具备 54G(译注:Wireless-G 是未来 54Mbps 无线网联网标准)无线网络路由器中,并在 10 英尺范围内测试时.该吞吐量就急剧的下降.在最大就为 54 MBits 的可能下,笔记本电脑可实现总吞吐量就为 14 MBits.

client# netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.215 (192.168.1.215) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 30.10 14.09

如果在 50 英尺范围内呢,则进一步会下降至 5 MBits.

netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.215 (192.168.1.215) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 30.64 5.05

如果从 LAN 切换到互联网上,则吞吐量跌至 1 Mbits 下了.

netperf -H litemail.org -p 1500 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
litemail.org (72.249.104.148) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 31.58 0.93

最后是一个 VPN 连接环境,这是所有网络环境中最槽糕的吞吐量了.

netperf -H 10.0.1.129 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
10.0.1.129 (10.0.1.129) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 31.99 0.51

另外,netperf 可以帮助测试每秒总计有多少的 TCP 请求和响应数.通过建立单一 TCP 连接并顺序地发送多个

请求/响应(ack 包来回在 1 个 byte 大小).有点类似于 RDBMS 程序在执行多个交易或者邮件服务器在同一个连接管道中发送邮件.

以下例子在 30 秒的持续时间内,模拟 TCP 请求/响应:

```
client# netperf -t TCP_RR -H 192.168.1.230 -l 30
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET
to 192.168.1.230 (192.168.1.230) port 0 AF_INET
Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 1 1 30.00 4453.80
16384 87380
```

在输出中看出,这个网络支持的处理速率为每秒 4453 psh/ack(包大小为 1 byte).这其实是理想状态下,因为实际情况时,多数 requests(译注:请求),特别是 responses(译注:响应),都大于 1 byte.

现实情况下,netperf 一般 requests 默认使用 2K 大小,responses 默认使用 32K 大小:

```
client# netperf -t TCP_RR -H 192.168.1.230 -l 30 -- -r 2048,32768
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.230 (192.168.1.230) port 0 AF_INET
Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 2048 32768 30.00 222.37
16384 87380
```

这个处理速率减少到了每秒 222.

8.2.2 使用 iperf 评估网络效率

基于都是需要在 2 端检查连接情况下,iperf 和 netperf 很相似.不同的是,iperf 更深入的通过 windows size 和 QOS 设备来检查 TCP/UDP 的效率情况.这个工具,是给需要优化 TCP/IP stacks 以及测试这些 stacks 效率的管理员们量身定做的.

iperf 作为一个二进制程序,可运行在 server 或者 client 任一模式下.默认使用 50001 端口.

首先启动 server 端(192.168.1.215):

```
server# iperf -s -D
Running Iperf Server as a daemon
The Iperf daemon process ID : 3655
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

在以下例子里,一个无线网络环境下,其中 client 端重复运行 iperf,用于测试网络的吞吐量情况.这个环境假定处于被充分利用状态,很多主机都在下载 ISO images 文件.

首先 client 端连接到 server 端(192.168.1.215),并在总计 60 秒时间内,每 5 秒进行一次带宽测试的采样.

client# iperf -c 192.168.1.215 -t 60 -i 5

Client connecting to 192.168.1.215, TCP port 5001
TCP window size: 25.6 KByte (default)
---
[3] local 192.168.224.150 port 51978 connected with
192.168.1.215 port 5001
[ID] Interval Transfer Bandwidth
[3] 0.0- 5.0 sec 6.22 MBytes 10.4 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 5.0-10.0 sec 6.05 MBytes 10.1 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 10.0-15.0 sec 5.55 MBytes 9.32 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 15.0-20.0 sec 5.19 MBytes 8.70 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 20.0-25.0 sec 4.95 MBytes 8.30 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 25.0-30.0 sec 5.21 MBytes 8.74 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 30.0-35.0 sec 2.55 MBytes 4.29 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 35.0-40.0 sec 5.87 MBytes 9.84 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 40.0-45.0 sec 5.69 MBytes 9.54 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 45.0-50.0 sec 5.64 MBytes 9.46 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 50.0-55.0 sec 4.55 MBytes 7.64 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 55.0-60.0 sec 4.47 MBytes 7.50 Mbits/sec
[ID] Interval Transfer Bandwidth
[3] 0.0-60.0 sec 61.9 MBytes 8.66 Mbits/sec

这台主机的其他网络传输,也会影响到这部分的带宽采样.所以可以看到总计 60 秒时间内,都在 4 - 10 MBits 上下起伏.

除了 TCP 测试之外,iperf 的 UDP 测试主要是评估包丢失和抖动.

接下来的 iperf 测试,是在同样的 54Mbit G 标准无线网络中.在早期的示范例子中,目前的吞吐量只有 9 Mbits.

iperf -c 192.168.1.215 -b 10M
WARNING: option -b implies udp testing

Client connecting to 192.168.1.215, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 107 KByte (default)

