**原创；微信公众号：千里行走；**

**受限图片大小限制，有些图片不是很清晰，可以到微信公众号查看；**

在一次业务升级后，发现服务边的不稳定，zabbix各项监控指标相对上线前异常上升。

通过prometheus等监控发现是redis调用大增，看代码在循环里对smember有大量调用。

**目录：**

**(1).原因**

**(2).定位过程**

**(3).解决方式**

**(4).总结**

1.循环里不要调用redis

2.找到一个办法或者参照系去衡量一个服务的cpu健康状态

**(1).原因**

循环中大量对redis的单次调用会产生大量的redis/io调用，导致业务JVM和redis的cpu的线程上下文和软中断同时飙升。

**业务JVM线程上下文和软中断同时飙升原因：**

smember的时间复杂度是O(1)，很快，jvm大量调用smember后，相关IO也会很快得到返回数据从而进入就绪状态，那么从就绪IO读取数据的线程会始终处于繁忙状态。

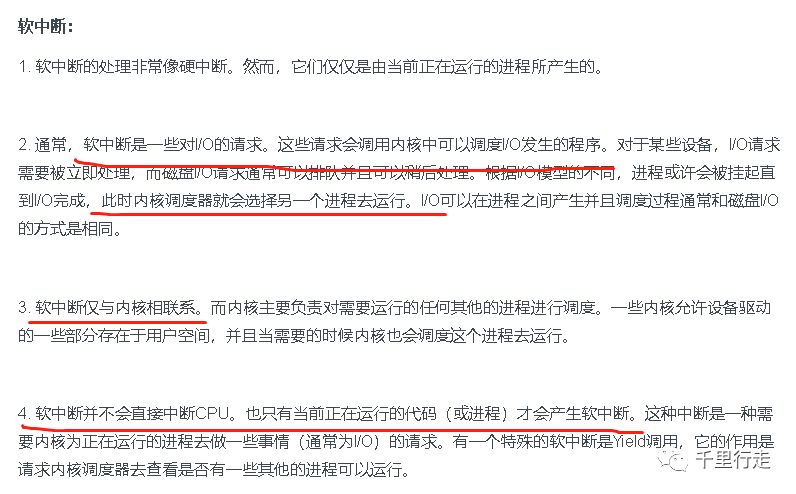
而且我们的cpu核数不多，information只有2core，而且jvm访问redis是同步读取，是block的，调用后进入block，cpu必然要切走线程做别的事。

大量/频繁/反复的会进入cpu调度/线程切换的过程中。

大量cpu调度/线程切换–>导致cpu %sy（内核占用cpu时间）飙升。

大量的io事件->导致cpu %si（软中断占用cpu时间）飙升。

关于软中断：

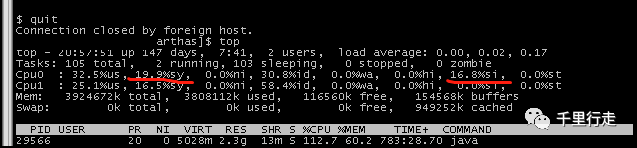


**(2).定位过程**

循环中大量对redis的单次调用会产生大量的redis/io调用，导致业务JVM和redis的cpu的线程上下文和软中断同时飙升。

top命令：

可以看到%sy(cpu内核时间)，%si(cpu软中断时间)非常高，实际上在繁忙时段，%us, %sy, %si可以各占1/3，这个对于%sy, %si非常非常高了，我们希望cpu时间尽可能多的用于%us（cpu 用户态时间）。

