

Linux 系统 CPU 100% 异常排查实践与总结

1、问题背景

昨天下午突然收到运维邮件报警，显示数据平台服务器cpu利用率达到了98.94%，而且最近一段时间一直持续在70%以上，看起来像是硬件资源到瓶颈需要扩容了，但仔细思考就会发现咱们的业务系统并不是一个高并发或者CPU密集型的应用，这个利用率有点太夸张，硬件瓶颈应该不会这么快就到了，一定是哪里的业务代码逻辑有问题。

2、排查思路

2.1 定位高负载进程 pid

首先登录到服务器使用top命令确认服务器的具体情况，根据具体情况再进行分析判断。

```
load average: 17.94, 20.40, 21.05
```

通过观察load average，以及负载评判标准（8核），可以确认服务器存在负载较高的情况；

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
682	work	20	0	25.6g	20g	19m	S	785.9	66.2	5500:18	java
29705	root	20	0	3347m	219m	14m	S	1.3	0.7	300:30.35	java
4989	root	20	0	799m	12m	4376	S	0.7	0.0	226:18.90	falcon-agent
35	root	20	0	0	0	0	S	0.3	0.0	56:54.82	events/0
4510	work	20	0	15036	1284	932	R	0.3	0.0	0:00.26	top
4606	work	20	0	15132	1300	928	S	0.3	0.0	0:00.21	top
25772	mysql	20	0	2087m	193r	5984	S	0.3	0.6	1004:58	mysqld
1	root	20	0	19356	10r	736	S	0.0	0.0	0:33.48	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.15	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	16:04.88	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	19:01.27	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/0

观察各个进程资源使用情况，可以看出进程id为682的进程，有着较高的CPU占比

2.2 定位具体的异常业务

这里咱们可以使用 `pwdx` 命令根据 `pid` 找到业务进程路径，进而定位到负责人和项目：

```
work@online_zeye_master 10.48. 17:32:31
pwdx 682
682: /opt/web/zhuanzhuan_analysis
work@online_zeye_master 10.48. 17:32:36
```

可得出结论：该进程对应的就是数据平台的web服务。

2.3 定位异常线程及具体代码行

传统的方案一般是4步：

- 1、**top oder by with P**: 1040 // 首先按进程负载排序找到 `maxLoad(pid)`
- 2、**top -Hp 进程PID**: 1073 // 找到相关负载 线程PID
- 3、**printf "0x%x " 线程PID**: 0x431 // 将线程PID转换为 16进制，为后面查找 `jstack` 日志做准备
- 4、`jstack 进程PID | vim +/-十六进制线程PID -` // 例如: `jstack 1040|vim +/-0x431 -`

但是对于线上问题定位来说，分秒必争，上面的 4 步还是太繁琐耗时了，之前介绍过淘宝的 `oldratlee` 同学就将上面的流程封装为了一个工具：`show-busy-java-threads.sh`，可以很方便的定位线上的这类问题：

```

[1] Busy(83.2%) thread(26817/0x68c1) stack of java process(26473) under user(work):
"Timer-8" daemon prio=10 tid=0x00007f7d91126800 nid=0x68c1 runnable [0x00007f7db46a9000]
  java.lang.Thread.State: RUNNABLE
    at java.text.SimpleDateFormat.initializeDefaultCentury(SimpleDateFormat.java:894)
    at java.text.SimpleDateFormat.initialize(SimpleDateFormat.java:672)
    at java.text.SimpleDateFormat.<init>(SimpleDateFormat.java:585)
    at java.text.SimpleDateFormat.<init>(SimpleDateFormat.java:560)
    at com.bj58.zhuanzhuan.analysis.util.TimestampUtil.TimestampToDate(TimestampUtil.java:31)
    at com.bj58.zhuanzhuan.analysis.util.DateUtil.getTimesBySplitToday(DateUtil.java:745)
    at com.bj58.zhuanzhuan.analysis.service.impl.MonitorNewServiceImpl.queryNewMonitorDatasFromHbaseByDateOptimize(MonitorNewServiceImpl.java:817)
    at com.bj58.zhuanzhuan.analysis.service.impl.MonitorNewServiceImpl.queryNewMonitorDatasFromHbaseOptimize(MonitorNewServiceImpl.java:648)
    at com.bj58.zhuanzhuan.analysis.thread.ScreenMonitorRunnable.screenMonitor(ScreenMonitorRunnable.java:128)
    at com.bj58.zhuanzhuan.analysis.util.TimeTaskUtils$.run(TimeTaskUtils.java:115)
    at java.util.TimerThread.mainLoop(Timer.java:555)
    at java.util.TimerThread.run(Timer.java:505)

[2] Busy(4.0%) thread(26483/0x6773) stack of java process(26473) under user(work):
"Gang worker#7 (Parallel GC Threads)" prio=10 tid=0x00007f7e18027800 nid=0x6773 runnable

