Kafka vs RocketMQ—— Topic数量对单机性能的影响

引言

上一期我们对比了三类消息产品(Kafka、RabbitMQ、RocketMQ)单纯发送小消息的性能，受到了程序猿们的广泛关注，其中大家对这种单纯的发送场景感到并不过瘾，因为没有任何一个网站的业务只有发送消息。本期，我们就来模拟一个真实的场景：

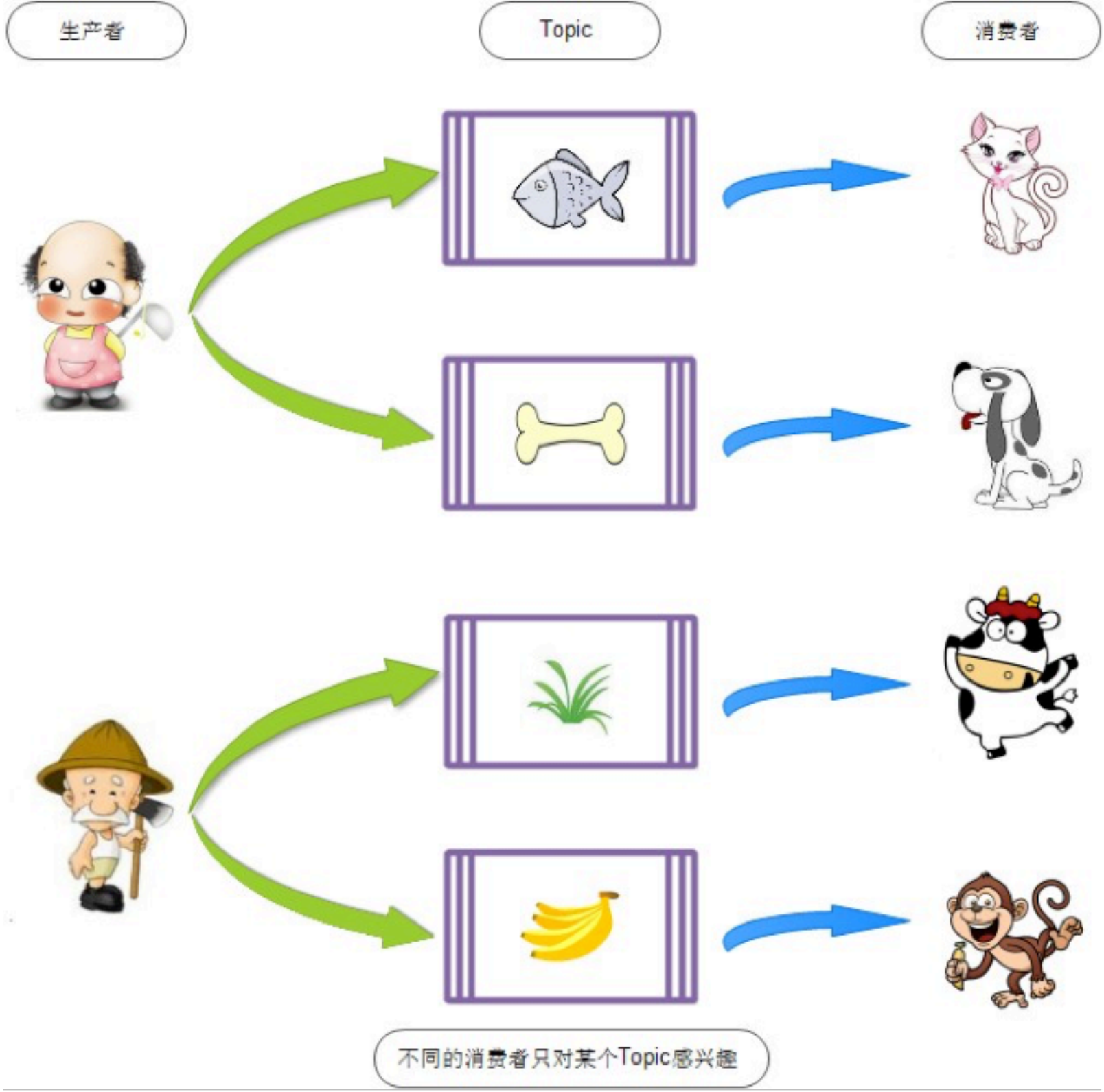
1. 消息的发送和订阅一定是共存的
2. 要支持多个订阅端订阅自己感兴趣的消息

鉴于上一期Kafka和RocketMQ的指标和关注度很高，本期我们将只针对这两个产品，对比在上述场景中，究竟谁更胜一筹。在正式开始测试之前，首先要向大家明确2个概念：

