达观数据应对大规模消息处理的经验

前言

达观数据作为一家提供大数据处理的公司，经常会遇到客户上报类似于用户点击行为的需求。这样的上报请求不要求马上返回处理结果， 而是需要后台将一系列的上报数据进行统一归档整理挖掘之后将数据呈现给客户。这就需要达观提供一个高可靠，低延时的API接口，而达观后台则需要一个高吞吐，可扩展的消息队列。本文将介绍达观数据在消息队列方面的处理经验， 和大家一起分享。

问题分析：

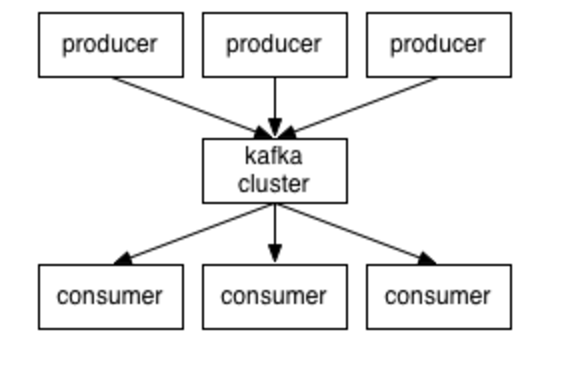
从业务场景看达观后台需要解决以下问题：

1，低延时，高并发： 能及时暂存用户上报数据，并返回暂存结果， 并且能处理大量上报请求；

2，高容错， 持久化： 允许集群中节点失败；后台消费暂存队列数据时能及时回滚到之前消费成功的位置；

3，易扩展： 能在并发量增大的情况下实现热扩展；

4，可监控： 在实际应用中必须可对队列长度进行监控，保证消息被及时处理；

基于以上需求，达观选择了LinkedIn 开源的kafka 消息队列。Kafka现为Apache下的一个子项目，是一个跨语言的高性能pub/sub（发布订阅） 消息队列系统。接下来我将详细介绍kafa以及测试过程。

**Kafka**原理简介：

kafka的pub/sub对象是topic， 一个topic即是一个消息通道(可以理解为一个队列)， 客户上报数据（生产者/producer）时从一个topic写入；后台处理上报数据（消费者/consumer）则从相应的topic中读取数据， 图1，展示了这种生产消费行为。

图1

一个topic包含多个partition（分区），当一条数据（消息）上报过来时， kafka会通过负载均衡不断地被追加到partition 的日志上， 其中每一条消息都被赋予了该partition下的一个唯一的offset值（这个offset值可以达到非常大， 不必考虑offset重复），Kafka允许用户为每个topic设置副本数量，副本数量决定了有几个broker来存放写入的数据。如果你的副本数量设置为3，那么一份数据就会被存放在3台不同的机器上，那么就允许有2个机器失败。一般推荐副本数量至少为2。每一个topic 是一个逻辑上的队列， 每一个partition则为一个物理上的队列。其组织结构如图2。