```

可得出结论：是系统中一个时间工具类方法的执行cpu占比较高，定位到具体方法后，查看代码逻辑是否存在性能问题。

※ 如果线上问题比较紧急，可以省略 2.1、2.2 直接执行 2.3，这里从多角度剖析只是为了给大家呈现一个完整的分析思路。

3、根因分析

经过前面的分析与排查，最终定位到一个时间工具类的问题，造成了服务器负载以及cpu使用率的过高。

- **异常方法逻辑：**是把时间戳转成对应的具体的日期时间格式；
- **上层调用：**计算当天凌晨至当前时间所有秒数，转化成对应的格式放入到set中返回结果；
- **逻辑层：**对应的是数据平台实时报表的查询逻辑，实时报表会按照固定的时间间隔来，并且在一次查询中有多次（n次）方法调用。

那么可以得到结论，如果现在时间是当天上午10点，一次查询的计算次数就是 $10 \times 60 \times 60 \times n$ 次 = 36,000 * n 次计算，而且随着时间增长，越接近午夜单次查询次数会线性增加。由于实时查询、实时报警等模块大量的查询请求都需要多次调用该方法，导致了大量CPU资源的占用与浪费。

4、解决方案

定位到问题之后，首先考虑是要减少计算次数，优化异常方法。排查后发现，在逻辑层使用时，并没有使用该方法返回的set集合中的内容，而是简单的用set的size数值。确认逻辑后，通过新方法简化计算（当前秒数-当天凌晨的秒数），替换调用的方法，解决计算过多的问题。上线后观察服务器负载和cpu使用率，对比异常时间段下降了30倍，恢复至正常状态，至此该问题得已解决。

```
top - 22:05:16 up 623 days, 4:12, 4 users, load average: 0.64, 0.84, 0.91
Tasks: 188 total, 1 running, 187 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.2%us, 0.5%sy, 0.0%ni, 97.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 32749608k total, 14721208k used, 18028400k free, 286612k buffers
Swap: 32767996k total, 0k used, 32767996k free, 8796912k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
11786	work	20	0	25.7g	3.7g	13m	S	22.5	12.0	141:05.94	java
29705	root	20	0	3477m	222m	12m	S	2.0	0.7	635:08.20	java
1081	work	20	0	15036	1284	928	R	0.3	0.0	0:00.06	top
4989	root	20	0	799m	12m	4380	S	0.3	0.0	276:37.45	falcon-agent
13579	zabbix	20	0	18016	1148	960	S	0.3	0.0	402:49.97	zabbix_agentd
1	root	20	0	19356	1052	736	S	0.0	0.0	0:38.83	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.15	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	16:26.41	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	19:17.25	ksoftirqd/0

5、总结

- 在编码的过程中，除了要实现业务的逻辑，也要注重代码性能的优化。一个业务需求，能实现，和能实现的更高效、更优雅其实是两种截然不同的工程师能力和境界的体现，而后者也是工程师的核心竞争力。
- 在代码编写完成之后，多做 review，多思考是不是可以用更好的方式来实现。
- 线上问题不放过任何一个小细节！细节是魔鬼，技术的同学需要有刨根问底的求知欲和追求卓越的精神，只有这样，才能不断的成长和提升。

Refer:

[1] 线上服务 CPU 100%? 一键定位 so easy!

[2] linux 系统监控、诊断工具之 top 详解

<https://my.oschina.net/leejun2005/blog/157910>

top命令按内存和cpu排序

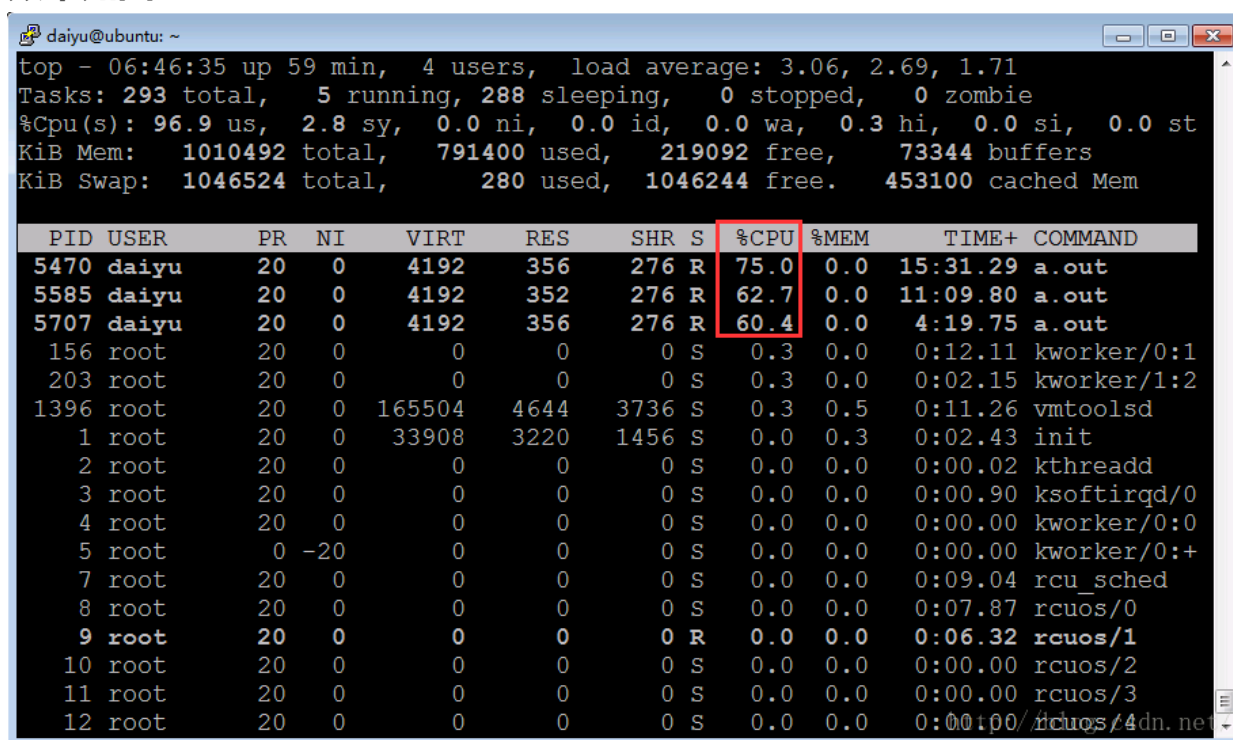
一、按进程的CPU使用率排序

运行top命令后，键入大写P。

有两种途径：

- a) 打开大写键盘的情况下，直接按P键
- b) 未打开大写键盘的情况下，Shift+P键

效果如图：



