# 讨论课2

> 2016.4.23
>
> Created by Fowafolo

## 讨论内容

> 参考业界架构，讨论程序的扩展
>
> 1. 分布式：高可用性、高吞吐量
> 2. 数据存储，分区，一致性，缓存
> 3. 负载均衡
> 4. ID分配
> 5. 通信可靠高效
> 6. 协议
> 7. 消息队列
> 8. 垃圾消息过滤
> 9. 安全

基于上述要求，我进行了下面的学习。

## 学习过程

1. 查询业界目前比较流行的分布式消息系统架构：Kafka
2. 了解Kafka的架构
3. Kafka吞吐量、负载均衡的实现方案
4. Kafka可靠性的方案
5. Kafka的数据持久化方案
6. 常见消息系统的安全方案
7. 常见消息系统的垃圾过滤机制
8. 与其他消息队列对比

# 详细资料

## 查询业界目前比较流行的分布式消息系统架构：Kafka

Kafka是一个消息系统，原本开发自LinkedIn，用作LinkedIn的活动流（Activity Stream）和运营数据处理管道（Pipeline）的基础。现在它已被多家不同类型的公司 作为多种类型的数据管道和消息系统使用。

Kafka是一种分布式的，基于发布/订阅的消息系统。主要设计目标如下：

以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上数据也能保证常数时间复杂度的访问性能。 高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒100K条以上消息的传输。 支持Kafka Server间的消息分区，及分布式消费，同时保证每个Partition内的消息顺序传输。 同时支持离线数据处理和实时数据处理。 Scale out：支持在线水平扩展。

## 了解Kafka的架构

#### Broker

Kafka集群包含一个或多个服务器，这种服务器被称为broker

#### Topic

每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）

#### Partition

Parition是物理上的概念，每个Topic包含一个或多个Partition.

