**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；**

**目录**

**(1).性能测试方案**

1.理想测试方案

2.实际测试方案&测试结果

**(2).测试机型&资源分配**

**(3).相关监测数据**

1.整体概览

2.broker-master-1节点监控数据

3.broker-slave-1节点监控数据

4.consumer1节点监控数据

5.施压机(producer)监控数据

**(4).最终选型**

**(5).总结**

1.不需要担心rocketmq的处理能力/TPS简单估算方法

2.不需要使用sync\_master

3.我们为什么选择async\_master，async\_flush

4.我们需要让开发同学低成本的使用rocketmq

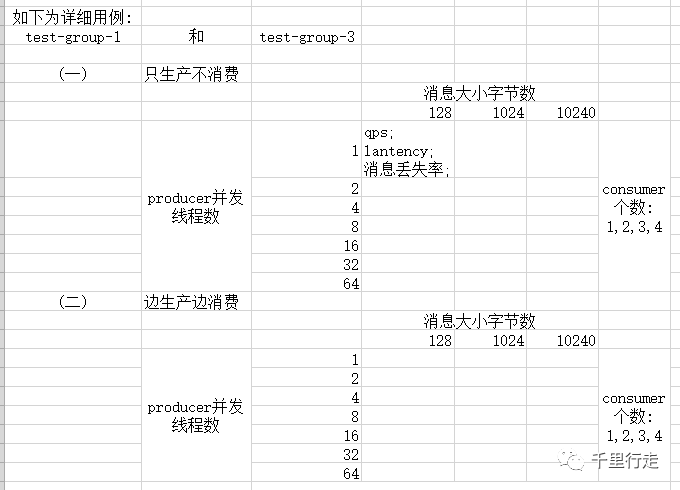
5.以后也会将rocketmq容器化

**正文**

**(1).性能测试方案**

**1.理想测试方案**

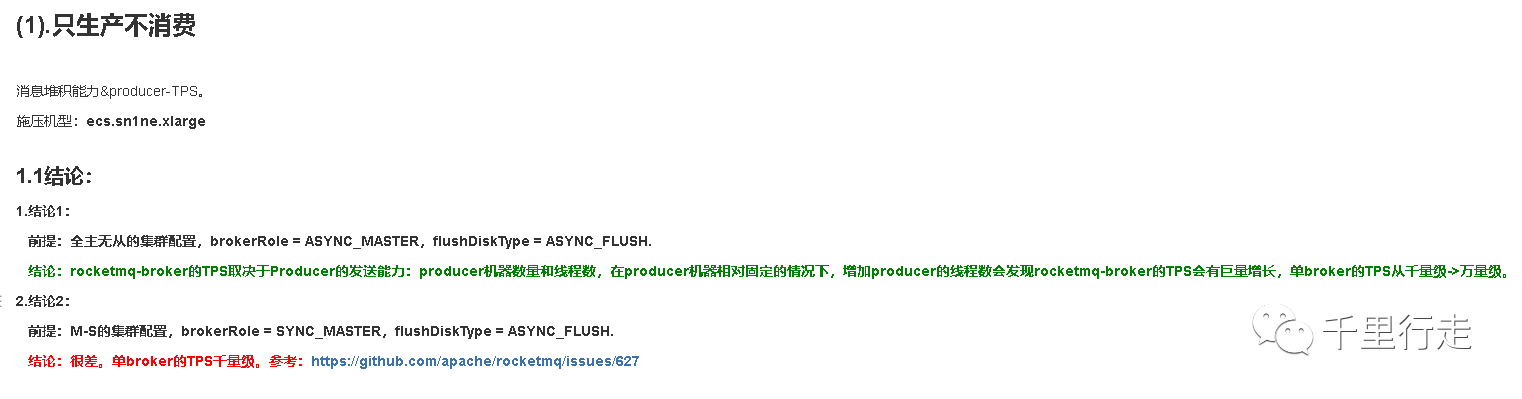


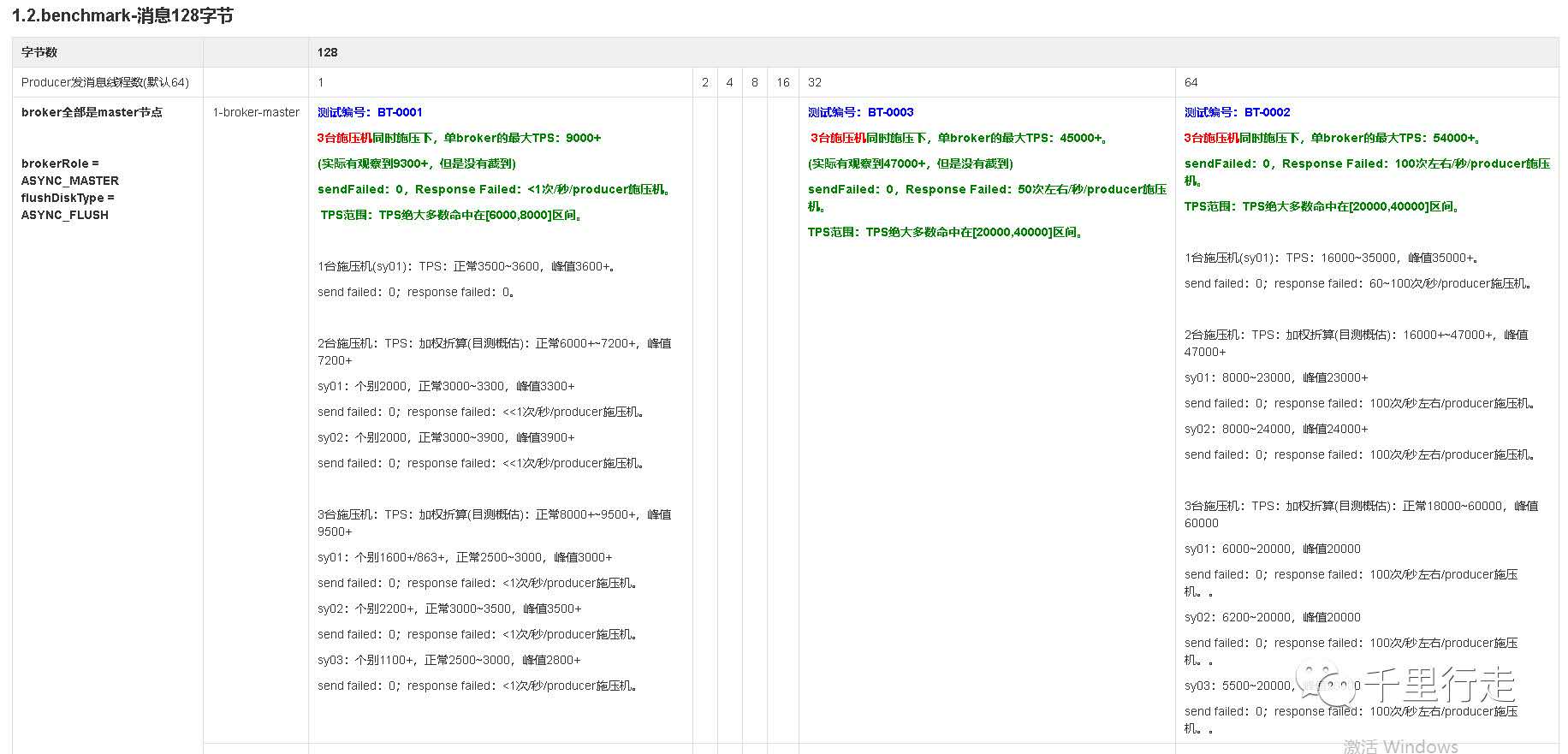


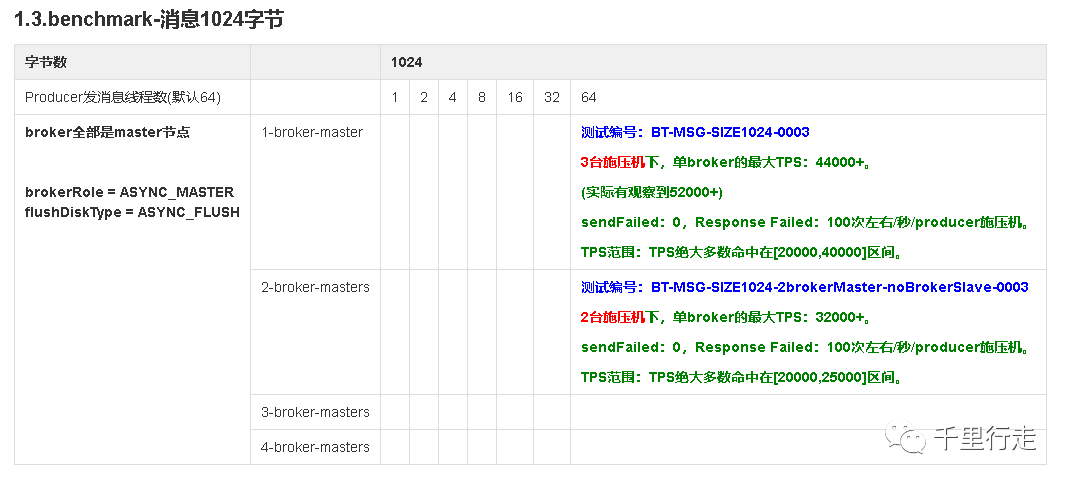
关注qps和lantency即可，消息丢失需要使用者在开发时处理，比如消息发送加重试机制(这里有讲究，也不是随便写的，也涉及到rocketmq-broker的流控机制，下一篇聊)。