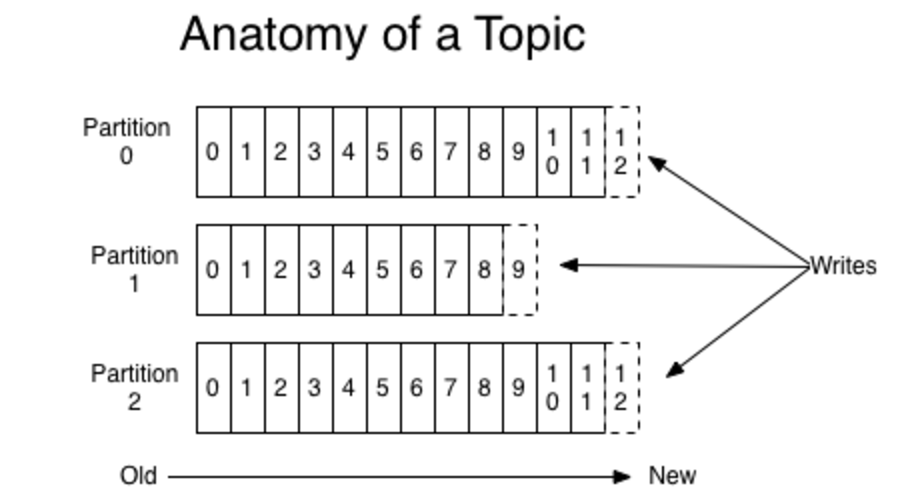
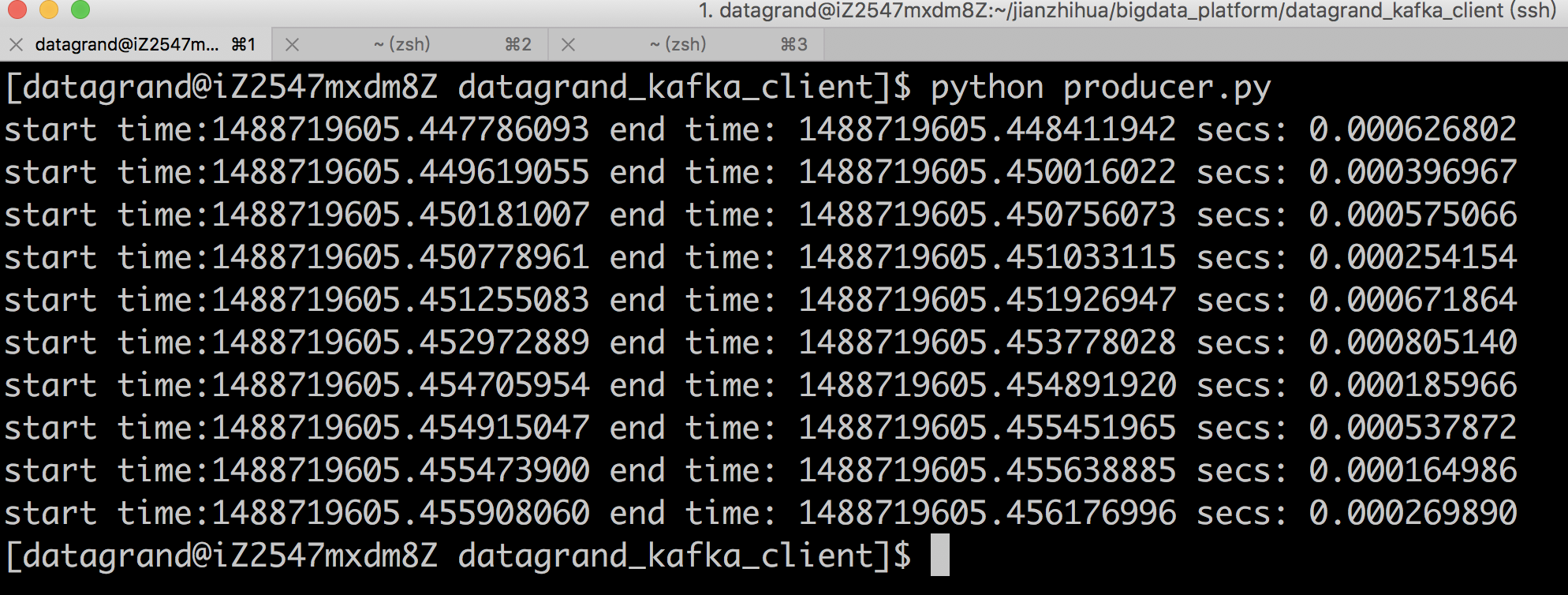


图2

**Kafka**性能测试：

通过以上的描述， 我们已经基本知道kafka消息队列的工作原理了， 那么下面我们将来讨论它的性能到底怎么样呢，是否满足我们上述要求呢？为了简单我们这里采用kafka-Python（https://github.com/dpkp/kafka-python）来实现producer和consumer的功能。