#### Producer

负责发布消息到Kafka broker

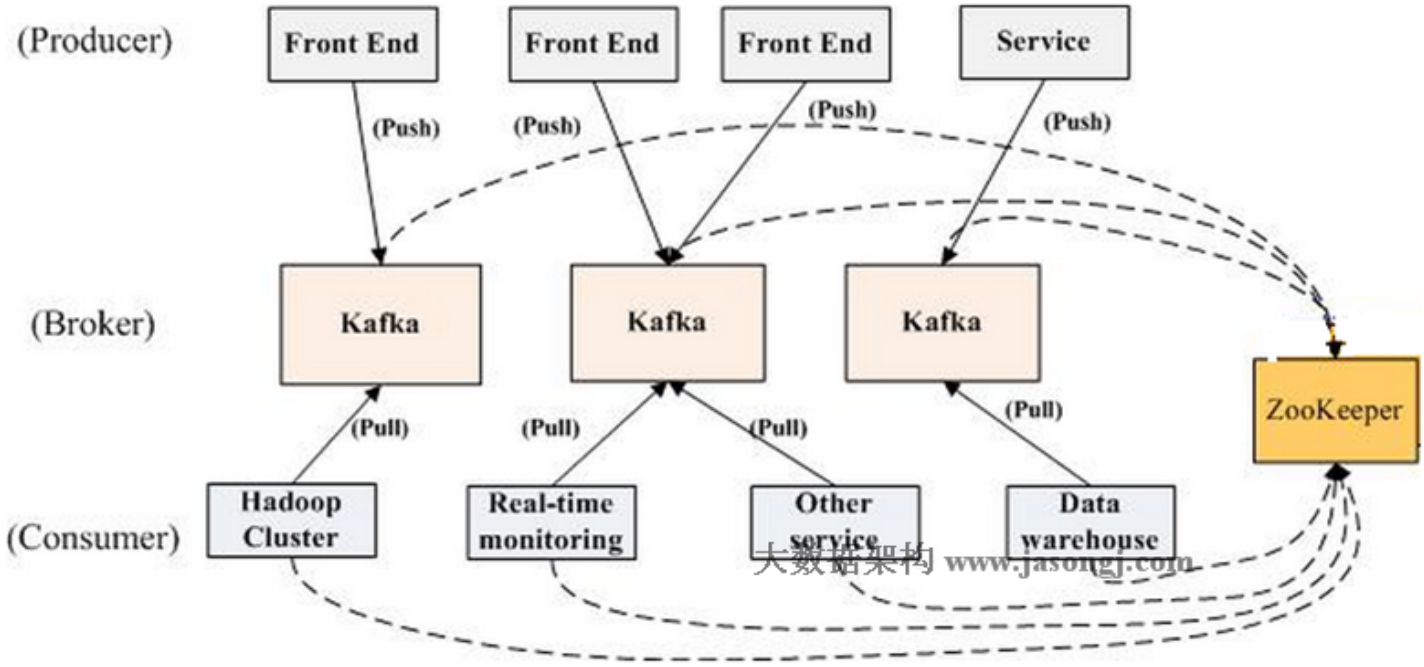
#### Consumer

消息消费者，向Kafka broker读取消息的客户端。

#### Consumer Group

每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）。

#### 架构图




如上图所示，一个典型的kafka集群中包含若干producer（可以是web前端产生的page view，或者是服务器日志，系统CPU、memory等），若干broker（Kafka支持水平扩展，一般broker数量越多，集群吞吐率越高），若干consumer group，以及一个Zookeeper集群。Kafka通过Zookeeper管理集群配置，选举leader，以及在consumer group发生变化时进行rebalance。producer使用push模式将消息发布到broker，consumer使用pull模式从broker订阅并消费消息。

> Topic & Partition
>
> Topic在逻辑上可以被认为是一个queue。每条消费都必须指定它的topic，可以简单理解为必须指明把这条消息放进哪个queue里。为了使得Kafka的吞吐率可以水平扩展，物理上把topic分成一个或多个partition，每个partition在物理上对应一个文件夹，该文件夹下存储这个partition的所有消息和索引文件。

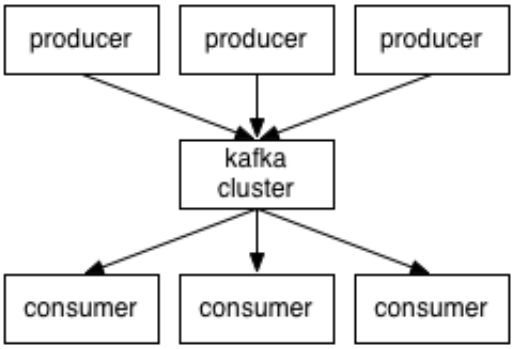
## Kafka吞吐量、负载均衡的实现方案

因为每条消息都被append到该partition中，是顺序写磁盘，因此效率非常高（经验证，顺序写磁盘效率比随机写内存还要高，这是Kafka高吞吐率的一个很重要的保证）。

Producer发送消息到broker时，会根据Paritition机制选择将其存储到哪一个Partition。如果Partition机制设置合理，所有消息可以均匀分布到不同的Partition里，这样就实现了负载均衡。如果一个Topic对应一个文件，那这个文件所在的机器I/O将会成为这个Topic的性能瓶颈，而有了Partition后，不同的消息可以并行写入不同broker的不同Partition里，极大的提高了吞吐率。

## Kafka可靠性的方案

虽然Kafka牺牲了一些可靠性来提升吞吐量，有人自然会担心消息的丢失，但是Kafka对消息可靠性也有自己的解决方案。




先来看消息投递可靠性，一个消息如何算投递成功，Kafka提供了三种模式，第一种是啥都不管，发送出去就当作成功，这种情况当然不能保证消息成功投递到broker；第二种是对于Master Slave模型，只有当Master和所有Slave都接收到消息时，才算投递成功，这种模型提供了最高的投递可靠性，但是损伤了性能；第三种模型，即只要Master确认收到消息就算投递成功；实际使用时，根据应用特性选择，绝大多数情况下都会中和可靠性和性能选择第三种模型。

我们再来看消息在broker上的可靠性，因为消息会持久化到磁盘上，所以如果正常stop一个broker，其上的数据不会丢失；但是如果不正常stop，可能会使存在页面缓存来不及写入磁盘的消息丢失，这可以通过配置flush页面缓存的周期、阈值缓解，但是同样会频繁的写磁盘会影响性能，又是一个选择题，根据实际情况配置。

接着，我们再看消息消费的可靠性，Kafka提供的是“At least once”模型，因为消息的读取进度由offset提供，offset可以由消费者自己维护也可以维护在zookeeper里，但是当消息消费后consumer挂掉，offset没有即时写回，就有可能发生重复读的情况，这种情况同样可以通过调整commit offset周期、阈值缓解，甚至消费者自己把消费和commit offset做成一个事务解决，但是如果你的应用不在乎重复消费，那就干脆不要解决，以换取最大的性能。