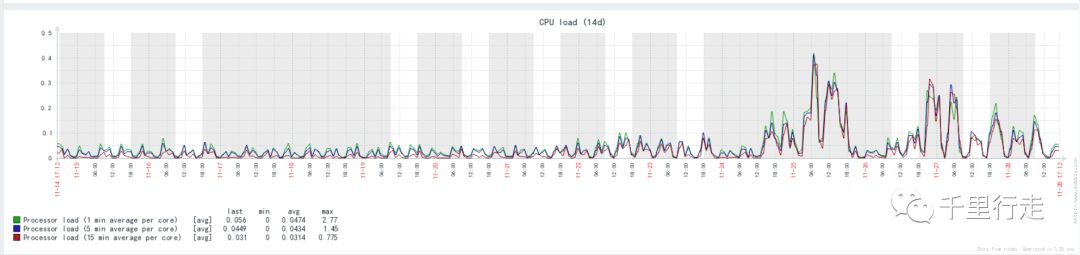


zabbix的cpu相关指标：

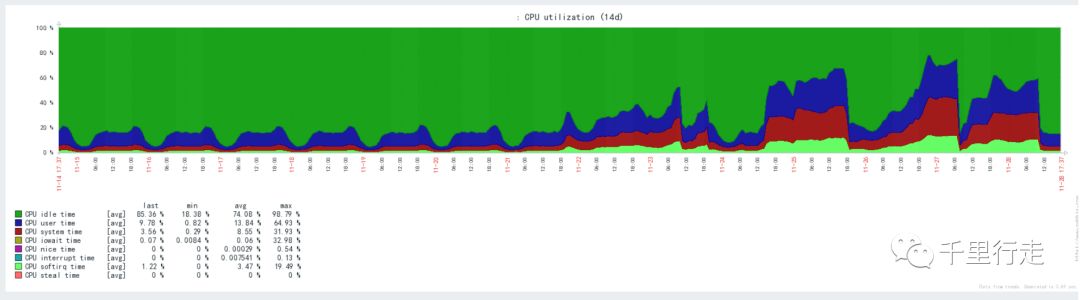
相关服务的cpu jumps：绿线是上下文切换频率；蓝线是中断频率。



cpu load：相对之前有很大提升。



cpu利用率情况：可以很明显看到cpu的%sy，%si飙升，之所以没有影响线上服务，是因为us%时间够用，仔细观察，%us时间和之前的差不多。

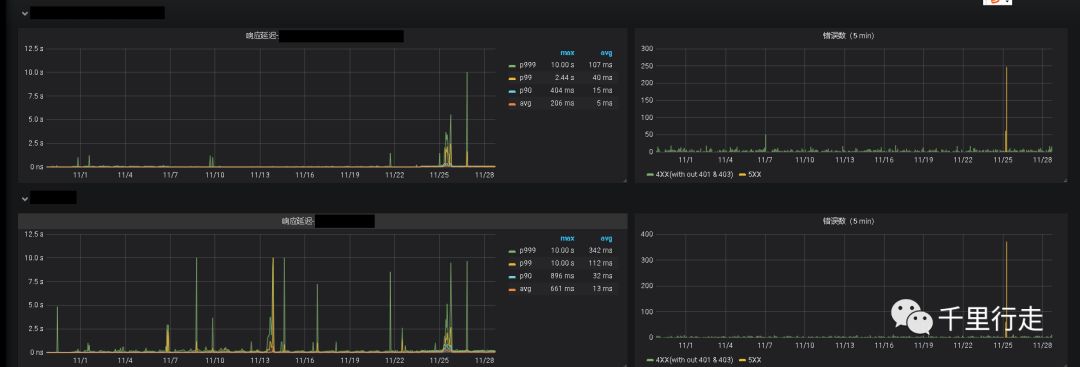


服务线上接口情况：

qps和错误率正常基本没有变化：



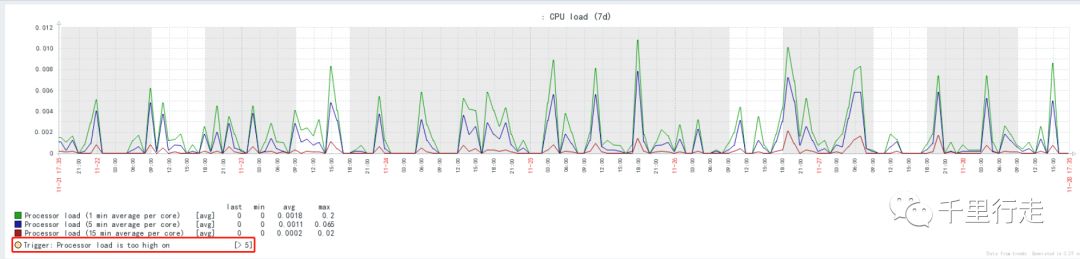