```
daiyu@ubuntu: ~  
top - 06:46:35 up 59 min, 4 users, load average: 3.06, 2.69, 1.71  
Tasks: 293 total, 5 running, 288 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 96.9 us, 2.8 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st  
KiB Mem: 1010492 total, 791400 used, 219092 free, 73344 buffers  
KiB Swap: 1046524 total, 280 used, 1046244 free. 453100 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5470	daiyu	20	0	4192	356	276	R	75.0	0.0	15:31.29	a.out
5585	daiyu	20	0	4192	352	276	R	62.7	0.0	11:09.80	a.out
5707	daiyu	20	0	4192	356	276	R	60.4	0.0	4:19.75	a.out
156	root	20	0	0	0	0	S	0.3	0.0	0:12.11	kworker/0:1
203	root	20	0	0	0	0	S	0.3	0.0	0:02.15	kworker/1:2
1396	root	20	0	165504	4644	3736	S	0.3	0.5	0:11.26	vmtoolsd
1	root	20	0	33908	3220	1456	S	0.0	0.3	0:02.43	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.90	ksoftirqd/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
7	root	20	0	0	0	0	S	0.0	0.0	0:09.04	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:07.87	rcuos/0
9	root	20	0	0	0	0	R	0.0	0.0	0:06.32	rcuos/1
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/2
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/3
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/4