```
------------------------------------------------------------
[ 3] local 192.168.224.150 port 33589 connected with 192.168.1.215 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0-10.0 sec 11.8 MBytes 9.90 Mbits/sec
[ 3] Sent 8420 datagrams
[ 3] Server Report:
[ ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[ 3] 0.0-10.0 sec 6.50 MBytes 5.45 Mbits/sec 0.480 ms 3784/ 8419 (45%)
[ 3] 0.0-10.0 sec 1 datagrams received out-of-order
```

从输出中可看出,在尝试传输 10M 的数据时,实际上只产生了 5.45M.却有 45% 的包丢失.

8.3 Individual Connections with tcptrace

tcptrace 工具提供了对于某一具体连接里,详细的 TCP 相关信息.该工具使用 libcap 来分析某一具体 TCP sessions.该工具汇报的信息,有时很难在某一 TCP stream 被发现.这些信息

包括了有:

1,TCP Retransmissions(译注:IP 转播) - 所有数据大小被发送所需的包总额
2,TCP Windows Sizes - 连接速度慢与小的 windows sizes 有关
3,Total throughput of the connection - 连接的吞吐量
4,Connection duration - 连接的持续时间

8.3.1 案例学习 - 使用 tcptrace

tcptrace 工具可能已经在部分 Linux 发布版中有安装包了,该文作者通过网站,下载的是源码安装包:http://dag.wieers.com/rpm/packages /tcptrace.tcptrace 需要 libcap 基于文件输入方式使用.在 tcptrace 没有选项的情况下,默认每个唯一的连接过程都将被捕获.

以下例子是,使用 libcap 基于输入文件为 bigstuff:

```
# tcptrace bigstuff
1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov 4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:01.634065, 89413 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
1: 192.168.1.60:pcanywherestat - 192.168.1.102:2571 (a2b) 404> 450<
2: 192.168.1.60:3356 - ftp.strongmail.net:21 (c2d) 35> 21<
3: 192.168.1.60:3825 - ftp.strongmail.net:65023 (e2f) 5> 4<
(complete)
4: 192.168.1.102:1339 - 205.188.8.194:5190 (g2h) 6> 6<
5: 192.168.1.102:1490 - cs127.msg.mud.yahoo.com:5050 (i2j) 5> 5<
6: py-in-f111.google.com:993 - 192.168.1.102:3785 (k2l) 13> 14<
```

上面的输出中,每个连接都有对应的源主机和目的主机.tcptrace 使用-l 和-o 选项可查看某一连接更详细的数据.

以下的结果,就是在 bigstuff 文件中,#16 连接的相关统计数据:

```
# tcptrace -l -o1 bigstuff
```

1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov 4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:00.529361, 276008 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
32 TCP connections traced:
TCP connection 1:
host a: 192.168.1.60:pcanywherestat
host b: 192.168.1.102:2571
complete conn: no (SYNs: 0) (FINs: 0)
first packet: Sun Jul 20 15:58:05.472983 2008
last packet: Sun Jul 20 16:00:04.564716 2008
elapsed time: 0:01:59.091733
total packets: 854
filename: bigstuff
a->b: b->a:
total packets: 404 total packets: 450
ack pkts sent: 404 ack pkts sent: 450
pure acks sent: 13 pure acks sent: 320
sack pkts sent: 0 sack pkts sent: 0
dsack pkts sent: 0 dsack pkts sent: 0
max sack blks/ack: 0 max sack blks/ack: 0
unique bytes sent: 52608 unique bytes sent: 10624
actual data pkts: 391 actual data pkts: 130
actual data bytes: 52608 actual data bytes: 10624
rexmt data pkts: 0 rexmt data pkts: 0
rexmt data bytes: 0 rexmt data bytes: 0
zwnd probe pkts: 0 zwnd probe pkts: 0
zwnd probe bytes: 0 zwnd probe bytes: 0
outoforder pkts: 0 outoforder pkts: 0
pushed data pkts: 391 pushed data pkts: 130
SYN/FIN pkts sent: 0/0 SYN/FIN pkts sent: 0/0
urgent data pkts: 0 pkts urgent data pkts: 0 pkts
urgent data bytes: 0 bytes urgent data bytes: 0 bytes
mss requested: 0 bytes mss requested: 0 bytes
max segm size: 560 bytes max segm size: 176 bytes
min segm size: 48 bytes min segm size: 80 bytes
avg segm size: 134 bytes avg segm size: 81 bytes
max win adv: 19584 bytes max win adv: 65535 bytes
min win adv: 19584 bytes min win adv: 64287 bytes
zero win adv: 0 times zero win adv: 0 times
avg win adv: 19584 bytes avg win adv: 64949 bytes
initial window: 160 bytes initial window: 0 bytes
initial window: 2 pkts initial window: 0 pkts
ttl stream length: NA ttl stream length: NA

missed data: NA missed data: NA
truncated data: 36186 bytes truncated data: 5164 bytes
truncated packets: 391 pkts truncated packets: 130 pkts
data xmit time: 119.092 secs data xmit time: 116.954 secs
idletime max: 441267.1 ms idletime max: 441506.3 ms
throughput: 442 Bps throughput: 89 Bps

8.3.2 案例学习 - 计算转播率

几乎不可能确定说哪个连接会有严重不足的转播问题,只是需要分析,使用 tcptrace 工具可以通过过滤机制和布尔表达式来找出出问题的连接.一个很繁忙的网络中,会有很多的连接,几乎所有的连接都会有转播.找出其中最多的一个,这就是问题的关键.

下面的例子里,tcptrace 将找出那些转播大于 100 segments(译注:分段数)的连接:

tcptrace -f'rexmit_segs>100' bigstuff
Output filter: ((c_rexmit_segs>100)OR(s_rexmit_segs>100))
1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov 4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:00.687788, 212431 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
16: ftp.strongmail.net:65014 - 192.168.1.60:2158 (ae2af) 18695> 9817<