接口的延迟时间和往常基本一致：



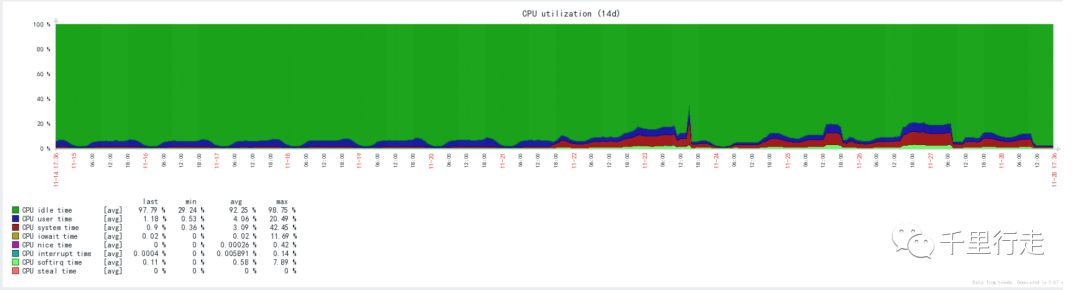
服务的cpu jumps：



redis实例的cpu load：



redis的cpu 利用率：可以很明显看到cpu的%sy，%si飙升。



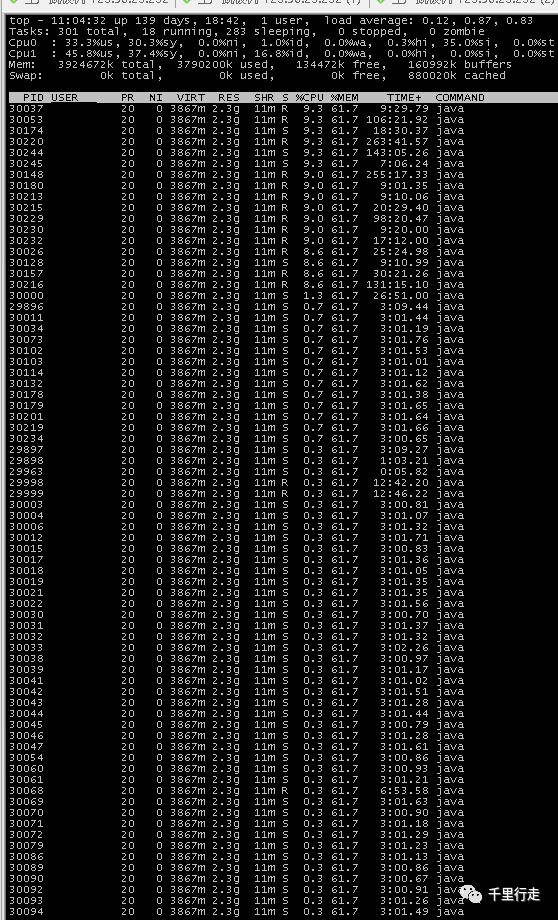
观察到这里，只是确定了cpu的使用确实出现了很大的问题，那么接下来我们需要确认cpu为什么高，由何引起。

另外，为什么要先观察grafana和zabbix呢：因为我需要确认一件事：是否影响了线上，决定我是否要回滚，因为回滚代价很高，回滚后不方便定位问题，如果不影响线上，可以短时间内接受。