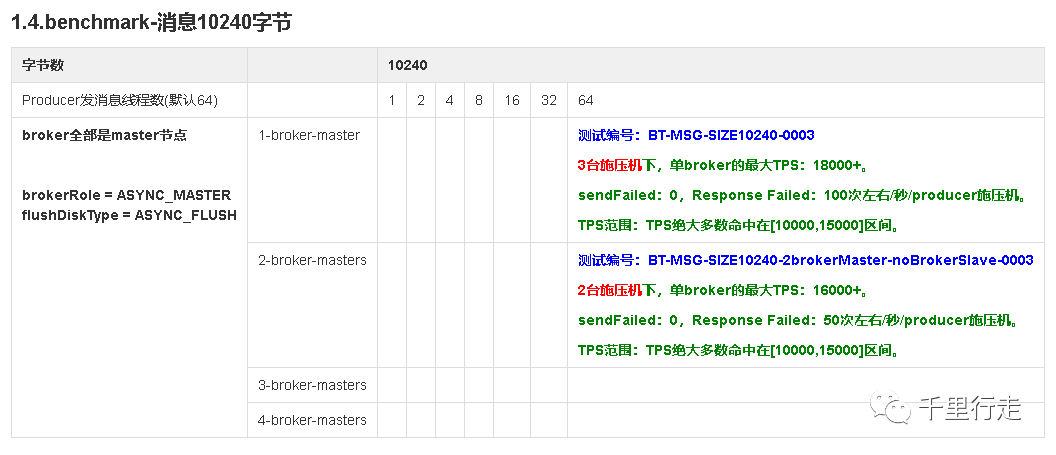
但实际上，不可能这样操作，原因：机器资源占用太多，最主要的是时间不允许。

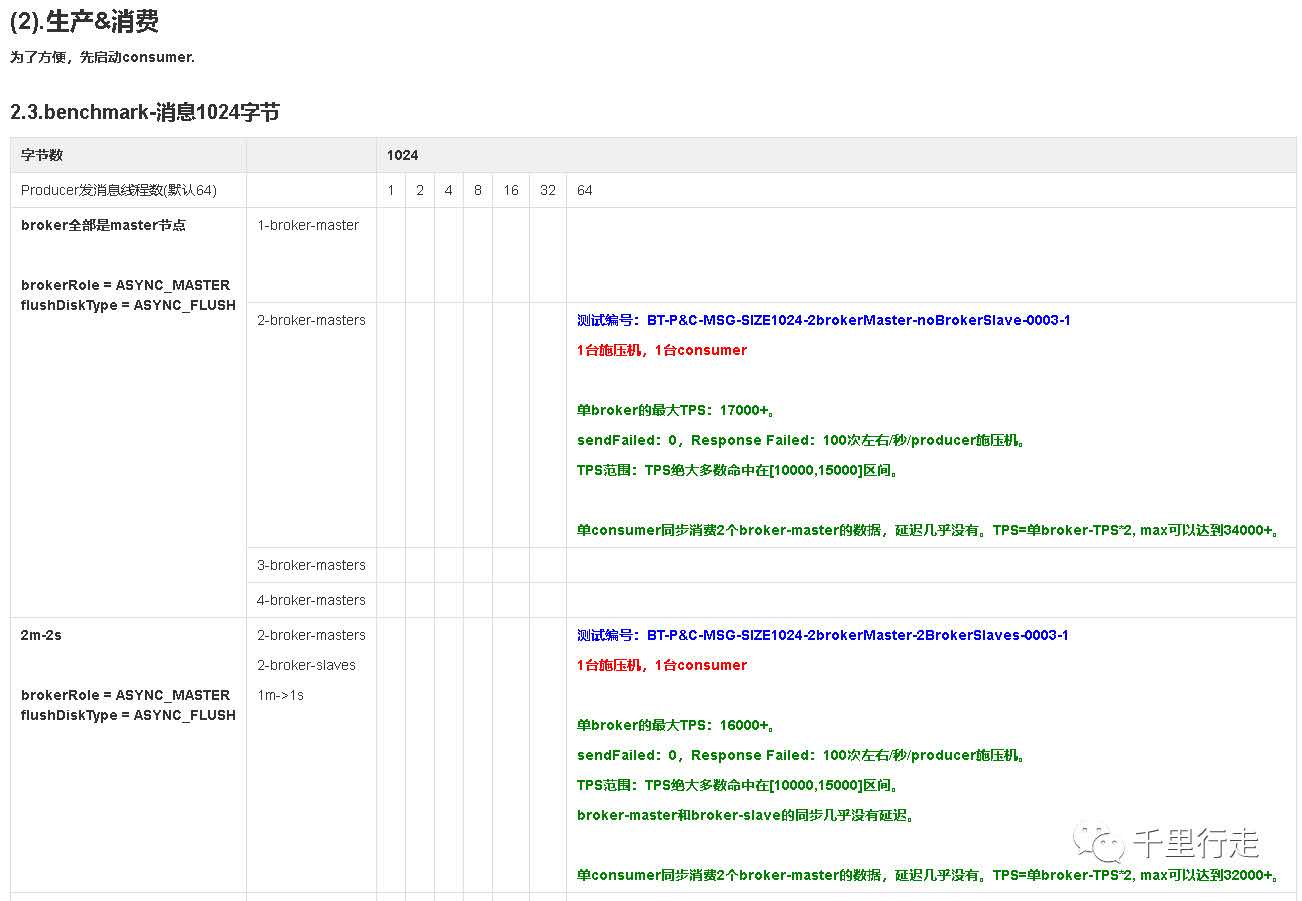
**2.实际测试方案&测试结果**

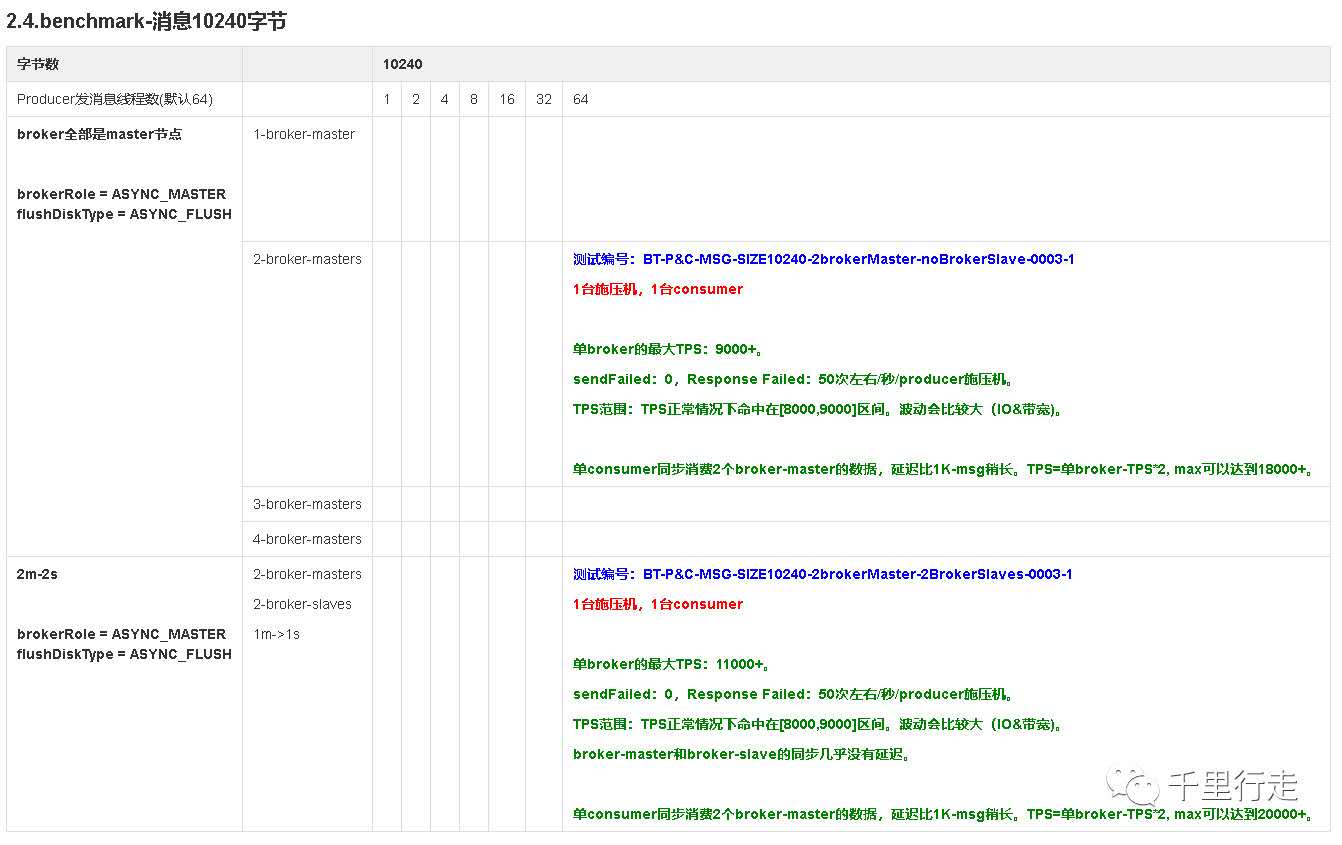












**(2).测试机型&资源分配**

全部使用阿里云的ecs.sn1ne.xlarge机型，都是4core8G，相对来说性价比最高，网络有加强，内核参数有优化，如下：