Topic为何物

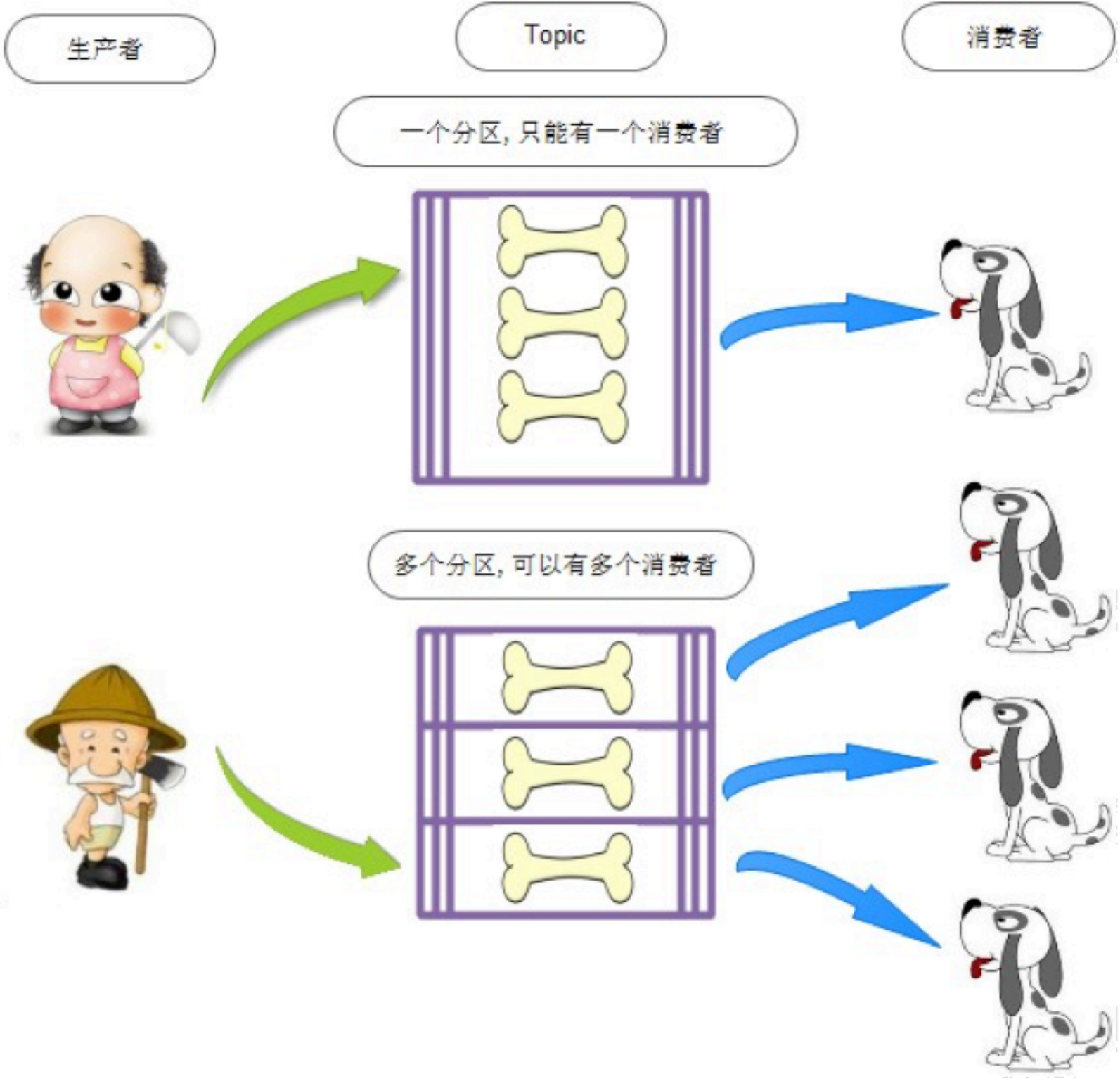
Topic是消息中间件里一个重要的概念，每一个Topic代表了一类消息，有了多个Topic，就可以对消息进行归类与隔离。