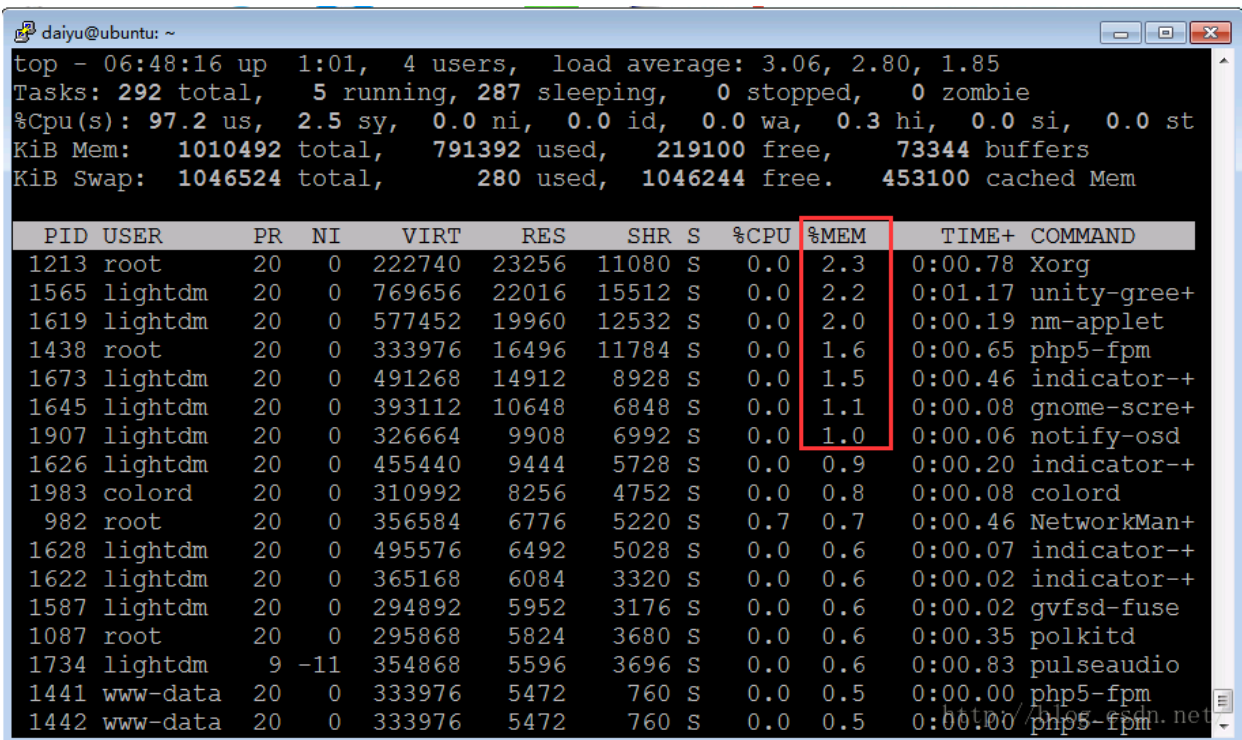
二、按进程的内存使用率排序

运行top命令后，键入大写M。

有两种途径：

- a) 打开大写键盘的情况下，直接按M键
- b) 未打开大写键盘的情况下，Shift+M键

效果如图：



```
top - 06:48:16 up 1:01, 4 users, load average: 3.06, 2.80, 1.85
Tasks: 292 total, 5 running, 287 sleeping, 0 stopped, 0 zombie
%Cpu(s): 97.2 us, 2.5 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st
KiB Mem: 1010492 total, 791392 used, 219100 free, 73344 buffers
KiB Swap: 1046524 total, 280 used, 1046244 free. 453100 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1213	root	20	0	222740	23256	11080	S	0.0	2.3	0:00.78	Xorg
1565	lightdm	20	0	769656	22016	15512	S	0.0	2.2	0:01.17	unity-gree+
1619	lightdm	20	0	577452	19960	12532	S	0.0	2.0	0:00.19	nm-applet
1438	root	20	0	333976	16496	11784	S	0.0	1.6	0:00.65	php5-fpm
1673	lightdm	20	0	491268	14912	8928	S	0.0	1.5	0:00.46	indicator-+
1645	lightdm	20	0	393112	10648	6848	S	0.0	1.1	0:00.08	gnome-scre+
1907	lightdm	20	0	326664	9908	6992	S	0.0	1.0	0:00.06	notify-osd
1626	lightdm	20	0	455440	9444	5728	S	0.0	0.9	0:00.20	indicator-+
1983	colord	20	0	310992	8256	4752	S	0.0	0.8	0:00.08	colord
982	root	20	0	356584	6776	5220	S	0.7	0.7	0:00.46	NetworkMan+
1628	lightdm	20	0	495576	6492	5028	S	0.0	0.6	0:00.07	indicator-+
1622	lightdm	20	0	365168	6084	3320	S	0.0	0.6	0:00.02	indicator-+
1587	lightdm	20	0	294892	5952	3176	S	0.0	0.6	0:00.02	gvfsd-fuse
1087	root	20	0	295868	5824	3680	S	0.0	0.6	0:00.35	polkitd
1734	lightdm	9	-11	354868	5596	3696	S	0.0	0.6	0:00.83	pulseaudio
1441	www-data	20	0	333976	5472	760	S	0.0	0.5	0:00.00	php5-fpm
1442	www-data	20	0	333976	5472	760	S	0.0	0.5	0:00.00	php5-fpm