使用rocketmq默认提供的benchmark脚本工具进行压测。

topic：BenchmarkTest

queue：1024/broker

**(3).相关监测数据**

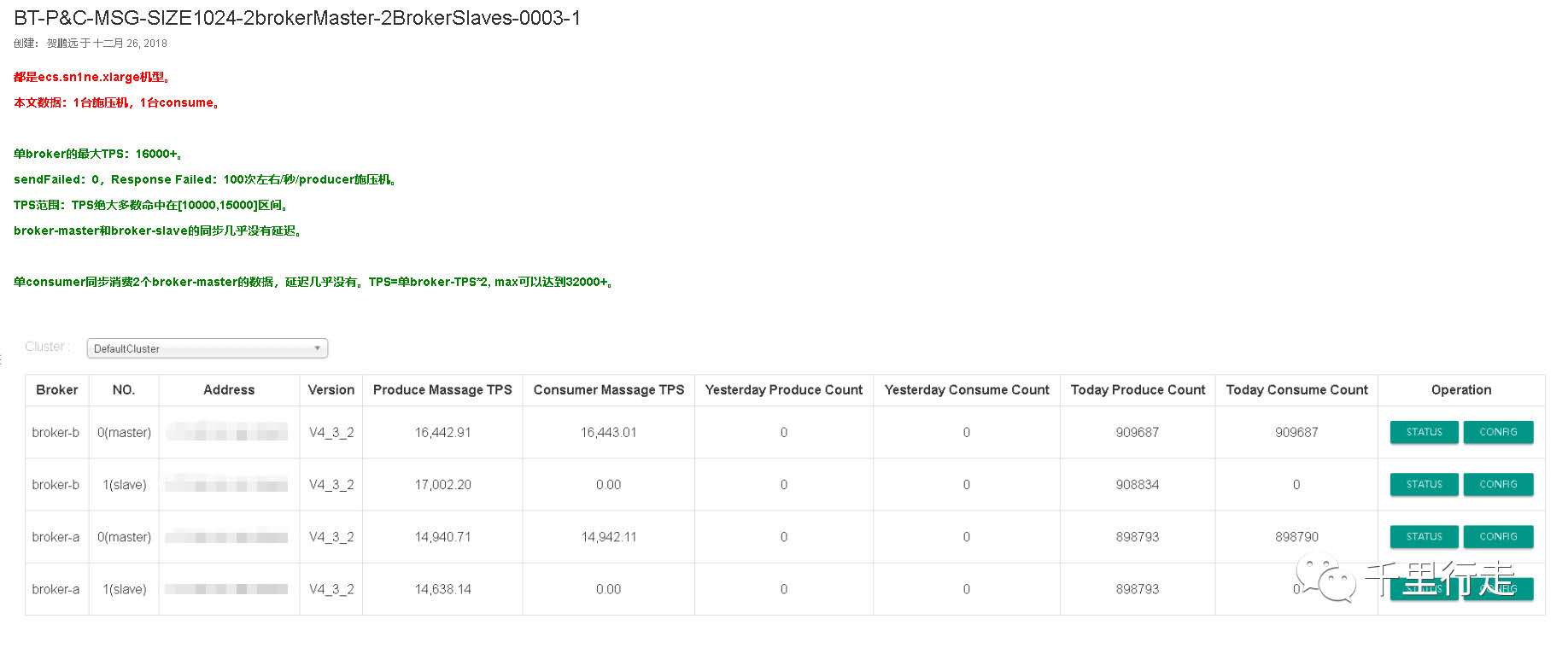
以测试用例BT-P&C-MSG-SIZE1024-2brokerMaster-2BrokerSlaves-0003-1为例：

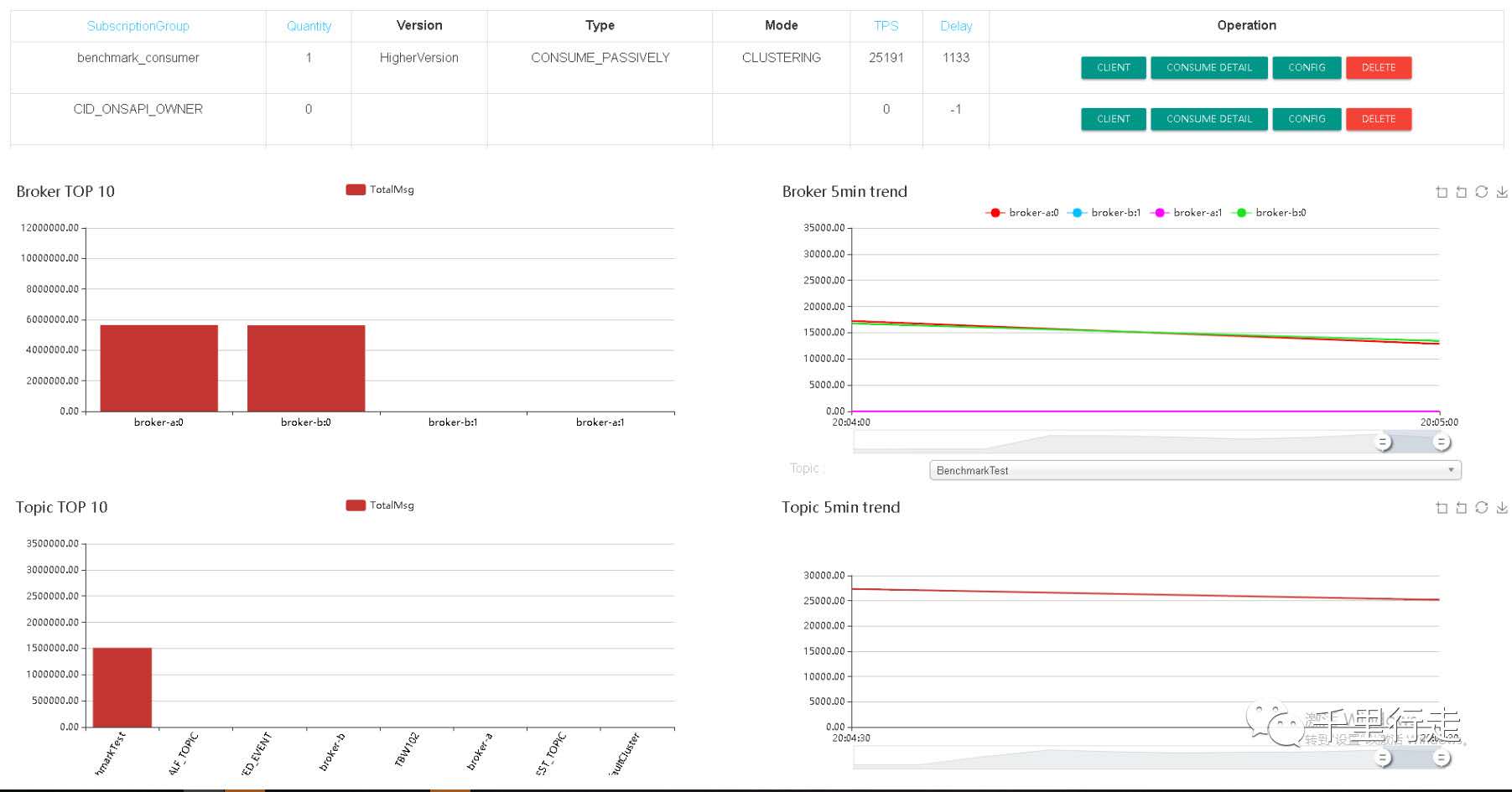
2个brokerMaster, 2个brokerSlave，且broker配置为：