可以参照下图的动物园喂食模型，每一种动物都只能消费相对应的食品。



分区为何物

Kafka和RocketMQ都是磁盘消息队列的模式，对于同一个消费组，一个分区只支持一个消费线程来消费消息。过少的分区，会导致消费速度大大落后于消息的生产速度。所以在实际生产环境中，一个Topic会设置成多分区的模式，来支持多个消费者，参照下图：



在互联网企业的实际生产环境中，Topic数量和分区都会比较多，这就要求消息中间件在多Topic共存的时候，依然能够保证服务的稳定性。下面就进入测试环节，看看消息发送端，订阅端共存时，Kafka和RocketMQ对多Topic的处理能力。

测试目的

对比发送端、接收端共存情况下，Topic数量对Kafka、RocketMQ的性能影响，分区数采用8个分区。这次压测我们只关注服务端的性能指标，所以压测的退出标准是:

不断增加发送端的压力,直到系统吞吐量不再上升,而响应时间拉长。此时服务端出现性能瓶颈，获取相应的系统最佳吞吐量，整个过程中保证消息没有累积。

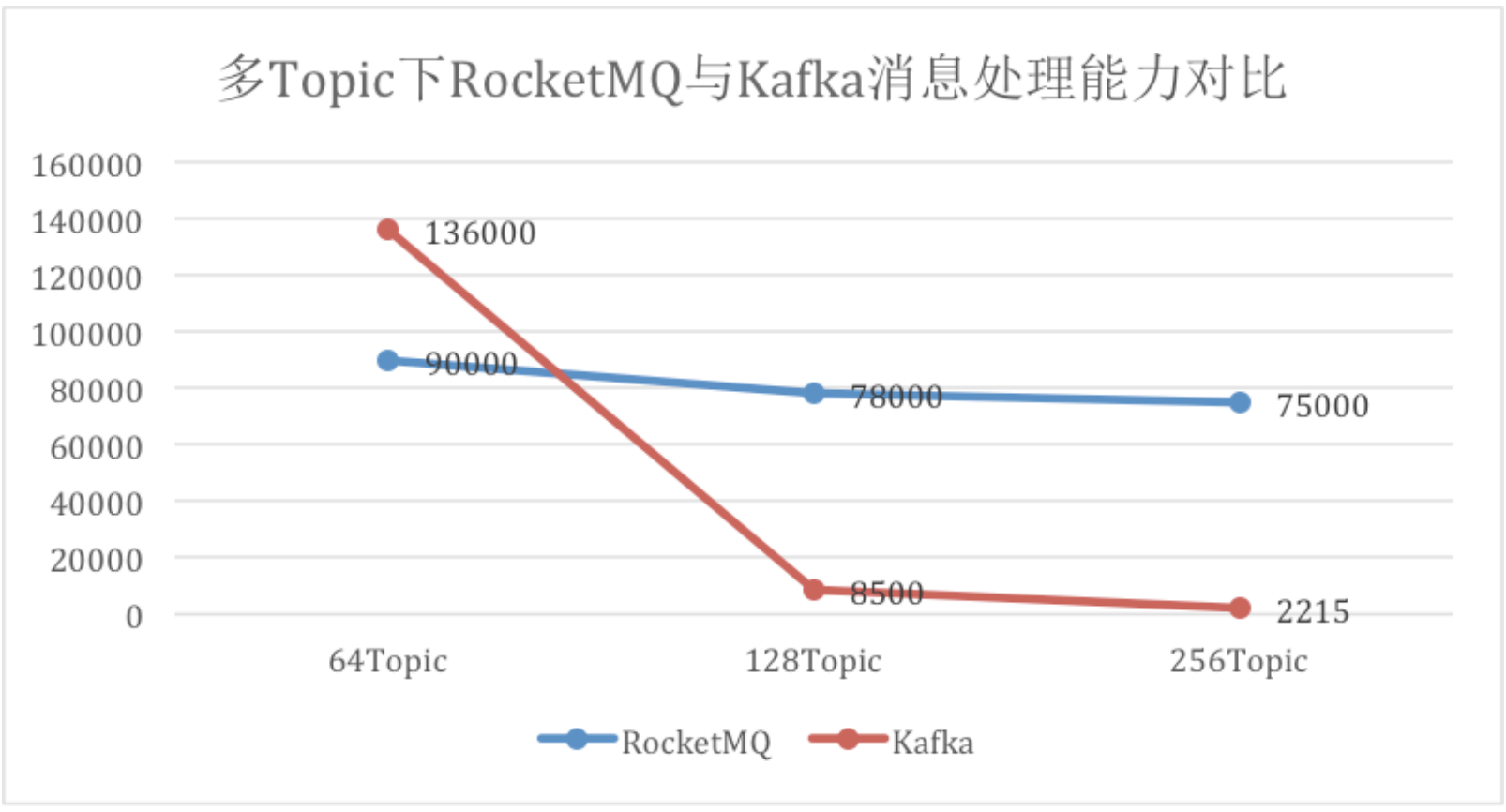
测试场景

默认每个Topic的分区数为8，每个Topic对应一个订阅者，逐步增加Topic数量。得到如下数据：

产品	Topic数量	发送端并发数	发送端RT（ms）	发送端TPS	消费端TPS	
RocketMQ	64	800	8	9w	8.6w	
	128	800	9	7.8w	7.7w	
	256	800	10	7.5w	7.5w	
Kafka	64	800	5	13.6w	13.6w	
	128	256	23	8500	8500	
	256	256	133	2215	2352	

可以看到，不论Topic数量是多少，Kafka和RocketMQ均能保证发送端和消费端的TPS持平，就是说，保证了消息没有累积。

根据Topic数量的变化，画出二者的消息处理能力的对比曲线如下图：



从图上可以看出：