在这个输出中,是#16 这个连接里,超过了 100 转播.现在,使用以下命令查看关于这个连接的其他信息:

tcptrace -l -o16 bigstuff
arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov 4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:01.355964, 107752 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
32 TCP connections traced:
==========
TCP connection 16:
host ae: ftp.strongmail.net:65014
host af: 192.168.1.60:2158
complete conn: no (SYNs: 0) (FINs: 1)
first packet: Sun Jul 20 16:04:33.257606 2008
last packet: Sun Jul 20 16:07:22.317987 2008
elapsed time: 0:02:49.060381
total packets: 28512
filename: bigstuff
ae->af: af->ae:

unique bytes sent: 25534744 unique bytes sent: 0

actual data pkts: 18695 actual data pkts: 0
actual data bytes: 25556632 actual data bytes: 0
rexmt data pkts: 1605 rexmt data pkts: 0
rexmt data bytes: 2188780 rexmt data bytes: 0

计算转播率:
rexmt/actual * 100 = Retransmission rate

1605/18695* 100 = 8.5%

这个慢连接的原因,就是因为它有 8.5% 的转播率.

8.3.3 案例学习 - 计算转播时间

tcptrace 工具有一系列的模块展示不同的数据,按照属性,其中就有 protocol(译注:协议),port(译注:端口),time 等等.Slice module 使得你可观察在一段时间内的 TCP 性能.你可以在一系列的转发过程中,查看其他性能数据,以确定找出瓶颈.

以下例子示范了,tcptrace 是怎样使用 slice 模式的:

tcptrace –xslice bigfile

以上命令会创建一个 slice.dat 文件在现在的工作目录中.这个文件内容,包含是每 15 秒间隔内转播的相关信息:

ls -l slice.dat
-rw-r--r-- 1 root root 3430 Jul 10 22:50 slice.dat
more slice.dat
date segs bytes rexsegs rexbytes new active
--------------- -------- -------- -------- -------- -------- --------
22:19:41.913288 46 5672 0 0 1 1
22:19:56.913288 131 25688 0 0 0 1
22:20:11.913288 0 0 0 0 0 0
22:20:26.913288 5975 4871128 0 0 0 1
22:20:41.913288 31049 25307256 0 0 0 1
22:20:56.913288 23077 19123956 40 59452 0 1
22:21:11.913288 26357 21624373 5 7500 0 1
22:21:26.913288 20975 17248491 3 4500 12 13
22:21:41.913288 24234 19849503 10 15000 3 5
22:21:56.913288 27090 22269230 36 53999 0 2
22:22:11.913288 22295 18315923 9 12856 0 2
22:22:26.913288 8858 7304603 3 4500 0 1

8.4 结论

监控网络性能由以下几个部分组成:

1,检查并确定所有网卡都工作在正确的速率.
2,检查每块网卡的吞吐量,并确认其处于服务时的网络速度.
3,监控网络流量的类型,并确定适当的流量优先级策略.

Linux System and Performance Monitoring(总结篇)

08 月 14th, 2009 Posted in Linux, Monitor

作者:tonnyom
原载: http://www.sanotes.net/html/y2009/393.html

Linux System and Performance Monitoring(总结篇)
Date: 2009.07.21
Author: Darren Hoch
译: Tonnyom[AT]hotmail.com

结束语: 这是该译文的最后一篇,在这篇中,作者提供了一个案例环境,用之前几篇所阐述的理论以及涉及到的工具,对其进行一个整体的系统性能检查.对大家更好理解系统性能监控,进行一次实战演习.
BTW:在中文技术网站上,类似内容的文章,大体是来自该作者 06-07 年所著论文,此译文是建立在作者为 OSCON 2009 重写基础上的.所以部分内容可能会存在重复雷同,特此说明下.

附录 A: 案例学习 - 性能监控之循序渐进

某一天,一个客户打电话来需要技术帮助,并抱怨平常 15 秒就可以打开的网页现在需要 20 分钟才可以打开.

具体系统配置如下:

RedHat Enterprise Linux 3 update 7
Dell 1850 Dual Core Xenon Processors, 2 GB RAM, 75GB 15K Drives
Custom LAMP software stack(译注:Llinux+apache+mysql+php 环境)

性能分析之步骤

1. 首先使用 vmstat 查看大致的系统性能情况:

```
# vmstat 1 10
procs memory swap io system cpu
r b swpd free buff cache si so bi bo in cs us sy id wa
1 0 249844 19144 18532 1221212 0 0 7 3 22 17 25 8 17 18
0 1 249844 17828 18528 1222696 0 0 40448 8 1384 1138 13 7 65 14
0 1 249844 18004 18528 1222756 0 0 13568 4 623 534 3 4 56 37
2 0 249844 17840 18528 1223200 0 0 35200 0 1285 1017 17 7 56 20
1 0 249844 22488 18528 1218608 0 0 38656 0 1294 1034 17 7 58 18
0 1 249844 21228 18544 1219908 0 0 13696 484 609 559 5 3 54 38
0 1 249844 17752 18544 1223376 0 0 36224 4 1469 1035 10 6 67 17
1 1 249844 17856 18544 1208520 0 0 28724 0 950 941 33 12 49 7
1 0 249844 17748 18544 1222468 0 0 40968 8 1266 1164 17 9 59 16
1 0 249844 17912 18544 1222572 0 0 41344 12 1237 1080 13 8 65 13
```

分析:
1,不会是内存不足导致,因为 swapping 始终没变化(si 和 so).尽管空闲内存不多(free),但 swpd 也没有变化.
2,CPU 方面也没有太大问题,尽管有一些运行队列(procs r),但处理器还始终有 50% 多的 idle(CPU id).
3,有太多的上下文切换(cs)以及 disk block 从 RAM 中被读入(bo).
4,CPU 还有平均 20% 的 I/O 等待情况.

结论:
从以上总结出,这是一个 I/O 瓶颈.

2. 然后使用 iostat 检查是谁在发出 IO 请求:

iostat -x 1
Linux 2.4.21-40.ELsmp (mail.example.com) 03/26/2007

avg-cpu: %user %nice %sys %idle
30.00 0.00 9.33 60.67

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 7929.01 30.34 1180.91 14.23 7929.01 357.84 3964.50 178.92 6.93 0.39 0.03 0.06 6.69
/dev/sda1 2.67 5.46 0.40 1.76 24.62 57.77 12.31 28.88 38.11 0.06 2.78 1.77 0.38
/dev/sda2 0.00 0.30 0.07 0.02 0.57 2.57 0.29 1.28 32.86 0.00 3.81 2.64 0.03
/dev/sda3 7929.01 24.58 1180.44 12.45 7929.01 297.50 3964.50 148.75 6.90 0.32 0.03 0.06 6.68

avg-cpu: %user %nice %sys %idle
9.50 0.00 10.68 79.82

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 0.00 1195.24 0.00 0.00 0.00 0.00 0.00 0.00 43.69 3.60 0.99 117.86
/dev/sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda2 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda3 0.00 0.00 1195.24 0.00 0.00 0.00 0.00 0.00 0.00 43.69 3.60 0.99 117.86

avg-cpu: %user %nice %sys %idle
9.23 0.00 10.55 79.22

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00 0.00 1200.37 0.00 0.00 0.00 0.00 0.00 0.00 41.65 2.12 0.99 112.51
/dev/sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda2 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
/dev/sda3 0.00 0.00 1200.37 0.00 0.00 0.00 0.00 0.00 0.00 41.65 2.12 0.99 112.51

分析:
1,看上去只有/dev/sda3 分区很活跃,其他分区都很空闲.
2,差不多有 1200 读 IOPS,磁盘本身是支持 200 IOPS 左右(译注:参考之前的 IOPS 计算公式).
3,有超过 2 秒,实际上没有一个读磁盘(rkb/s).这和在 vmstat 看到有大量 I/O wait 是有关系的.
4,大量的 read IOPS(r/s)和在 vmstat 中大量的上下文是匹配的.这说明很多读操作都是失败的.

结论:
从以上总结出,部分应用程序带来的读请求,已经超出了 I/O 子系统可处理的范围.

3. 使用 top 来查找系统最活跃的应用程序

top -d 1
11:46:11 up 3 days, 19:13, 1 user, load average: 1.72, 1.87, 1.80
176 processes: 174 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: cpu user nice system irq softirq iowait idle
total 12.8% 0.0% 4.6% 0.2% 0.2% 18.7% 63.2%
cpu00 23.3% 0.0% 7.7% 0.0% 0.0% 36.8% 32.0%
cpu01 28.4% 0.0% 10.7% 0.0% 0.0% 38.2% 22.5%
cpu02 0.0% 0.0% 0.0% 0.9% 0.9% 0.0% 98.0%
cpu03 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 100.0%
Mem: 2055244k av, 2032692k used, 22552k free, 0k shrd, 18256k buff

1216212k actv, 513216k in_d, 25520k in_c
Swap: 4192956k av, 249844k used, 3943112k free 1218304k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