## Kafka的数据持久化方案

Kafka通过自身独特的设计将消息持久化到磁盘上，以此同时支持在线和离线消费。

很多系统、组件为了提升效率一般恨不得把所有数据都扔到内存里，然后定期flush到磁盘上；可实际上，现代操作系统也是这样，所有的现代操作系统都乐于将空闲内存转作磁盘缓存（页面缓存），想不用都难；对于这样的系统，他的数据在内存中保存了一份，同时也在OS的页面缓存中保存了一份，这样不但多了一个步骤还让内存的使用率下降了一半；因此，Kafka决定直接使用页面缓存；但是随机写入的效率很慢，为了维护彼此的关系顺序还需要额外的操作和存储，而线性的写入可以避免这些，实际上，线性写入（linear write）的速度大约是300MB/秒，但随即写入却只有50k/秒，其中的差别接近10000倍。这样，Kafka以页面缓存为中间的设计在保证效率的同时还提供了消息的持久化，每个消费者自己维护当前读取数据的offser（也可委托给zookeeper），以此可同时支持在线和离线的消费。

## 常见消息系统的安全方案

这一方面我没有查到概述的安全方案，根据以往的项目，大概查了以下几个方面：

**信息加密**

这一块内容的解决方案有很多，业界也存在众多加密解密算法。比较常用的是MD5、TEA（如QQ）、RSA等。但是有的时候经过加密的信息传递起来，时间成本高于信息的本身价值，所以据说腾讯在使用TEA算法进行加密的时候没有采用国际通用的16轮加密，而是采用了8轮。

个人认为对信息的加密要分不同的情况，如果是极为隐私的数据，如银行卡账户、密码等等的传输，一定要高于通常标准，而对一些价值不高的数据，就要在安全性和效率上进行取舍了。

**可靠性**

详见上方对可靠性的探讨。

**备份**

即常见的问题，如果服务器崩了或者被黑了，或者出现各种各样的物理错误或者逻辑错误，保存的信息丢了怎么办？业界通用的防灾方案是进行备份，即定期将数据进行复制，转移到用于备份的机器。

## 常见消息系统的垃圾过滤机制

随着网络的发展,人们传递信息的手段日新月异,电子邮件、手机短信、QQ 聊天等信息传递手段以其方便快捷等优点,越来越成为人们信息传递的主要手段.但是垃圾邮件、垃圾短信、广告、色情等等不良信息充斥在我们周围,这不仅给我们带来了很多的不便,同时也浪费了大量的公共资源,侵害了用户合法权益,甚至有时会造成巨大的经济损失,因此垃圾信息的过滤问题越来越成为用户关注的主要问题之一[1011].

垃圾信息的过滤一般要经过以下步骤:

1. 步骤1 首先采集一定数量的信息,建立相应的垃圾信息集和合法信息集.
2. 步骤2 信息要进行预处理,信息预处理的过程也就是对信息进行空间向量化的过程.
3. 步骤3 利用相应的分类算法对已知的垃圾信息样本集进行训练,统计相应数据,获得相应参数和阈值,构建分类器.
4. 步骤4 利用获得的分类器,对未知信息进行过滤.

这里我查阅了一篇基于SVM的垃圾信息过滤的实现方案：

基于支持向量机的垃圾信息过滤方法

## 与其他消息队列对比

**RabbitMQ**

RabbitMQ是使用Erlang编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了Broker构架，这意味着消息在发

送给客户端时先在中心队列排队。对路由，负载均衡或者数据持久化都有很好的支持。

**Redis**

Redis是一个基于Key-Value对的NoSQL数据库，开发维护很活跃。虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能，所以完全可以当做一个轻量级的队列服务来使用。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。

**ActiveMQ**

ActiveMQ是Apache下的一个子项目。 类似于ZeroMQ，它能够以代理人和点对点的技术实现队列。同时类似于RabbitMQ，它少量代码就可以高效地实现高级应用场景。

**Kafka/Jafka**

Kafka是Apache下的一个子项目，是一个高性能跨语言分布式发布/订阅消息队列系统，而Jafka是在Kafka之上孵化而来的，即Kafka的一个升级版。具有以下特性：快速持久化，可以在O(1)的系统开销下进行消息持久化；高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统，Broker、Producer、Consumer都原生自动支持分布式，自动实现负载均衡；支持Hadoop数据并行加载，对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka通过Hadoop的并行加载机制统一了在线和离线的消息处理。Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。

# 参考资料

| 参考资料名称 | 链接地址 |
| --- | --- |
| 基于支持向量机的垃圾信息过滤方法 | http://journal.bit.edu.cn/zr/ch/reader/create*pdf.aspx?file*no=20131013 |
| Kafka剖析 | http://www.infoq.com/cn/articles/kafka-analysis-part-1 |
| Kafka深度解析 | http://www.jasongj.com/2015/01/02/Kafka%E6%B7%B1%E5%BA%A6%E8%A7%A3%E6%9E%90/ |
| 资料 | http://www.cnblogs.com/bbgasj/p/4176915.html |
| 一种高效垃圾短信过滤系统的实现 | http://wenku.baidu.com/view/61b87fef5ef7ba0d4a733b2e.html |