



咕泡学院 **JAVA VIP** 高级课程教案

性能优化系列专题

主讲	James QQ 2904270631
版本	V1.0
适用对象	咕泡学院 JAVA 高级 VIP 学员

本文档版权归咕泡学院所有，禁止未经允许私自传播。

目录

课程安排2

1. 性能优化是什么?2

 1.1 性能优化就是发挥机器本来的性能.....3

2. 性能的几个维度.....3

 1.1.1 CPU.....3

 1.1.2 IO.....3

 1.1.3 Memory4

 1.1.4 Network.....4

 1.1.5 监控软件6

3. 术语6

课程安排

1. 性能优化是什么？

1.1 性能优化就是发挥机器本来的性能

2. 性能的几个维度

1.1.1 CPU

命令 **vmstat**

procs		-----memory-----				---swap--		-----io-----		-system--		-----cpu-----				
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
1	0	0	1296776	237692	2303120	0	0	0	244	467	798	1	1	99	0	0
0	0	0	1294612	237692	2303120	0	0	0	72	693	1101	1	1	99	0	0
0	0	0	1294156	237692	2303124	0	0	0	4	507	823	1	1	99	0	0
1	0	0	1294172	237692	2303128	0	0	0	0	439	732	0	0	100	0	0
0	0	0	1301072	237692	2303128	0	0	0	0	832	1450	2	1	98	0	0
0	0	0	1301180	237692	2303100	0	0	0	0	443	730	1	0	100	0	0

<http://www.man7.org/linux/man-pages/man8/vmstat.8.html>

首先检查 **cpu**，cpu 使用率要提升而不是降低

CPU 空闲并不一定是没事做，也有可能是锁或者外部资源瓶颈。

命令 **Top**

top - 17:43:36 up 4 days, 6:13, 1 user, load average: 2.32, 1.69, 0.89															
Tasks: 180 total, 3 running, 177 sleeping, 0 stopped, 0 zombie															
%Cpu(s): 75.3 us, 1.7 sy, 0.0 ni, 23.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st															
KiB Mem : 8010688 total, 1406260 free, 4061864 used, 2542564 buff/cache															
KiB Swap: 0 total, 0 free, 0 used. 3545444 avail Mem															
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND				
19718	git	20	0	943532	572584	6624	R	84.7	7.1	3:21.68	bundle				
2617	git	20	0	857488	480220	5788	S	43.0	6.0	0:06.49	bundle				
2539	git	20	0	894380	512604	5976	S	18.0	6.4	0:14.14	bundle				
19815	gitlab-+	20	0	1224804	77208	16148	S	3.7	1.0	0:08.24	postgres				
2631	gitlab-+	20	0	1166744	16580	11432	R	2.3	0.2	0:00.14	postgres				
2554	gitlab-+	20	0	1169336	20620	12536	S	0.7	0.3	0:00.45	postgres				
26012	git	20	0	316936	28872	5712	S	0.7	0.4	2:22.10	gitlab-workhors				
26116	git	20	0	981716	653444	14120	S	0.7	8.2	34:10.68	bundle				

<http://man7.org/linux/man-pages/man1/top.1.html>

1.1.2 IO

命令 **iostat**

```
[root@izwz9a13nu3oo6utvy6duoz ~]# iostat -xd 1
Linux 3.10.0-514.26.2.el7.x86_64 (izwz9a13nu3oo6utvy6duoz) 01/02/2018 _x86_64_ (2 CPU)
```

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
vda	0.00	4.58	0.18	4.19	4.19	150.93	70.99	0.08	19.27	6.19	19.85	0.72	0.31
Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
vda	0.00	20.00	1.00	3.00	8.00	104.00	56.00	0.00	0.75	0.00	1.00	0.75	0.30