brokerRole = ASYNC\_MASTER

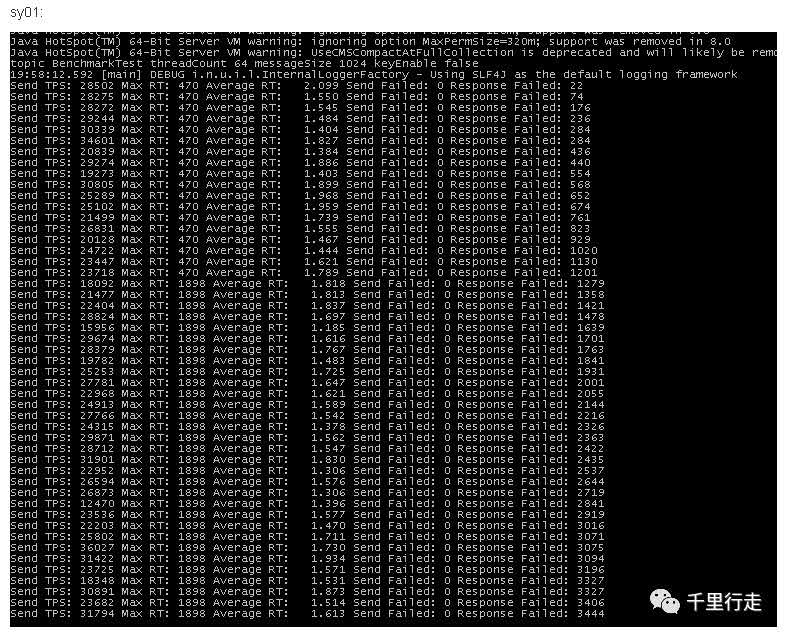
flushDiskType = ASYNC\_FLUSH

**1.整体概览**

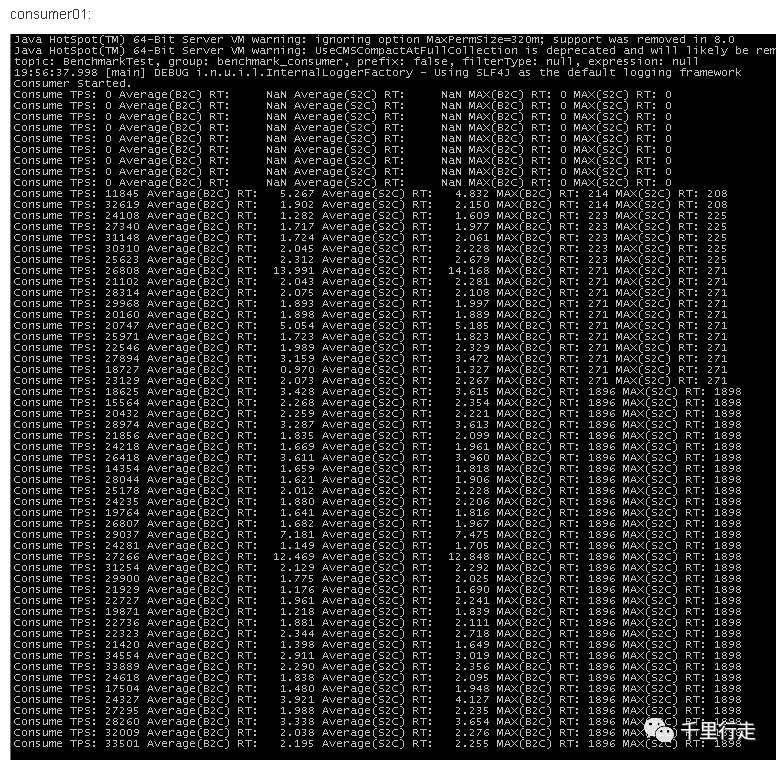




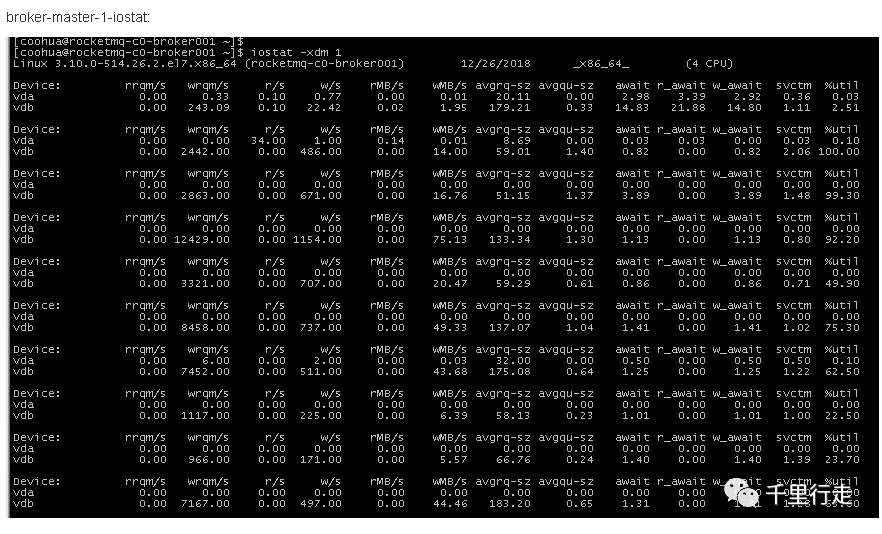
施压机消息发送情况：



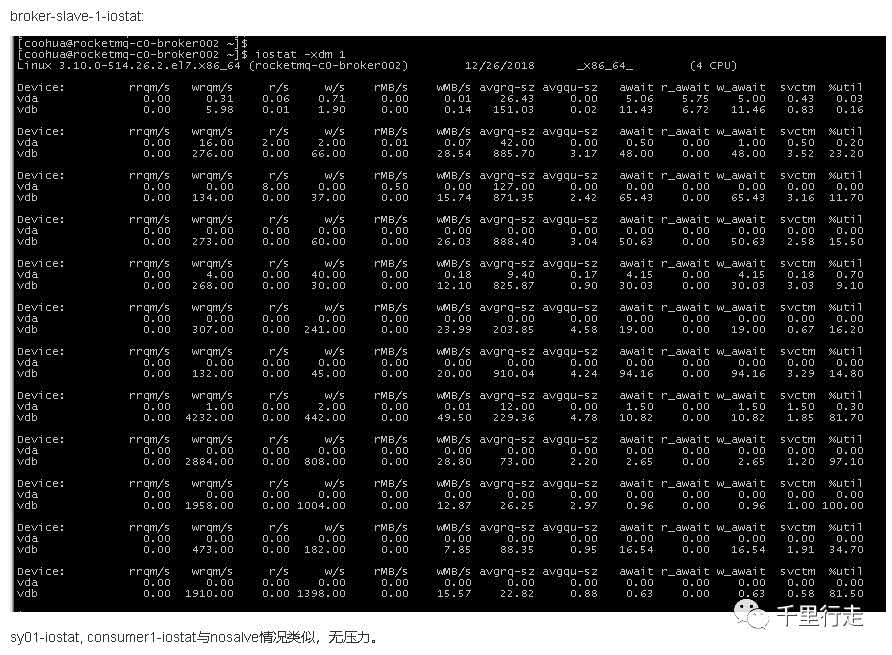
消费者消费情况：



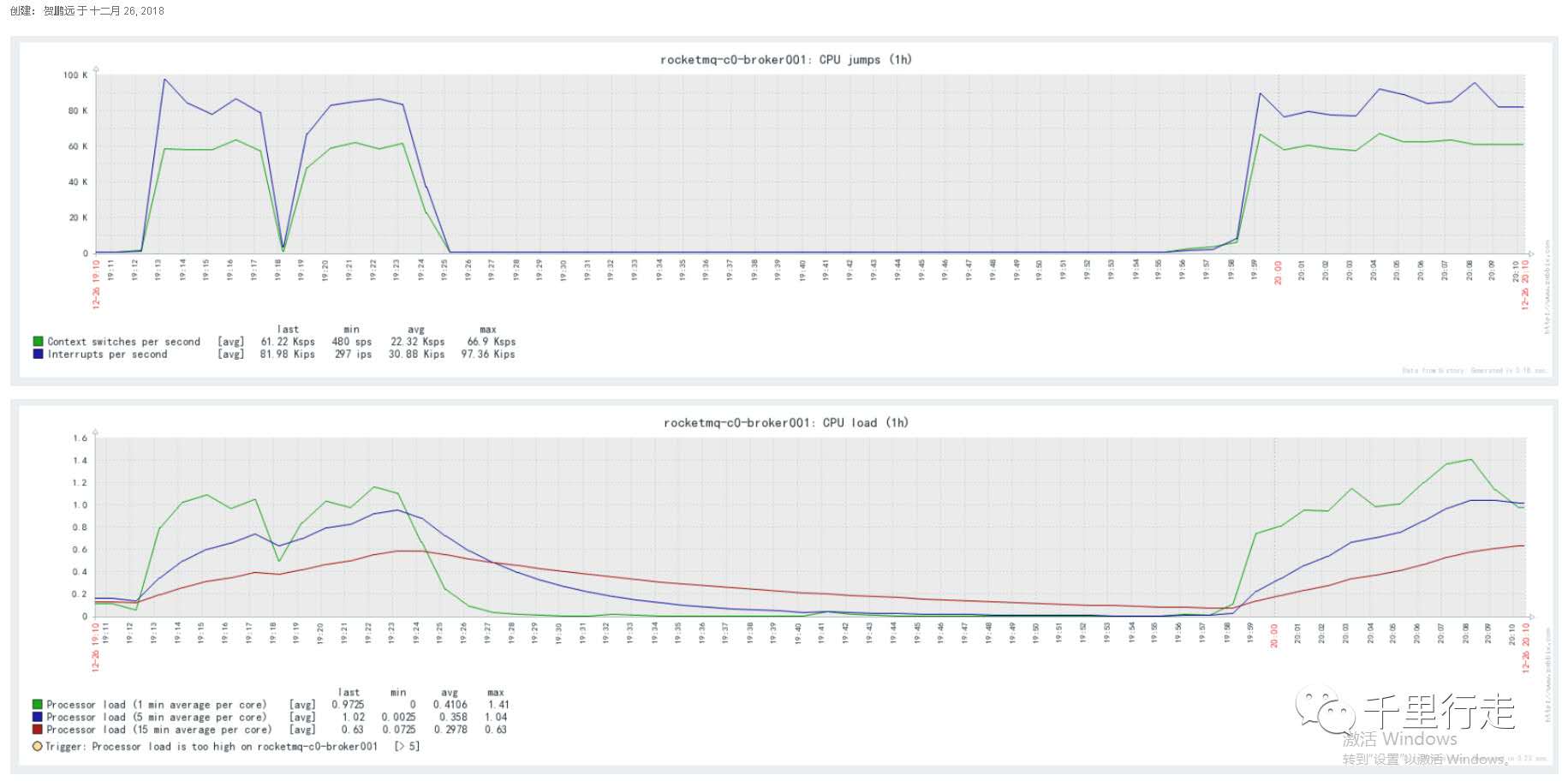
broker-master-1节点的iostat：

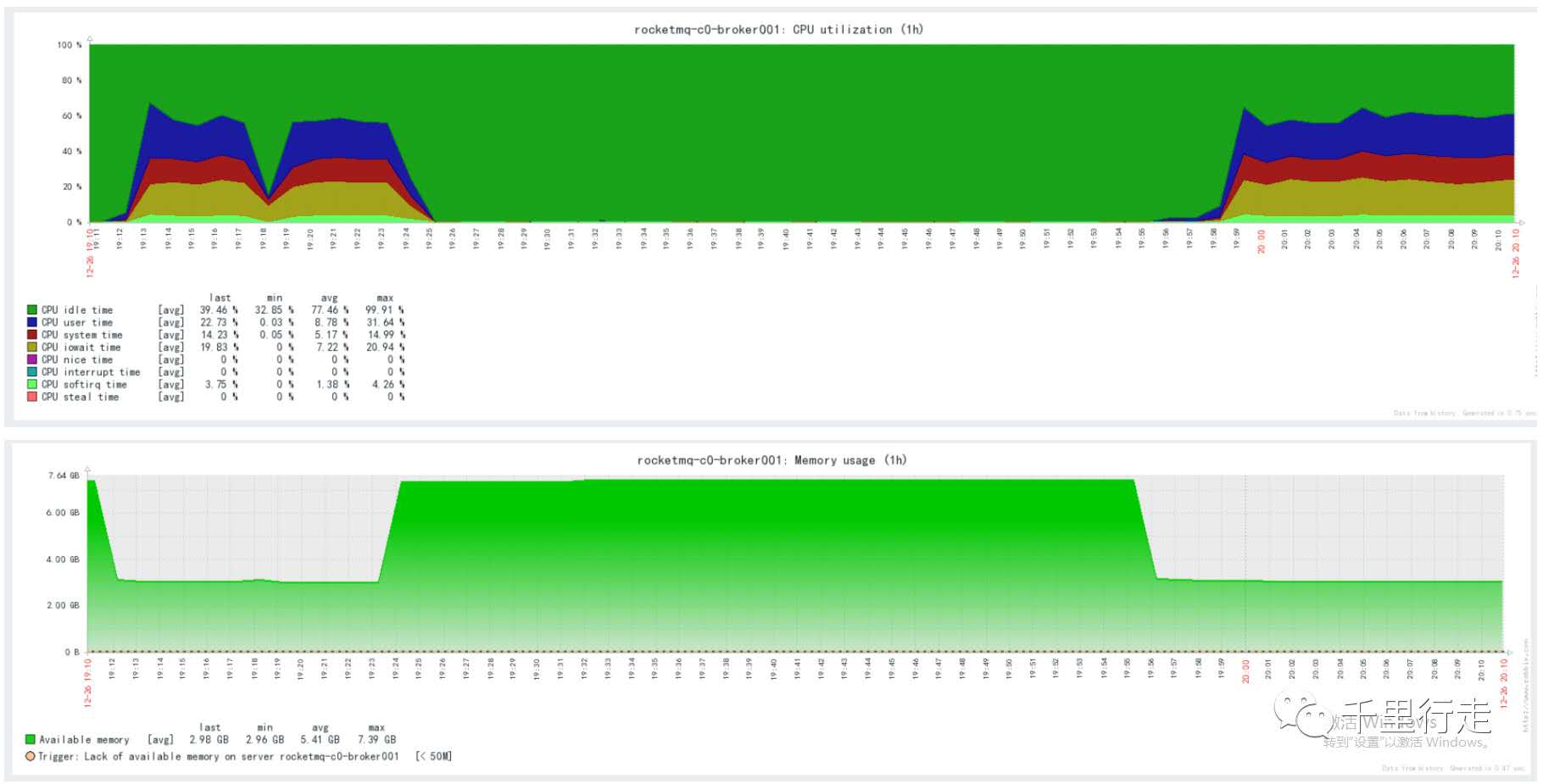


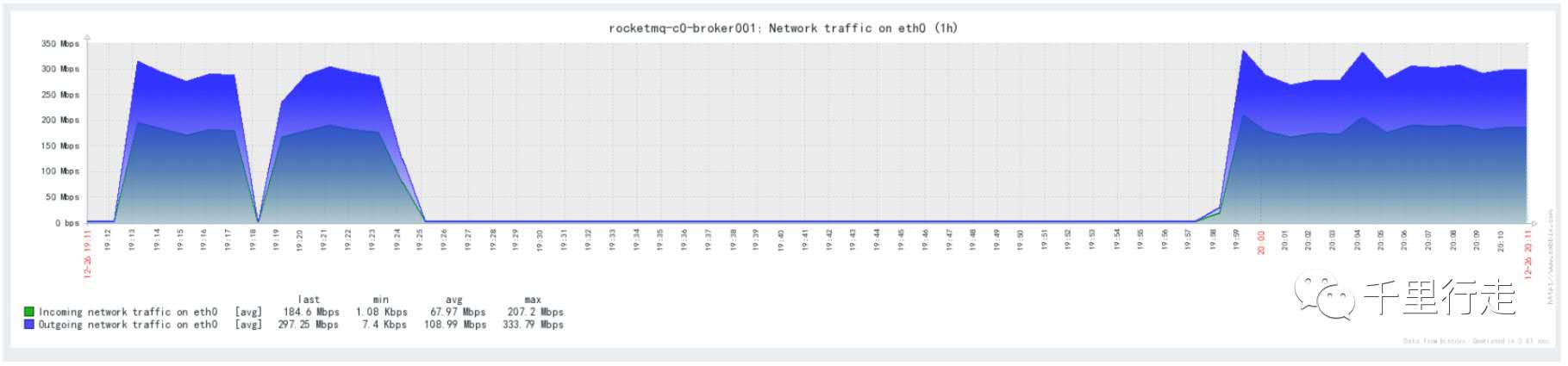
broker-slave-1节点的iostat：