用两个方法确认这个事情（但可惜最初我没信这个结果，坚持认为redis的smember不会引发，因为速度很快嘛是O(1)，但是忽略了IO）：

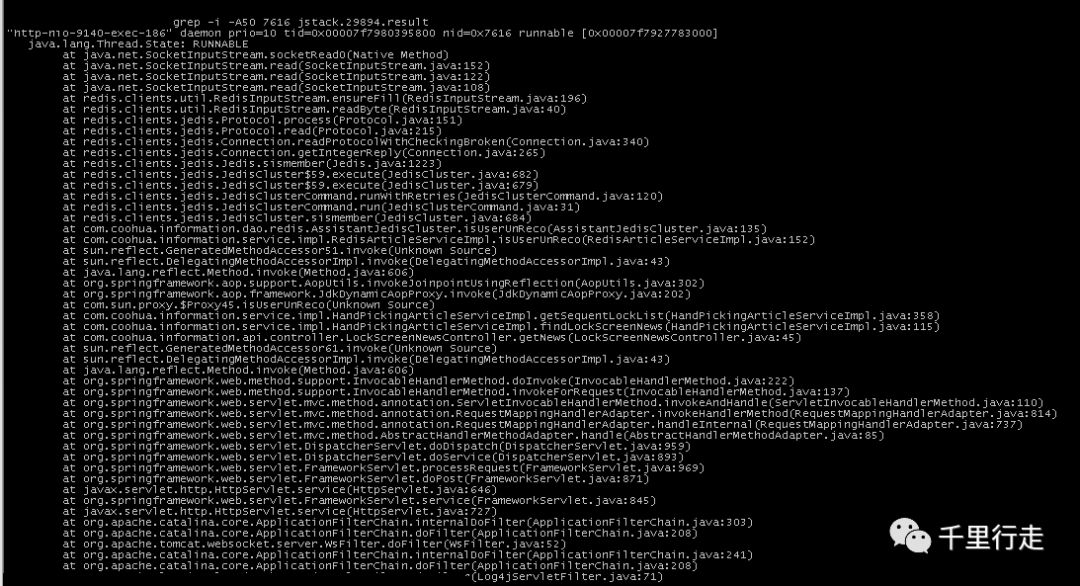
方法一：top -H -p pid

看某个linux下的进程下有哪些线程以及这些线程占用的cpu时间，找到耗费cpu时间最高的线程。



然后将cpu使用率最高的线程id转换为16进制，在jstack pid的结果里去查看这个线程的堆栈：

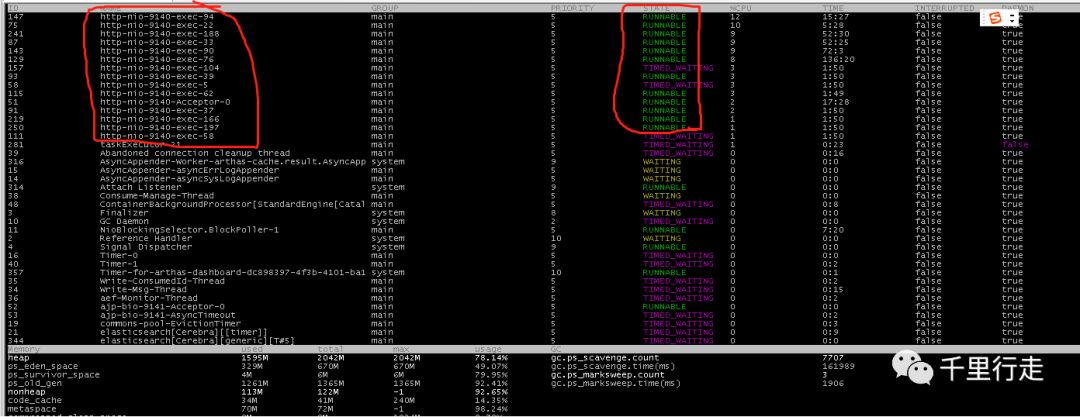
可以很清晰的看到smember的调用，这次优化上线只有这一处改动，其实这个时候已经可以确认问题了。



方法二：阿里的arthas

arthas安装&使用：https://github.com/alibaba/arthas  这个安装需要高权限，需要运维同学帮忙安装。

使用dashboard命令，有些新东西，有那么7,8个线程始终是RUNNABLE状态，且“永远是”，并且是高耗cpu的线程，一直RUNNABLE说明一直在切换（RUNNING是很难捕获到的）。



再用thread -n 10查找cpu耗损前10的线程堆栈，这个我没有保存现场图，看到的和之前方法一是一样的堆栈，也是smember的堆栈。

**原因确认：由于大量调用redis导致io事件爆炸–>从而造成cpu忙不过来。**

**(3).解决方式**

使用smembers一次获取多个数据，JVM内存里运算，规避大量IO事件。

**(4).总结**

**1.循环里不要调用redis**

但是像smember这类exist一定是封装在底层的，上层一层层调用很有可能放大，这个只能是在使用的时候注意，同时观察线上的zabbix监控，主要是cpu是否有异动。

我们还在自研框架中集成了prometheus，全方位监控redis的调用。

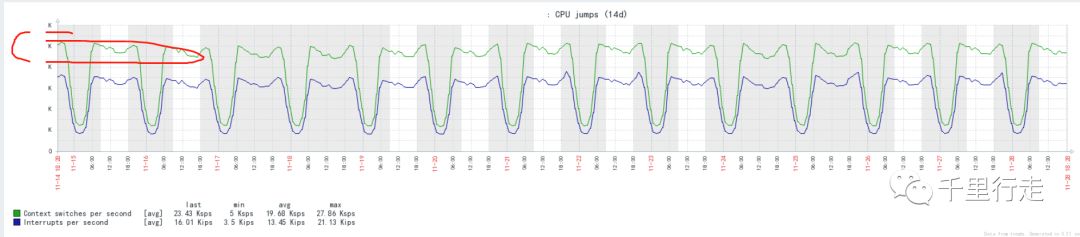
**2.找到一个办法或者参照系去衡量一个服务的cpu健康状态**

有的。分两部分来看：

2.1.对于redis的cpu健康状态，我们可以参照redis-passport的cpu指标

只有cpu jumps可以参照，cpu利用率和cpu时间取决于具体怎么用redis，差异太大，看各自的历史曲线。

因为按照正常业务来讲，每次验票都要调用一次redis，这个访问是省不了的，一定量很大，正常服务很难有比这个还高的。如果别的redis的cpu超过了redis-passport的cpu指标，那就需要注意了。下图关键数据脱敏。



2.2.对于nginx来说

由于nginx使用的是epoll，所以软中断个数要远大于线程切换次数，这也是nginx性能很高的一个原因。



2.3.对于一般的应用：

上下文切换次数要高于中断次数。

