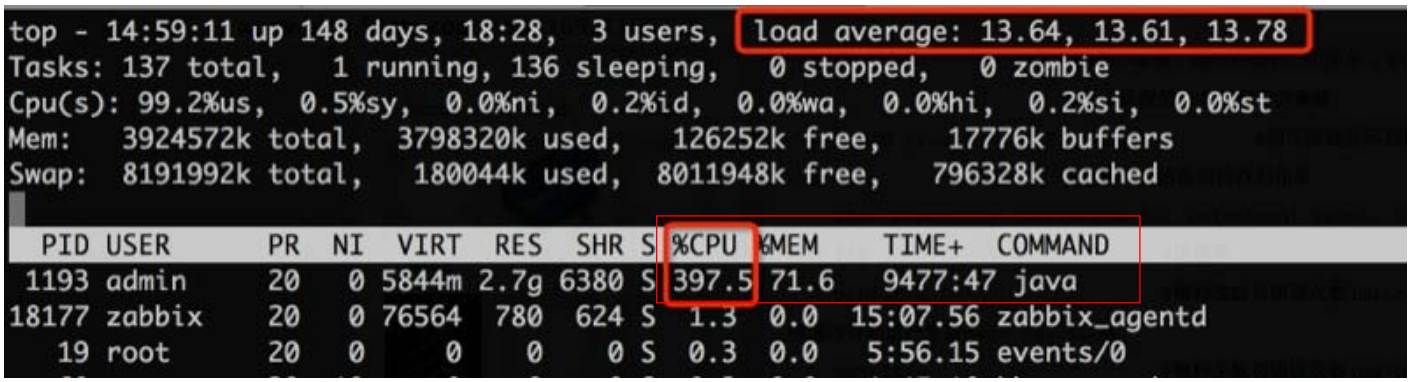
CPU Load过高问题分析和解决方案

问题现象：

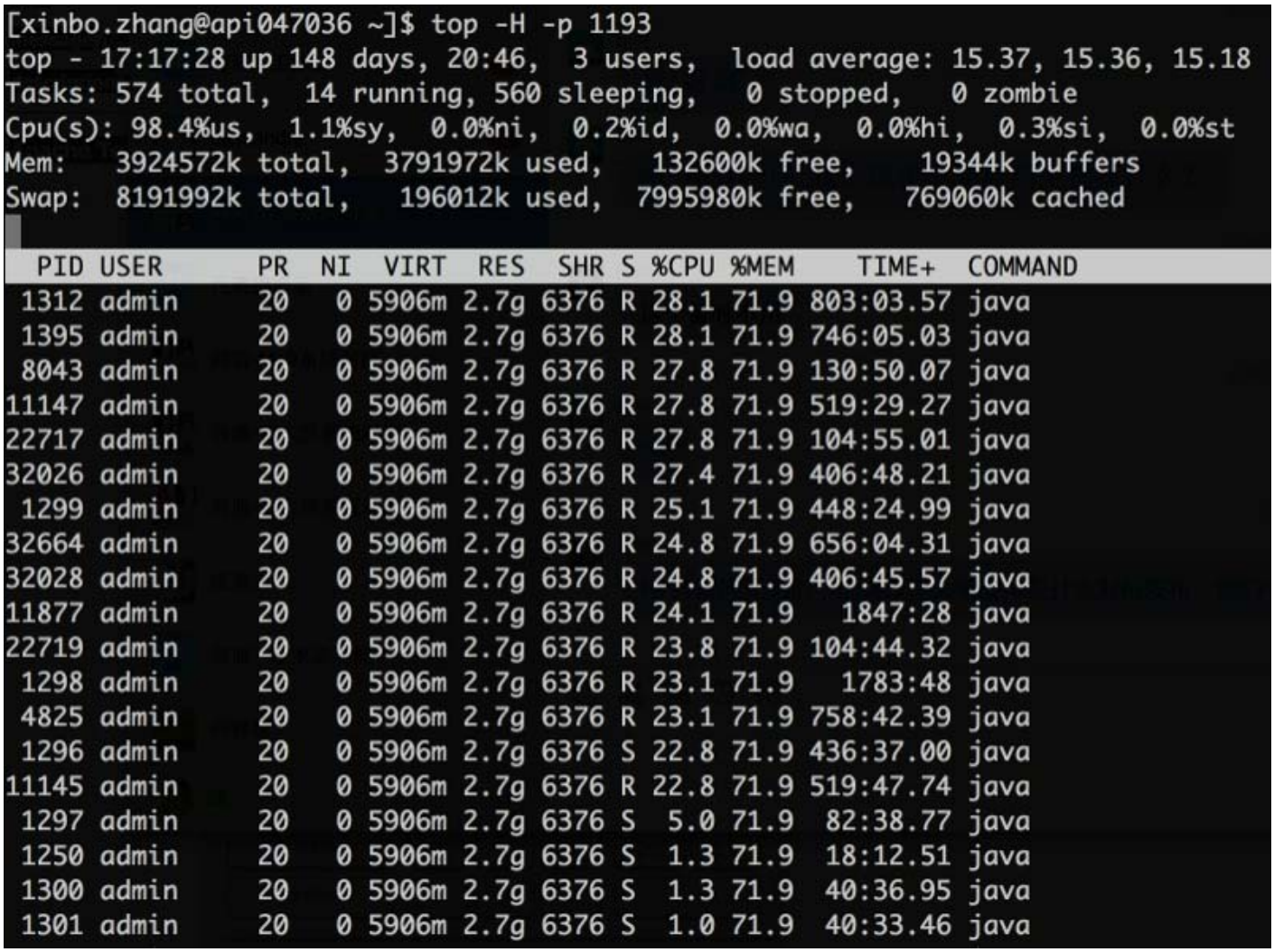
上个月有一天，突然发现线上服务器Load特别高，4核心CPU几乎跑满了，之前Load一般才不到1的样子，从服务器用 `Top` 命令查看如下：



```
top - 14:59:11 up 148 days, 18:28,  3 users,  load average: 13.64, 13.61, 13.78
Tasks: 137 total,   1 running, 136 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.2%us,  0.5%sy,  0.0%ni,  0.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   3924572k total,  3798320k used,   126252k free,    17776k buffers
Swap:  8191992k total,   180044k used,  8011948k free,   796328k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1193 admin     20   0 5844m 2.7g 6380 S 397.5 71.6   9477:47 java
18177 zabbix    20   0 76564  780  624 S  1.3  0.0  15:07.56 zabbix_agentd
   19 root      20   0     0    0    0 S  0.3  0.0   5:56.15 events/0
```

确定是Java应用程序把CPU占满了，使用 `top -H -p 1193` 查看占用CPU比较多的线程有哪些：



```
[xinbo.zhang@api047036 ~]$ top -H -p 1193
top - 17:17:28 up 148 days, 20:46,  3 users,  load average: 15.37, 15.36, 15.18
Tasks: 574 total,  14 running, 560 sleeping,   0 stopped,   0 zombie
Cpu(s): 98.4%us,  1.1%sy,  0.0%ni,  0.2%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   3924572k total,  3791972k used,   132600k free,    19344k buffers