n Kafka在Topic数量由64增长到256时，吞吐量下降了98.37%。

n RocketMQ在Topic数量由64增长到256时，吞吐量只下降了16%。

为什么两个产品的表现如此悬殊呢？这是因为Kafka的每个Topic、每个分区都会对应一个物理文件。当Topic数量增加时，消息分散的落盘策略会导致磁盘IO竞争激烈成为瓶颈。而RocketMQ所有的消息是保存在同一个物理文件中的，Topic和分区数对RocketMQ也只是逻辑概念上的划分，所以Topic数的增加对RocketMQ的性能不会造成太大的影响。

测试结论

在消息发送端，消费端共存的场景下，随着Topic数的增加Kafka吞吐量会急剧下降，而RocketMQ则表现稳定。因此Kafka适合Topic和消费端都比较少的业务场景，而RocketMQ更适合多Topic，多消费端的业务场景。

附录:

测试环境

服务端为单机部署，机器配置如下:

CPU	24核
内存	94G
硬盘	Seagate Constellation ES (SATA 6Gb/s) 2,000,398,934,016 bytes [2.00 TB] 7202 rpm
网卡	1000Mb/s

应用版本:

消息中间件	版本
Kafka	0.8.2
RocketMQ	3.4.6

测试脚本

压力端	Jmeter的java客户端

消息大小	128字节
并发数	能达到服务端最大TPS的最优并发
Topic分区数量	8
刷盘策略	异步落盘

未完待续

经过上面的测试，RocketMQ几乎是完胜Kafka，其实这并不奇怪，因为RocketMQ就是针对互联网的生产要求孕育而生的，读者现在也应该明白为什么RocketMQ可以支撑阿里集团的海量消息业务了吧。

本期测试暂时告一段落了，测试中涉及到的多Topic场景，其实压测时间均只有20分钟，对于一个消息中间件产品来说，过短的执行时间是无法判断它们的稳定性的。下一期我们会继续探索多分区场景下，Kafka和RocketMQ对外服务的稳定性。敬请期待后续的比拼！