14939 mysql 25 0 379M 224M 1117 R 38.2 25.7% 15:17.78 mysqld
4023 root 15 0 2120 972 784 R 2.0 0.3 0:00.06 top
1 root 15 0 2008 688 592 S 0.0 0.2 0:01.30 init
2 root 34 19 0 0 0 S 0.0 0.0 0:22.59 ksoftirqd/0
3 root RT 0 0 0 0 S 0.0 0.0 0:00.00 watchdog/0
4 root 10 -5 0 0 0 S 0.0 0.0 0:00.05 events/0

分析:
1,占用资源最多的好像就是 mysql 进程,其他都处于完全 idle 状态.
2,在 top(wa) 看到的数值,和在 vmstat 看到的 wio 数值是有关联的.

结论:
从以上总结出,似乎就只有 mysql 进程在请求资源,因此可以推论它就是导致问题的关键.

4. 现在已经确定是 mysql 在发出读请求,使用 strace 来检查它在读请求什么.

strace -p 14939

Process 14939 attached - interrupt to quit
read(29, "\3\1\237\1\366\337\1\222%\4\2\0\0\0\0\0012P/d", 20) = 20
read(29, "ata1/strongmail/log/strongmail-d"..., 399) = 399
_llseek(29, 2877621036, [2877621036], SEEK_SET) = 0
read(29, "\1\1\241\366\337\1\223%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 400) = 400
_llseek(29, 2877621456, [2877621456], SEEK_SET) = 0
read(29, "\1\1\235\366\337\1\224%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 396) = 396
_llseek(29, 2877621872, [2877621872], SEEK_SET) = 0
read(29, "\1\1\245\366\337\1\225%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 404) = 404
_llseek(29, 2877622296, [2877622296], SEEK_SET) = 0
read(29, "\3\1\236\2\366\337\1\226%\4\2\0\0\0\0\0012P/d", 20) = 20

分析:
1,大量的读操作都在不断寻道中,说明 mysql 进程产生的是随机 IO.
2,看上去似乎是,某一 sql 查询导致读操作.

结论:
从以上总结出,所有的读 IOPS 都是 mysql 进程在执行某些读查询时产生的.

5. 使用 mysqladmin 命令,来查找是哪个慢查询导致的.

./mysqladmin -pstrongmail processlist

+----+------+-----------+------------+---------+------+----------+-----------------------------------
| Id | User | Host | db | Command | Time | State | Info
+----+------+-----------+------------+---------+------+----------+-----------------------------------
| 1 | root | localhost | strongmail | Sleep | 10 | |
| 2 | root | localhost | strongmail | Sleep | 8 | |

| 3 | root | localhost | root | Query | 94 | Updating | update `failures` set
`update_datasource`='Y' where database_id='32' and update_datasource='N' and |
| 14 | root | localhost | | Query | 0 | | show processlist

分析:
1,MySQL 数据库里,似乎在不断的运行 table update 查询.
2,基于这个 update 查询,数据库是对所有的 table 进行索引.

结论:
从以上总结出,MySQL 里这些 update 查询问题,都是在尝试对所有 table 进行索引.这些产生的读请求正是导致系统性能下降的原因.

后续

把以上这些性能信息移交给了相关开发人员,用于分析他们的 PHP 代码.一个开发人员对代码进行了临时性优化.某个查询如果出错了,也最多到 100K 记录.数据库本身考虑最多存在 4 百万记录.最后,这个查询不会再给数据库带来负担了.

References

• Ezlot, Phillip – Optimizing Linux Performance, Prentice Hall, Princeton NJ 2005 ISBN – 0131486829
• Johnson, Sandra K., Huizenga, Gerrit – Performance Tuning for Linux Servers, IBM Press, Upper Saddle River NJ 2005 ISBN 013144753X
• Bovet, Daniel Cesati, Marco – Understanding the Linux Kernel, O'Reilly Media, Sebastoppl CA 2006, ISBN 0596005652
• Blum, Richard – Network Performance Open Source Toolkit, Wiley, Indianapolis IN 2003, ISBN 0-471-43301-2
• Understanding Virtual Memory in RedHat 4, Neil Horman, 12/05 http://people.redhat.com/nhorman/papers/rhel4_vm.pdf
• IBM, Inside the Linux Scheduler, http://www.ibm.com/developerworks/linux/library/l-scheduler/
• Aas, Josh, Understanding the Linux 2.6.8.1 CPU Scheduler, http://josh.trancesoftware.com/linux/linux_cpu_scheduler.pdf
• Wieers, Dag, Dstat: Versatile Resource Statistics Tool, http://dag.wieers.com/home-made/dstat/