Swap:  8191992k total,   196012k used,  7995980k free,   769060k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1312 admin     20   0 5906m 2.7g 6376 R 28.1 71.9 803:03.57 java
 1395 admin     20   0 5906m 2.7g 6376 R 28.1 71.9 746:05.03 java
 8043 admin     20   0 5906m 2.7g 6376 R 27.8 71.9 130:50.07 java
11147 admin     20   0 5906m 2.7g 6376 R 27.8 71.9 519:29.27 java
22717 admin     20   0 5906m 2.7g 6376 R 27.8 71.9 104:55.01 java
32026 admin     20   0 5906m 2.7g 6376 R 27.4 71.9 406:48.21 java
 1299 admin     20   0 5906m 2.7g 6376 R 25.1 71.9 448:24.99 java
32664 admin     20   0 5906m 2.7g 6376 R 24.8 71.9 656:04.31 java
32028 admin     20   0 5906m 2.7g 6376 R 24.8 71.9 406:45.57 java
11877 admin     20   0 5906m 2.7g 6376 R 24.1 71.9   1847:28 java
22719 admin     20   0 5906m 2.7g 6376 R 23.8 71.9 104:44.32 java
 1298 admin     20   0 5906m 2.7g 6376 R 23.1 71.9   1783:48 java
 4825 admin     20   0 5906m 2.7g 6376 R 23.1 71.9 758:42.39 java
 1296 admin     20   0 5906m 2.7g 6376 S 22.8 71.9 436:37.00 java
11145 admin     20   0 5906m 2.7g 6376 R 22.8 71.9 519:47.74 java
 1297 admin     20   0 5906m 2.7g 6376 S  5.0 71.9  82:38.77 java
 1250 admin     20   0 5906m 2.7g 6376 S  1.3 71.9  18:12.51 java
 1300 admin     20   0 5906m 2.7g 6376 S  1.3 71.9  40:36.95 java
 1301 admin     20   0 5906m 2.7g 6376 S  1.0 71.9  40:33.46 java
```