1，低延时：

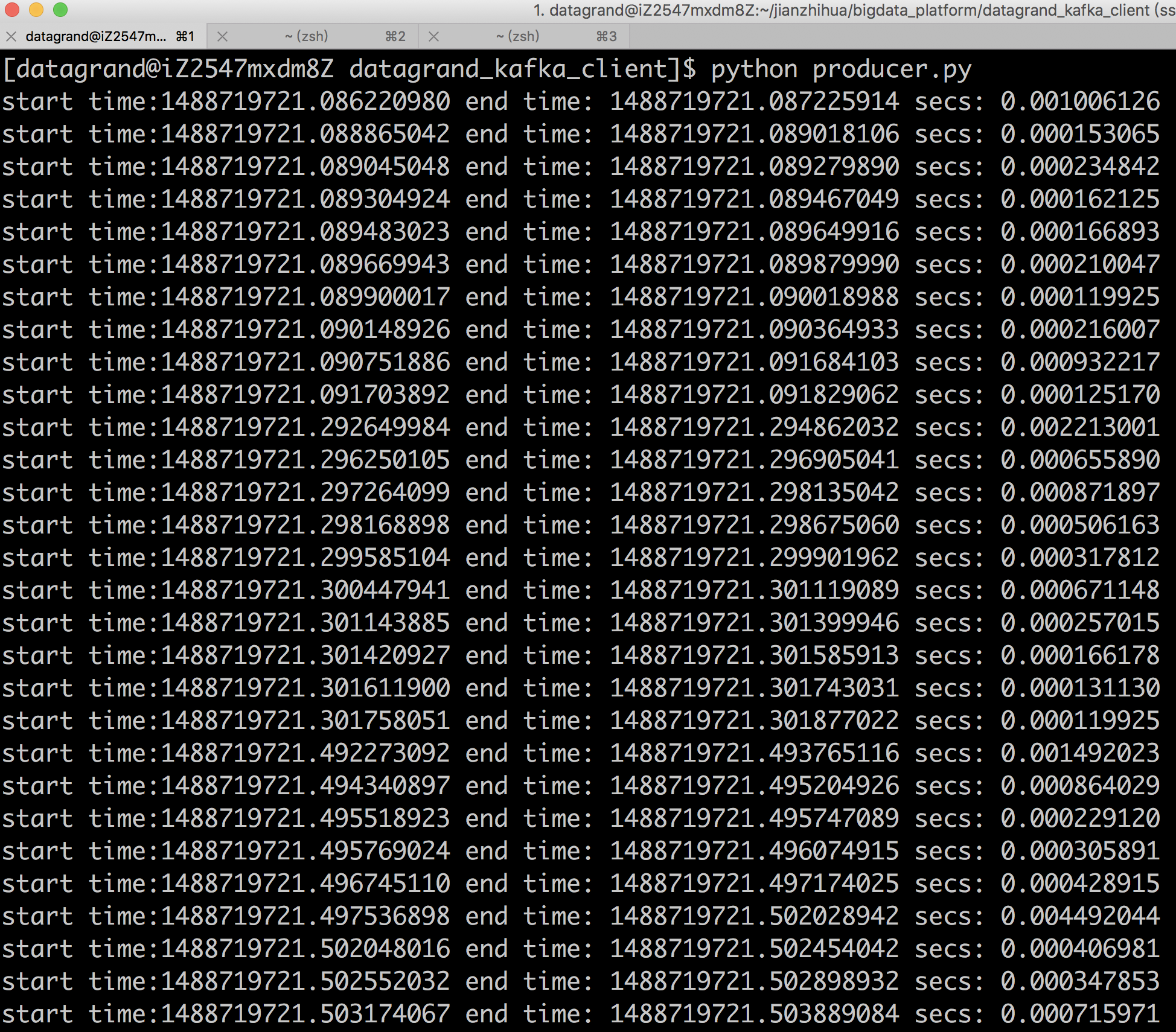


首先我们向一个topic中写入10条消息数据， 看看它的延时情况是怎么样的呢？见图3.

图3

由图3我们可以看到消息发送延时是微妙级别， 统计上面测试情况我们可以得知发送10条消息的平均延时约为0.0005秒， 一个网络请求只需经过0.0005秒的后台逻辑处理就可以返回结果， 无疑这样的结果即使是采用线程池的网络框架也是完全能满足的。

2，高并发：

一个producer 发送数据的情况令我们满意，那么多个进程并发的的效果怎么样呢？我们将同样的逻辑放入到多进程中， 采用10个进程并发的发送数据效果会怎么样呢（部分截图如图4）。 经过测试， 10 个进程每个进程发送10条消息平均延时约0.0006秒。可见kafka在多个producer产生消息的情况依然表现良好， 延时并没有因为producer的数量增长而成倍增加。