**2.broker-master-1节点监控数据**

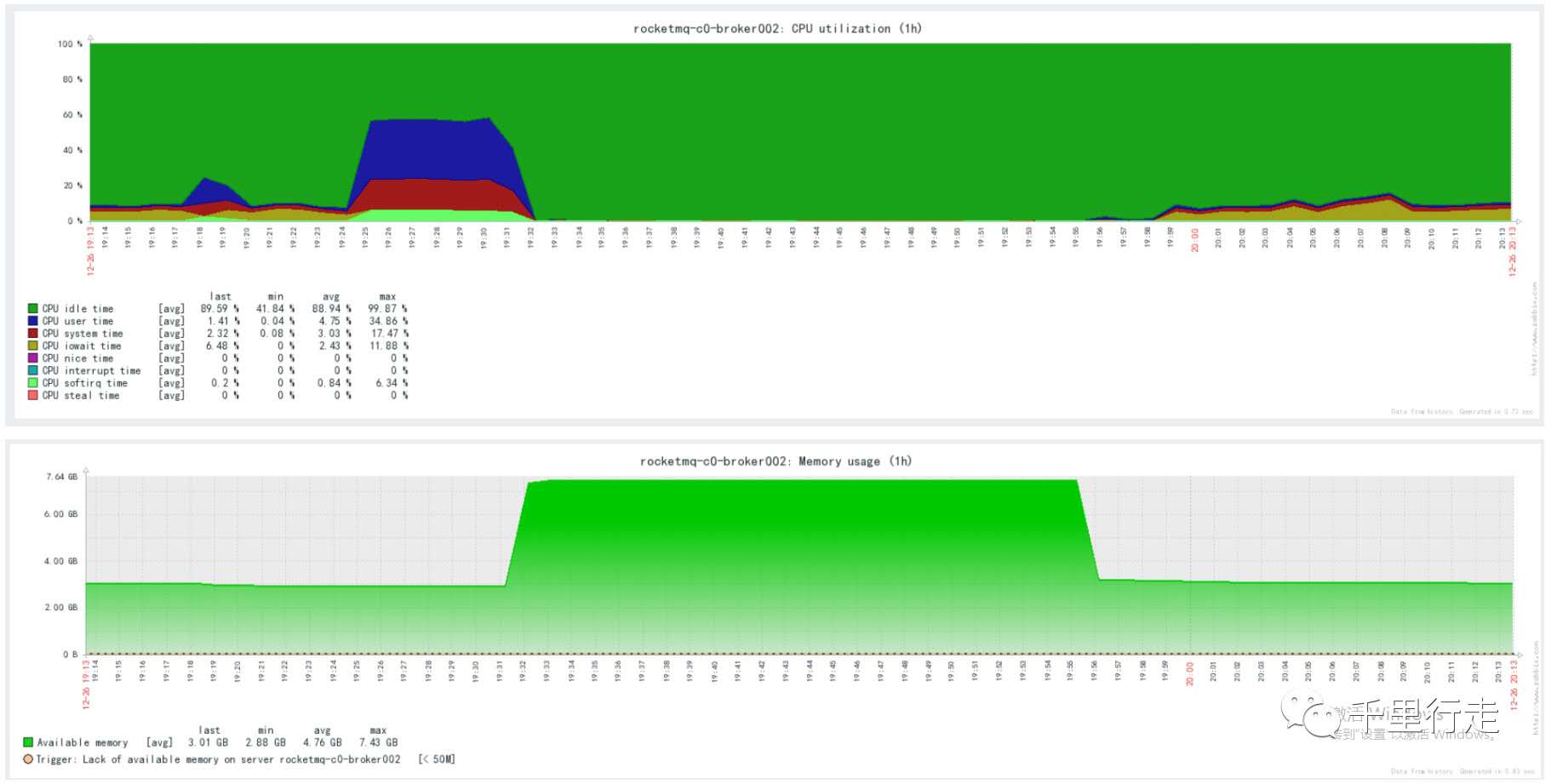


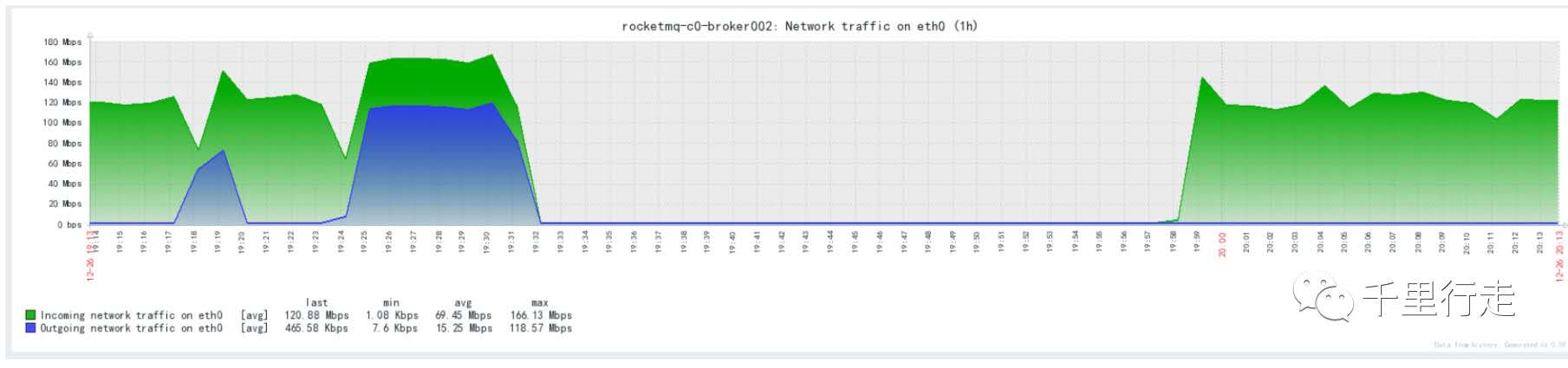




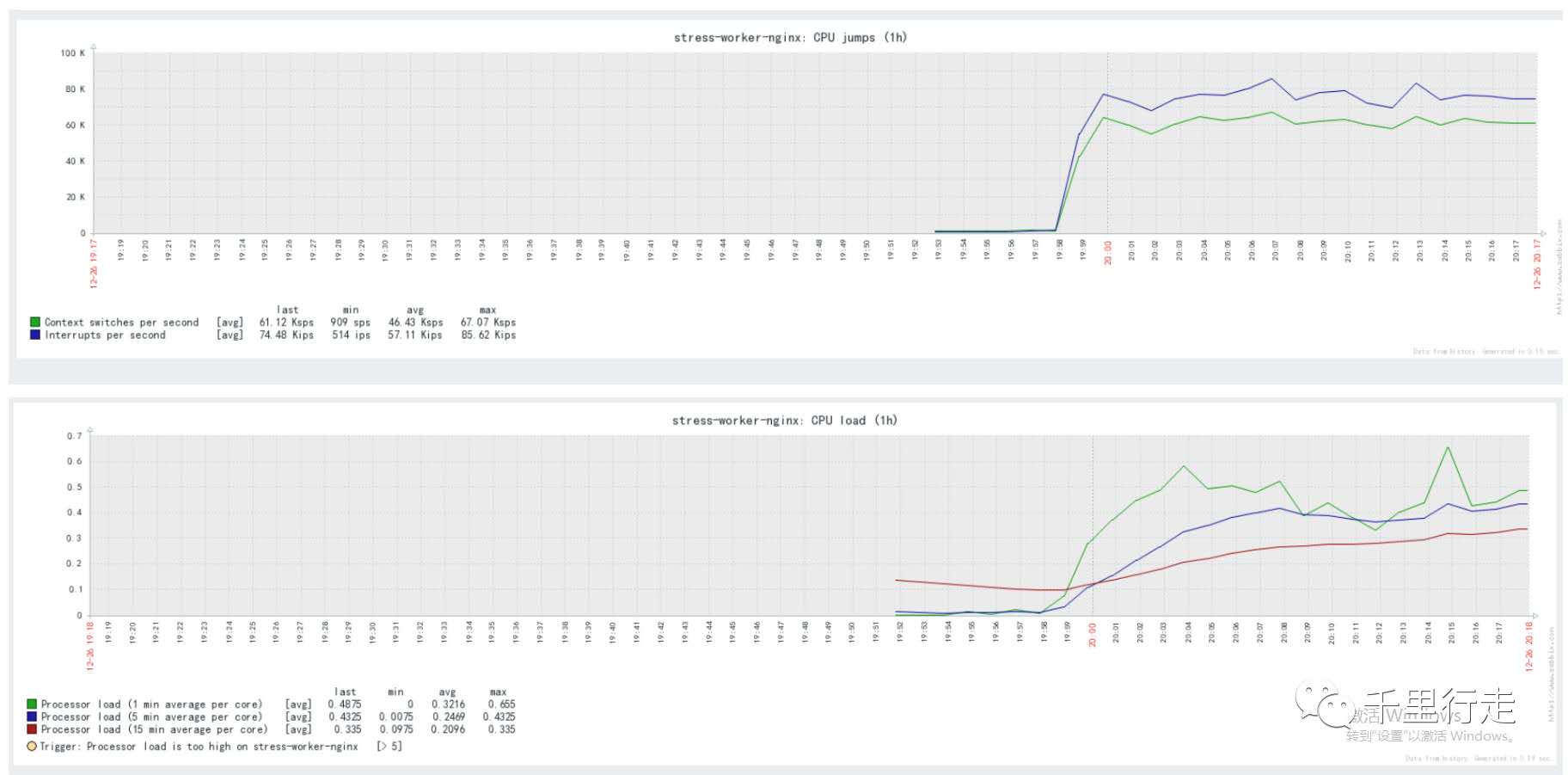
**3.broker-slave-1节点监控数据**



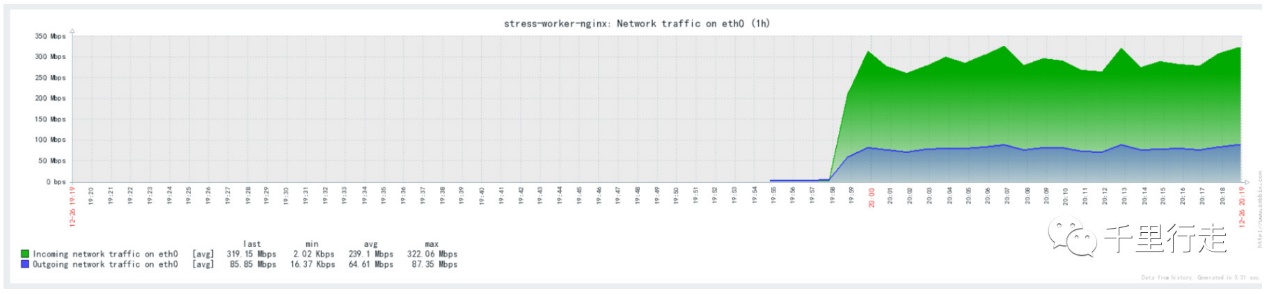




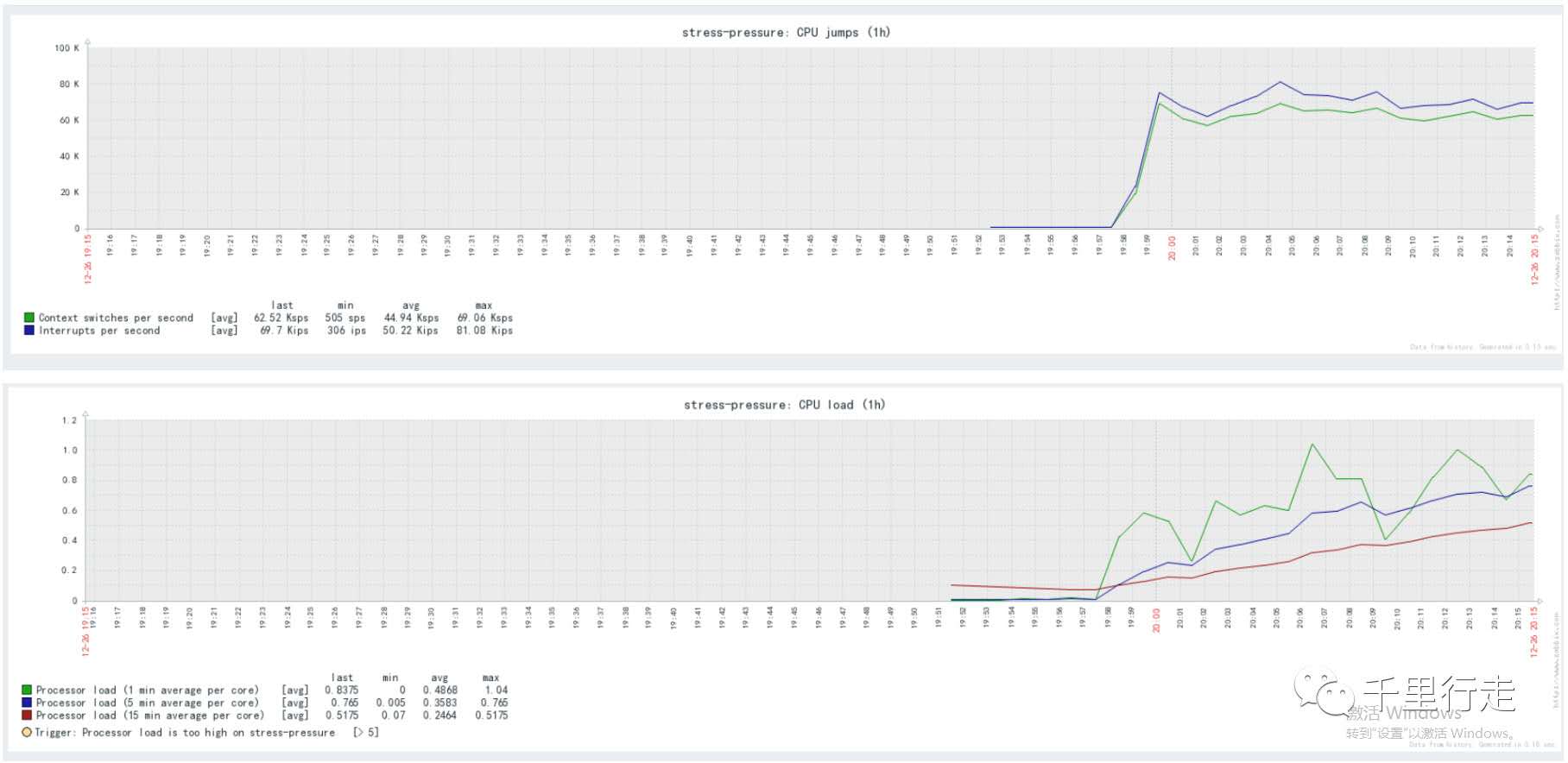
**4.consumer1节点监控数据**

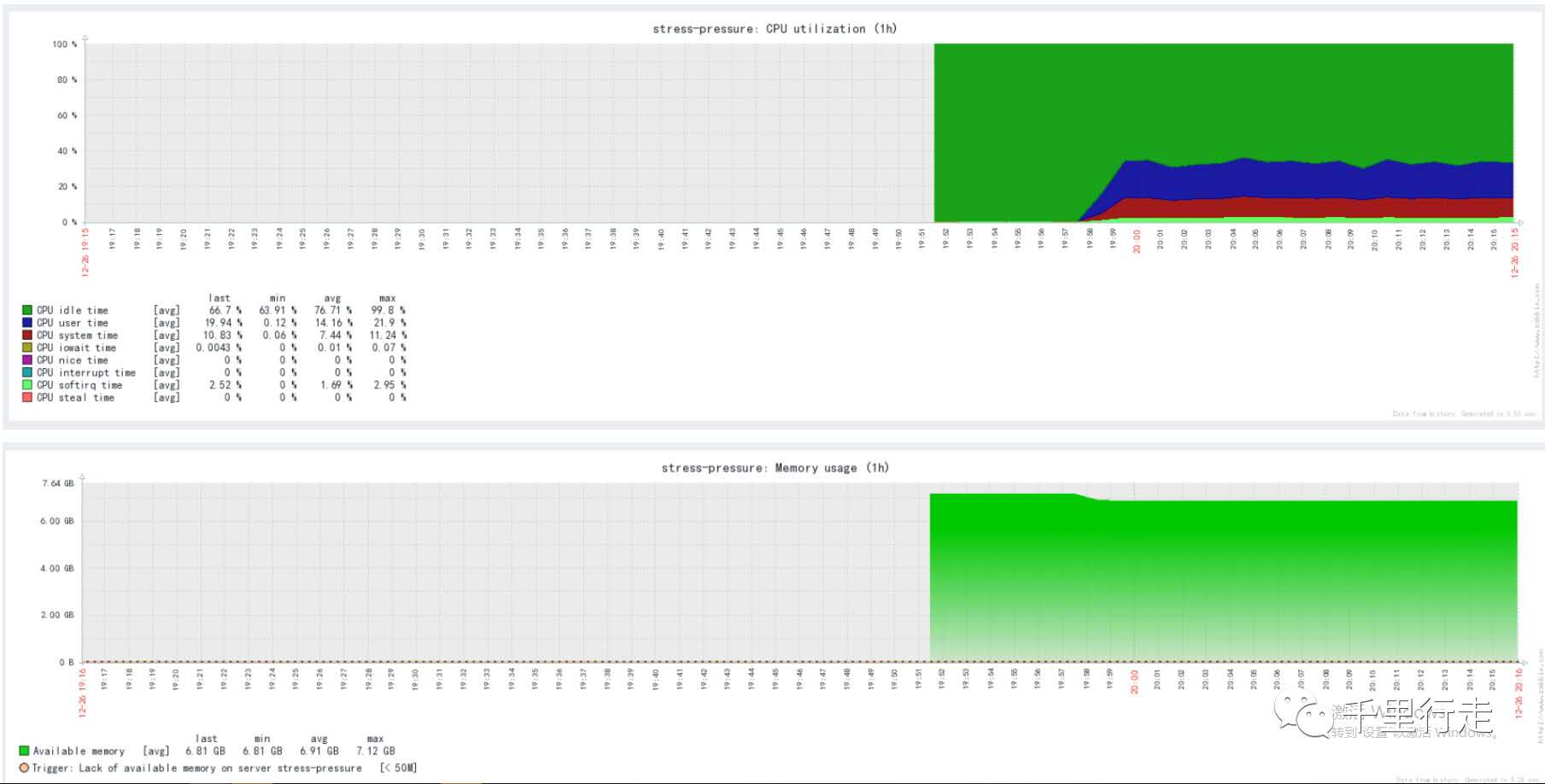






**5.施压机(producer)监控数据**







**(4).最终选型**

机器选型：



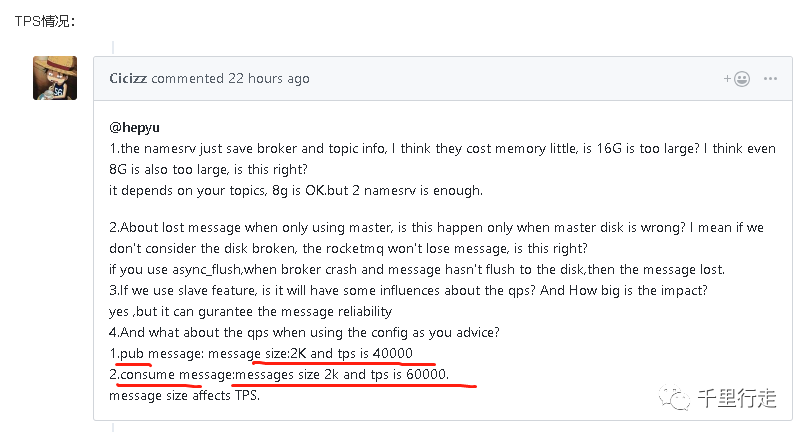
磁盘选型：



