讨论课2

讨论参考中间件－－Tube

Tube

Tube－－腾讯实时数据采集系统（TDBank，Tencent Data Bank）的消息存储中心，是TDBank的核心组件，但Tube可作为单独的消息中间件被使用。

Tube由腾讯数据平台部自行研发，其系统架构思想源于Apache Kafka，在实现上，则完全采取了自己的方式。

功能与特性

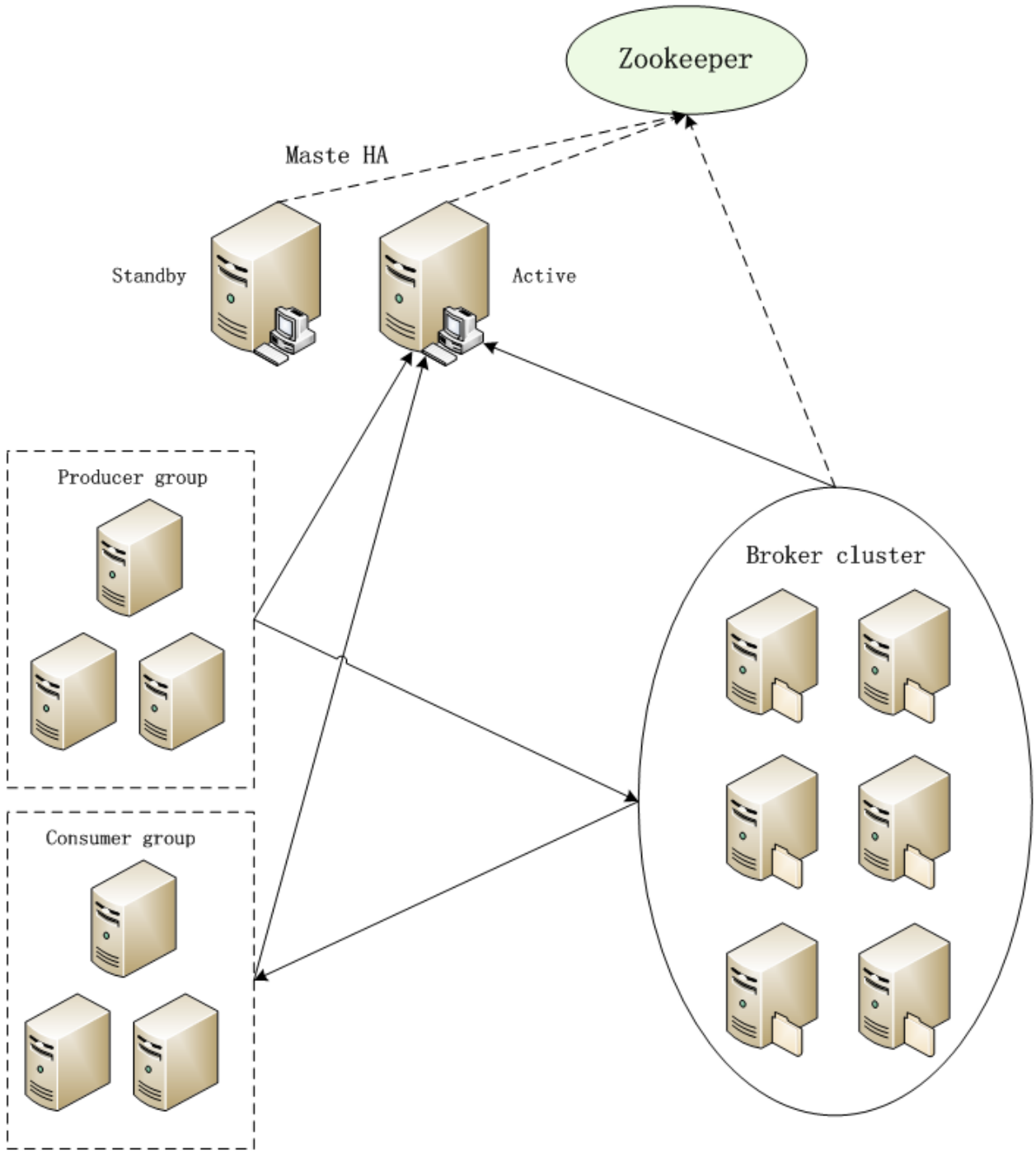
Tube实现JMS规范中的发布/订阅模型，且发布端、订阅端的可使用集群模式，同时支持负载均衡发布端和消费端；消费端以Group分组，同一分组下的消费者消费同一份数据，且可以重复消费多次；不同分组之间相互独立、互不影响。通常情况下，Tube可以做到消息的“零误差”，即不丢失、不重复、不损坏；极端场景下，也仅会丢失或重复少量的消息。

架构设计

Tube系统主要有三部分组成：客户端（producer&consumer）、服务端（broker）和中心节点；其中，客户端包括Producer和Consumer API，主要用来生产和消费消息数据；服务端则是用来接受Topic的发布和订阅，并存储消息数据；而中心节点主要为Master节点，用来收集和监控整个集群的状态。

另外，目前在生产环境部署的Tube集群还使用了Zookeeper，目前主要用来保存Consumer的消费位置和Master HA的选举，不过Zookeeper的存在主要是历史遗留，全新的Tube系统设计是可以摆脱对Zookeeper依赖的。

当前Tube系统的整体架构如下图所示：



其中，Broker负责数据的存储，为集群部署模式，Producer为生产者，它将消息发送到broker集群，Consumer为消费者，从Broker读取消息到自己的本地处理；Producer和Consumer可以是单机模式，也可以是集群模式；Master为中控节点，负责收集整个集群的状态信息，并控制消费者的分区消费平衡。