多进程发送消息时截图（部分）图4

3， 高容错性：

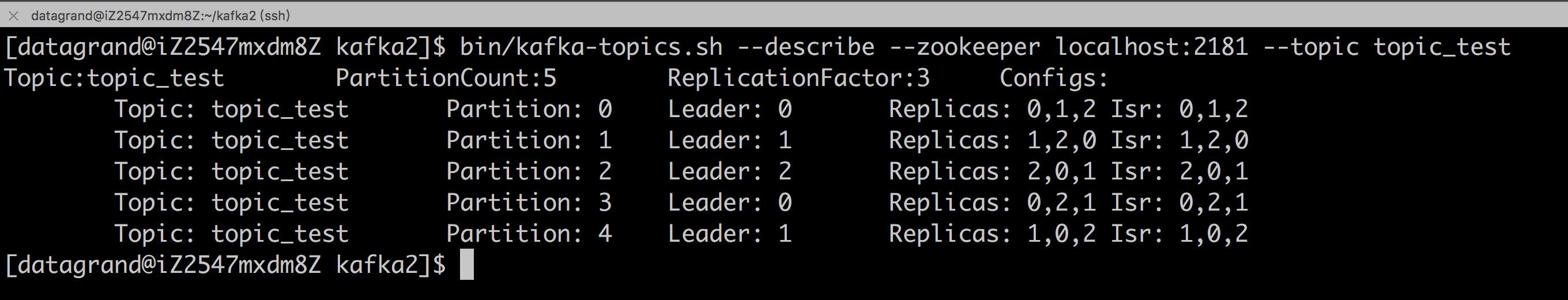
在实际部署kafka集群时我们必须要考虑某些节点失败的情况，这里我们部署了3台kafka的broker（节点）设置的复制因子为2，表示我们允许一台broker失败，下面我们测试一下， 是否一个节点失败了我们还是能正确写入和读出数据。

图5

图5显示了有一个拥有5个partition的topic的详细信息， 现在我们kill掉一个kafka进程，结果和图3一样， 依然可以正常写入。

**4**，持久化：

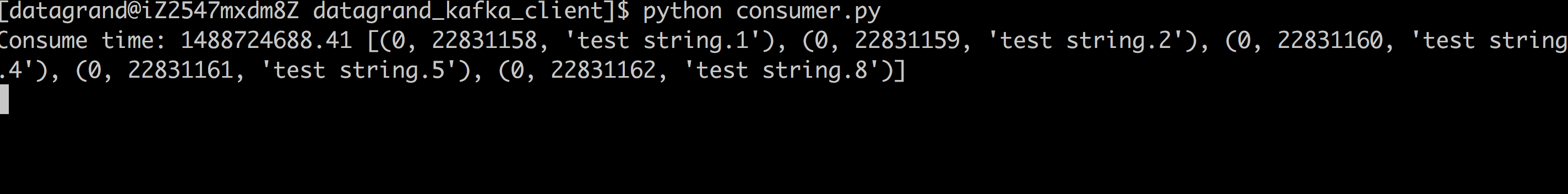
在实际的应用过程中我们发现，消费kafka中的数据时有时候会消费失败， 比如说有些客户上报的信息中将一个字段写错了，导致数据没有正确入库，这时候我们就需要重新消费之前的数据。那这时候我们需要怎么办呢？根据前文我们知道每一条消息存入kafka时都会产生一个该partition下的offset，我们只需要将该partition下的消费提交到这个offset上，就可以重新消费。

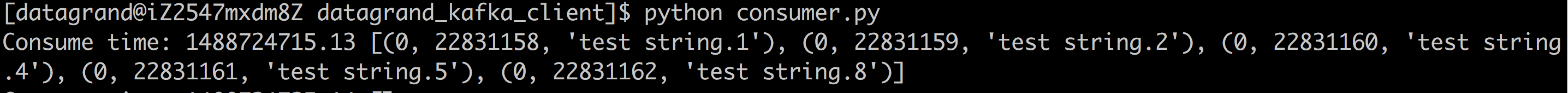
图6

图7

如上图6，图7显示了不同消费时间消费了partition 0 下的相同offset的消息。说明kafka可以完美的解决消费失败重新消费的问题。一般我们会在kafka的配置文件中设置消息的过期时间，超过过期时间的消息将会被kafka自动删除， 所以如果你提交的offset必须要大于被清理的offset。

5， 可扩展：

Kafka可通过集群部署，这对于kafka来说不是什么难事，只需要再添加一个broker，并且把新的broker配置信息逐个加入，重启kafka即可。

6， 可监控：

在实际应用过程中我们需要监控kafka中各个队列的长度， 以确保topic中的消息尽可能快的被消费，也可以防止大量数据涌入造成磁盘紧张的情况。kafka监控可参考KafkaOffsetMonitor（https://github.com/quantifind/KafkaOffsetMonitor）

总结：