<http://www.man7.org/linux/man-pages/man1/iostat.1.html>

1.1.3 Memory

命令 **free**

```
[root@izwz9a13nu3oo6utvy6duoz ~]# free -g
```

	total	used	free	shared	buff/cache	available
Mem:	7	4	1	0	2	3
Swap:	0	0	0			

<http://www.man7.org/linux/man-pages/man1/free.1.html>

两者都是RAM中的数据。简单来说，buffer是即将要被写入磁盘的，cache是被从磁盘读出出来的。这二者是为了提高IO性能的，并由OS管理，并非应用自己分配的内存，而是OS自己根据需要对空闲内存进行的额外利用。因为这部分只是缓存，降低IO，提升性能，只要应用程序有需要，OS可以直接将buffer写入磁盘，将cache删掉来得到空闲内存给应用程序使用。

buffer是用于存储速度不同步的设备或优先级不同的设备之间传输数据的区域。缓冲（buffers）是根据磁盘的读写设计的，把分散的写操作集中进行，减少磁盘碎片和硬盘的反复寻道，从而提高系统性能。

cache经常被用在磁盘的I/O请求上，如果有多个进程都要访问某个文件，于是该文件便被做成cache以便下次被访问，这样可提供系统性能。缓存（cached）是把读取过的数据保存起来，重新读取时若命中（找到需要的数据）就不要去读硬盘了，若没有命中就读硬盘。其中的数据会根据读取频率进行组织，把最频繁读取的内容放在最容易找到的位置，把不再读的内容不断往后排，直至从中删除。

因此：

-/+ buffers/cache的含义即：使用内存是实际当前使用内存减去buffers/cache之和；空闲内存是实际空闲内存加上buffers/cache之和。所以是-/+

查看空闲内存，确定应用是否有内存泄漏时，只能以Free的第三行为依据，第二行其实作用不大，只是可以看到OS当前的buffer和cache大小。

1.1.4 Network

命令 **netstat** (需要安装)

```
wget http://sourceforge.net/projects/nicstat/files/nicstat-1.92.tar.gz
```

```
tar -zxvf nicstat-1.92.tar.gz
```

```
sudo vim Makefile
```

```
CFLAGS = $(COPT) -m32#将此行修改为如下:
```

```
CFLAGS = $(COPT)
```

```
sudo make -f Makefile install
```

Time列:表示当前采样的响应时间.

lo and eth0 : 网卡名称.

rKB/s : 每秒接收到千字节数.

wKB/s : 每秒写的千字节数.

rPk/s : 每秒接收到的数据包数目.

wPk/s : 每秒写的数据包数目.

rAvs : 接收到的数据包平均大小.

wAvs : 传输的数据包平均大小.

%Util : 网卡利用率(百分比).

Sat : 网卡每秒的错误数.网卡是否接近饱满的一个指标.尝试去诊断网络问题的时候,推荐使用-x选项去查看更多的统计信息.

◦ 查看tcp相关信息(-t):

```
[root@centos192 nicstat-1.92]# ./nicstat.sh -t
05:15:05      InKB   OutKB   InSeg   OutSeg  Reset   AttF %ReTX InConn OutCon Drops
TCP          0.00    0.00    4.01    3.50   0.00   0.01 0.000    0.05   0.09   0.0
```

InKB : 表示每秒接收到的千字节.

OutKB : 表示每秒传输的千字节.

InSeg : 表示每秒接收到的TCP数据段(TCP Segments).

OutSeg : 表示每秒传输的TCP数据段(TCP Segments).

Reset : 表示TCP连接从ESTABLISHED或CLOSE-WAIT状态直接转变为CLOSED状态的次数.

AttF : 表示TCP连接从SYN-SENT或SYN-RCVD状态直接转变为CLOSED状态的次数,再加上TCP连接从SYN-RCVD状态直接转变为LISTEN状态的次数

%ReTX : 表示TCP数据段(TCP Segments)重传的百分比.即传输的TCP数据段包含有一个或多个之前传输的八位字节.

InConn : 表示TCP连接从LISTEN状态直接转变为SYN-RCVD状态的次数.

OutCon : 表示TCP连接从CLOSED状态直接转变为SYN-SENT状态的次数.

Drops : 表示从完成连接(completed connection)的队列和未完成连接(incomplete connection)的队列中丢弃的连接次数.