Tube采用了Kafka的分区设计思想，其各个节点主要交互流程如下： 1. 消息按照内容分为多类，每一类消息有一个Topic，每个Topic可以包含多个分区，分布在若干Broker上；Producer可以生产某一个或多个Topic数据，此后Consumer可以指定这些Topic的数据进行消费； 2. Broker向Master汇报自身信息，包括自身id、状态以及提供哪些Topic的发布和订阅服务，每个Topic下包含多少分区； 3. Producer向Master通报要发布的Topic名称，Master给Producer返回哪些Broker可以接收这些Topic的数据；此后Producer可直接向这些Broker生产数据； 4. Consumer向Master通报要订阅的Topic名称，Master给Consumer返回可以从哪些

Broker获取数据；此后Consumer则直接向这些Broker拉取数据；不同业务Consumer通过Group来区分，同一个Group下的Consumer消费同一份数据，每个Consumer会负责消费其中的一部分分区，Consumer之间消费的分区互不重叠，Consumer和分区的消费对应关系由Master统一分配；不同Group下的Consumer消费互不影响，既可以消费相同Topic的数据，也可以消费不同Topic的数据； 5. 集群节点间通过心跳和Master保持状态同步，当状态发生变化时，Master会生成相应时间并负责通知相关节点； 6. 为了避免中心节点的单点存在，Master采用主备模式，并通过Zookeeper来进行选举。

消息模型

Tube系统的实现使用的是发布/订阅（Publish/Subscribe，简称P/S）模型

另外由于体量的原因，这里的信息传递即会存在一个Push和Pull的选择问题。所谓Push模型，即是当Producer发出的消息到达后，服务端马上将这条消息投递给Consumer；而Pull则是服务端收到这条消息后什么也不做，只是等着Consumer主动到自己这里来读，即Consumer这里有一个“拉取”的动作。选择推（Push）方式还是拉（Pull）方式直接关乎到整个系统的使用场景，吞吐量、实时性以及性能问题

而Tube旨在服务于大数据平台，需要进行海量数据分发的消息系统，Pull方式无疑是更好的选择，牺牲一定实时性来获得更高的吞吐和更强的消息堆积能力。

消息一致性

在消息的一致性模型中，主要包含三个方面的内容，分别是消息的可靠性，消息的顺序和消息重复，下面分别讨论。

1. 消息的可靠性 在互联网大数据场景下，由于体量非常巨大，而数据的使用又偏向统计和分析，所以其对数据的可靠性要求并不像交易、金融等业务系统那么高，可以在一定程度上容忍少量的数据错误或丢失，而这一点也是Tube实现高吞吐量的一个前提。 目前Tube系统主要在两个地方可能会有数据丢失，第一是Tube采取了Consumer信任模型，即数据一旦被Consumer拉取到本地，就默认会消费成功，如果Consumer在实际消费的过程中出现错误（Tube可以保证消息数据的正确性，所以这种错误一般是由业务方消费逻辑或代码问题导致），则Tube并不负责恢复，所以这里存在一个丢失数据的风险；再一个就是由于操作系统Pagecache的利用，服务器断电或宕机而可能带来的数据丢失。
2. 顺序的消息性 由于Tube沿用了Kafka的分区设计思想，而分区的数据消费之间是没有先后顺序关系的，而且Tube支持消息的异步方式发送，在这种方式下，网络并不能保证先发送的消息就一定会先到达服务端，所以Tube一般不提供顺序性的保证； 目前系统结构下，只有使用单一分区并且采用同步发送接口的情况下，才可以实现消息的顺序写入和读取；但是这样的使用方式，其性能和吞吐则会大打折扣，完全背离了Tube服务于海量消息传输的初衷。
3. 消息的重复 在Tube集群中，Consumer的消费位置信息由Broker端进行管理，所以在某些异常情况下，Broker可能无准确获得Consumer的实际消费情况而导致数据重复；另外就是出于性能考虑， Consumer的消费位置信息在每次变化时，并不会实时更新到持久化存储中，而是暂存于内存，周期性更新，如果此时broker宕机即会导致少量的数据重复。所以在使用Tube时，业务本身对数据的处理要具有幂等性或者是能够容忍少量的数据重复；而在大数据的使用场景中，后者一般不是问题。

存储模型

存储的选择关系着整个系统所能提供的功能和性能。Tube在存储层上实现了接口化设计，具备根据实际情况更换存储引擎的能力，从而可以利用不同存储引擎的特性，来为消息系统增加更多丰富的功能。

通信框架

无论是消息系统还是其他的分布式系统，其高效的数据通信都要依赖于底层的通讯框架，一个是高效的RPC框架更是必不可少的。而高性能的RPC框架设计，主要体现在两个方面：高效的I/O模型和优秀的序列化/反序列化框架。

Tube的RPC实现便采用Netty作为其I/O通信模型部分，Netty是基于Java NIO实现的一个经典Reactor模型的框架，其性能和稳定性都得到了充分地验证，也是当前Java领域很流行的一款网络开发框架。其Boss/Worker的线程模型可以实现对大量连接请求和数据请求的高效管理。不过在Tube RPC的接口设计中，完全是抽象化的通用的，并不会和某种特定的通信框架绑定，完全可以替换成其他的诸如Mina或Grizzly等NIO框架，以充分利用他们各自的特性和优势。

随着硬件的不断换代升级，在普通的PC Server中，速度的瓶颈其实往往是在网卡，所以只能在数据压缩上下功夫了。这里的数据压缩包含两个部分，一个是对象的序列化，另一个则是真正含义的数据压缩。 Tube在开发时使用了Google的Protobuf作为序列化框架，主要是看中它在资源消耗、压缩比上的综合优势，在易用性和稳定性方面经受了考验，而且有丰富的文档支持。