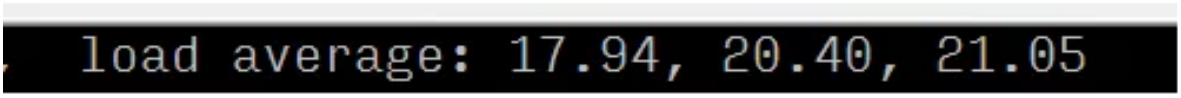
Linux CPU负载过高问题排查

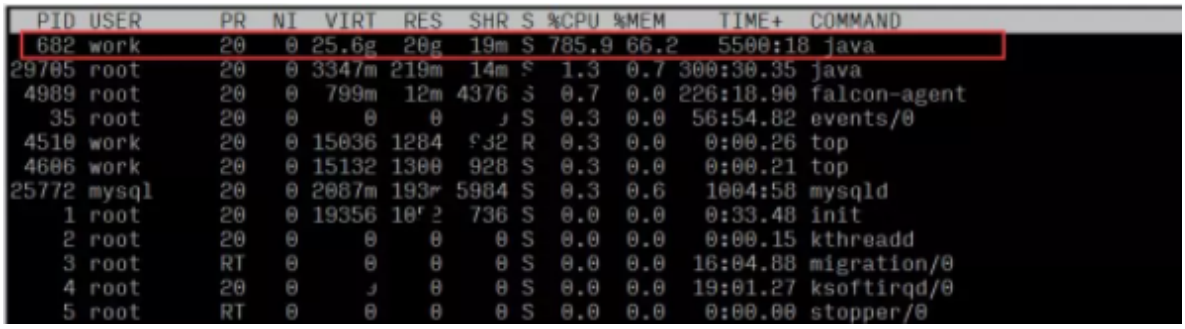
1、排查思路

1.1 定位高负载进程

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。



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；

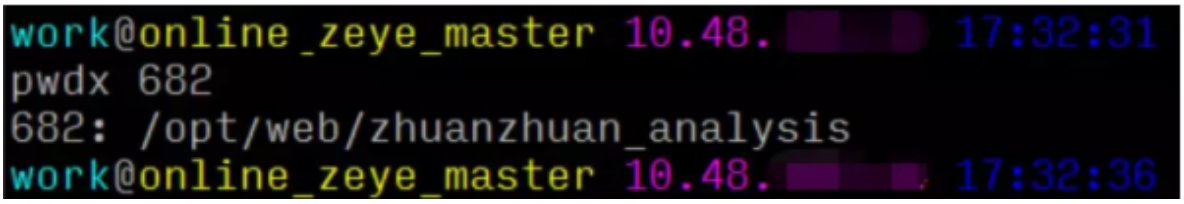


CPU负载过高异常排查实践与总结CPU负载过高异常排查实践与总结

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

1.2 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：



CPU负载过高异常排查实践与总结CPU负载过高异常排查实践与总结

可得出结论：该进程对应的就是数据平台的web服务。

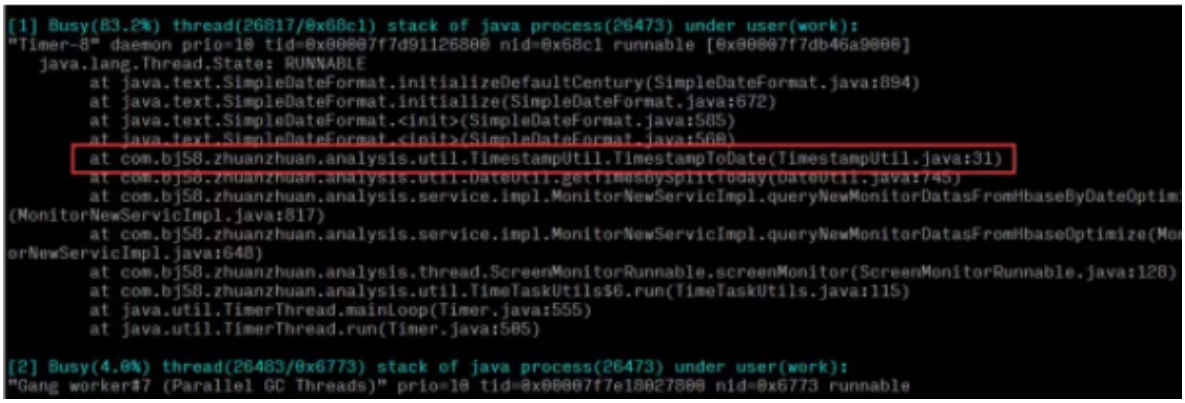
1.3 定位异常线程及具体代码行

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\n”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：



CPU负载过高异常排查实践与总结CPU负载过高异常排查实践与总结

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。

2、根因分析

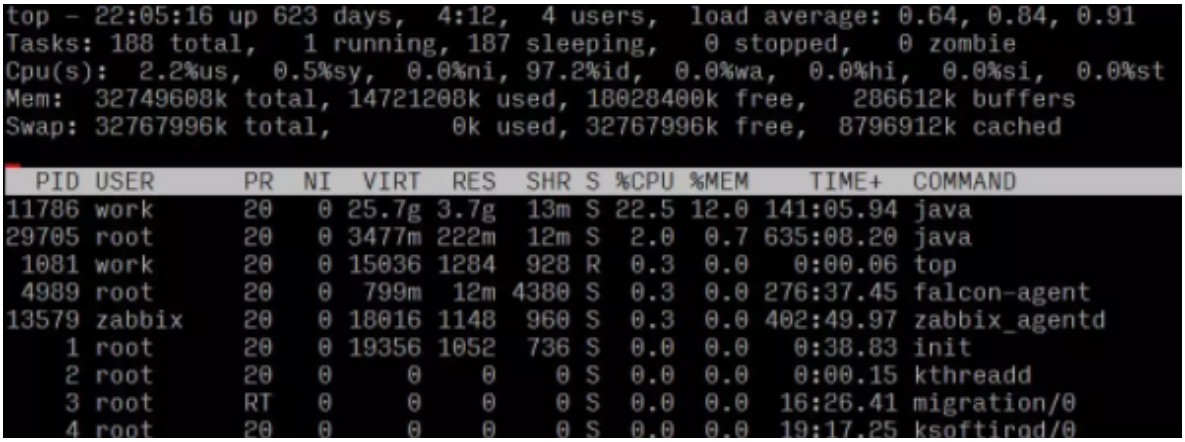
经过前面的分析与排查，最终定位到一个时间工具类的问题，造成了服务器负载以及cpu使用率的过高。

- 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$1+6+60+60n次=36,000$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。

3、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。



CPU负载过高异常排查实践与总结CPU负载过高异常排查实践与总结

4、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题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。

IT-Linux