- 查看udp相关信息(-u):

```
[root@centos192 nicstat-1.92]# ./nicstat.sh -u
06:39:42                InDG    OutDG      InErr  OutErr
UDP                0.01    0.01      0.00   0.00
```

InDG : 每秒接收到的UDP数据报(UDP Datagrams)

OutDG : 每秒传输的UDP数据报(UDP Datagrams)

InErr : 接收到的因包含错误而不能被处理的数据包

OutErr : 因错误而不能成功传输的数据包.

- 默认以KB为单位,现在以M单位查看:

```
[root@centos192 nicstat-1.92]# ./nicstat.sh -M
      Time      Int  rMbps  wMbps  rPk/s  wPk/s   rAvs   wAvs %Util   Sat
05:16:55      lo   0.01   0.01   2.55   2.55  341.2  341.2  0.00  0.00
05:16:55    eth0   0.01   0.00   1.61   1.08  667.3  163.7  0.00  0.00
```

1.1.5 监控软件

<https://www.zabbix.com/documentation/2.0/manual/appendix/api/api>

zabbix nagios prometheus

3. 术语

吞吐量：对单位时间内完成的工作量的度量

平均响应时间：提交请求和返回该请求的响应之间使用的时间

平均响应时间越短，系统吞吐量越大；平均响应时间越长，系统吞吐量越小；但是，系统吞吐量越大，未必平均响应时间越短；因为在某些情况（例如，不增加任何硬件配置）吞吐量的增大，有时会把平均响应时间作为牺牲，来换取一段时间处理更多的请求。

tps: Transactions per Second

qps: Queries per Second

4. 补充

CPU 负载高怎么定位：

A. top 找到 CPU 高的进程 （原理：方法是由线程执行的，线程是在进程下的，找到进程下 cpu 最高的线程就能定位到方法）

```
top - 22:36:29 up 2 days, 9:41, 6 users, load average: 1.00, 0.79, 0.42
Tasks: 414 total, 1 running, 413 sleeping, 0 stopped, 0 zombie
%Cpu0 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem: 3869040 total, 867700 used, 3001340 free, 0 buffers
KiB Swap: 4079612 total, 223408 used, 3856204 free, 156860 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	MEM	TIME+	COMMAND
40409	adminis+	20	0	4566620	413304	13496	S	100.0	10.7	8:18.68	java
40289	adminis+	20	0	135680	544	408	S	0.3	0.0	0:00.21	sshd
41274	adminis+	20	0	124008	1996	1172	R	0.3	0.1	0:02.48	top
1	root	20	0	53692	3588	1320	S	0.0	0.1	0:10.52	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.19	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:03.39	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.58	migration/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuob/0

B. Shift + H 切换到线程模型 找到线程执行 cpu 高的线程号

```
KiB Swap: 4079612 total, 223612 used, 3856000 free, 149708 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
40437	adminis+	20	0	4566620	413304	13496	R	99.9	10.7	1:47.22	java
41274	adminis+	20	0	124008	1996	1172	R	0.3	0.1	0:00.81	top
1	root	20	0	53692	3588	1320	S	0.0	0.1	0:10.52	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.19	kthreadd

C. Jstack pid > p.txt 用 jstack 导出线程的 dump (记住这个问题有时候没有那么明显一直 cpu100%，可能是间歇性的 cpu 高所以这个能抓住这个线程还是要看运气)

D. 把线程号转 16 进制 printf "%x \n" 40437

```
"0"9df5[administrator@localhost ~]$ printf "%x \n" 40437
9df5
```

F. 到刚刚导出的 p.txt 里面检索定位到

```
at java.lang.Thread.run(Thread.java:745)
"http-nio-8080-exec-6" #22 daemon prio=5 os_prio=0 tid=0x00007f65d909b800 nid=0x9dfe runnable [0x00007f65bbebf000]
java.lang.Thread.State: RUNNABLE
  at com.gupao.controllers.HelloWorldController.cpu(HelloWorldController.java:27)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:205)
  at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:133)
  at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:97)
                                                                    111,81      35%
```