这里有一个问题：实际上不需要这么大的盘，100GB的SSD足够用了，后续会替换为100GB的SSD，成本更划算，而且TPS和lantency会更加漂亮；消息发送的重试次数会大幅减少。

附，官方关于资源的分布反馈：





**(5).总结**

**1.不需要担心rocketmq的处理能力/TPS简单估算方法**

关于TPS的计算很简单（async\_master，async\_flush）：

在使用物理磁盘的前提下，max(broker-master单节点 TPS)=物理磁盘的最大写入速度/消息大小=max(rocketmq producer send TPS)

在使用SSD盘的前提下，max(broker-master单节点 TPS)=SSD盘的最大写入速度/消息大小=max(rocketmq producer send TPS)

另外，还要考虑带宽限制，三者大致关系：tps(物理磁盘)<tps(带宽)<tps(SSD盘)。

我使用1KB的消息测试其实已经很大了，实际生产环境不会这么用，消息体都是尽可能小的。

**2.不需要使用sync\_master**

会严重降低TPS，我的测试结果是直接降了一个数量级；而且也没有必要，除非是金融等高要求的场景必须保证副本，开启后多挂几组broker-master/slave，增加rocketmq的并行吞吐能力，提高TPS。

**3.我们为什么选择async\_master，async\_flush**

很简单，不是金融等高要求场景，而且这种模式下的可用性其实也是非常高的，更关注TPS，且同时关注成本。

但要注意前提是业务code要正确处理消息重试，消息重复消费，这个不是rocketmq保证的，以后会聊一聊这方面。

rocketmq的async\_master和async\_flush类似于kafka的acks，类似但不等价。

**4.我们需要让开发同学低成本的使用rocketmq**

我们的做法是自己开发了一个框架，完全是注解是开发，将rocketmq的producer和consumer封装到框架里，配置都在apollo，开发同学使用时直接加几个注解即可完成producer，consumer的对象实例化，非常方便，不会出错。

而且还集成了prometheus，可以将producer和consumer的发送全程，消费全程监控起来，比如TPS过高时，会触发rocketmq的流控，直接将msg写入请求拒绝且不会重试，此时要在框架中自己实现重试机制，且加入prometheus监控：



直接监控到节点的进程，哪个JVM实例发生rocketmq使用异常一目了然。

prometheus其他监控：



目前暂时只开发了两个维度：消息发送全程监控(粒度到进程)，消息堆积数监控(对于rocketmq集群，粒度到broker的queue；对于业务jvm，粒度到进程)。

非常有助于rocketmq的正确使用，和问题发现。

另：

官方也有一个rocketmq的prometheus-exporter(但是维度不全)，也会使用：

https://github.com/apache/rocketmq-externals/tree/master/rocketmq-prometheus-exporter

**5.以后也会将rocketmq容器化**

放入K8S自带守护，正在进行中。