然后，可以使用命令 `printf %x 1298` 将线程号转成十六进制，在线程栈日志中查找对应的线程细则，线程栈日志可以使用 `jstack -l 1193 > jstack.log` 生成。比如：1298转成十六进制是0x512，搜索后可以看到是通知代理IP服务的异步线程，如下所示：

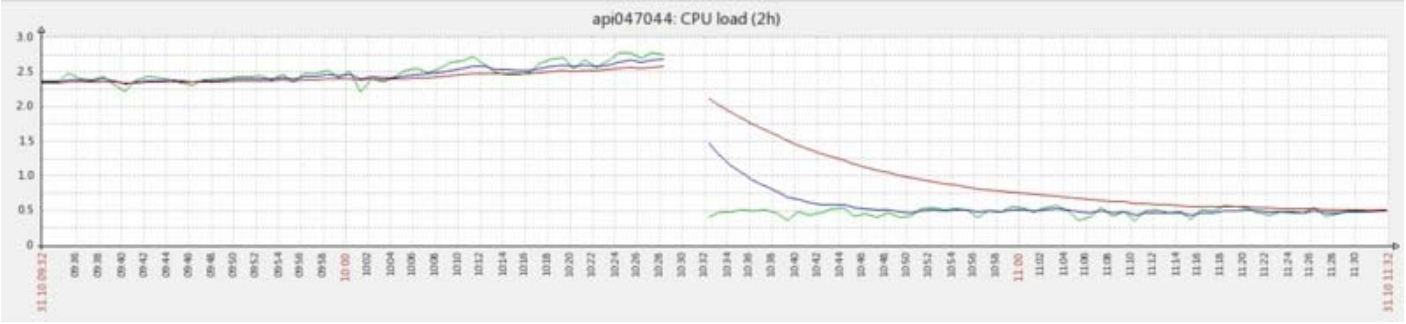
```
"IpNoticeWorker" daemon prio=10 tid=0x00007f2a314a2800 nid=0x512 runnable [0x00007f2a1b993000]
   java.lang.Thread.State: RUNNABLE
        at cn.fraudmetrix.forseti.api.worker.IpNoticeWorker.run(IpNoticeWorker.java:83)
        at java.lang.Thread.run(Thread.java:744)
```

问题根源：

查看这一段代码的源代码可以发现，异步线程在从本地队列中取数据时是一个死循环，但循环中间没有使用 `Thread.sleep()` ，线程会一直不停地运行，导致占用CPU过多，如果没有数据暂停个几毫秒问题就解决了。

其它几个占用CPU比较高的还有Json序列化、从HashMap中放置和读取数据时的线程过高，根本原因都是HashMap在多线程时存在并发问题，将之改成并发的ConcurrentHashMap问题就可以解决了。

最后的CPU监控结果如下，可以看到有明显的下降：



上面查找占用CPU比较高的方式比较麻烦，如果能用命令直接把占用比较高的线程打印出来就完美了，幸运的是发现了阿里的前同事写的脚本，可以直接执行打印占用高的线程，非常的方便好用，点击这里查看。

yikebocai / 2014-11-18