**[rabbitMQ、activeMQ、zeroMQ、Kafka、Redis 比较](http://www.cnblogs.com/valor-xh/p/6348009.html)**

Kafka作为时下最流行的开源消息系统，被广泛地应用在数据缓冲、异步通信、汇集日志、系统解耦等方面。相比较于RocketMQ等其他常见消息系统，Kafka在保障了大部分功能特性的同时，还提供了超一流的读写性能。

针对Kafka性能方面进行简单分析，相关数据请参考：[https://segmentfault.com/a/1190000003985468](https://segmentfault.com/a/1190000003985468" \t "http://www.cnblogs.com/valor-xh/p/_blank)，下面介绍一下Kafka的架构和涉及到的名词：

Topic：用于划分Message的逻辑概念，一个Topic可以分布在多个Broker上。

Partition：是Kafka中横向扩展和一切并行化的基础，每个Topic都至少被切分为1个Partition。

Offset：消息在Partition中的编号，编号顺序不跨Partition。

Consumer：用于从Broker中取出/消费Message。

Producer：用于往Broker中发送/生产Message。

Replication：Kafka支持以Partition为单位对Message进行冗余备份，每个Partition都可以配置至少1个Replication(当仅1个Replication时即仅该Partition本身)。

Leader：每个Replication集合中的Partition都会选出一个唯一的Leader，所有的读写请求都由Leader处理。其他Replicas从Leader处把数据更新同步到本地，过程类似大家熟悉的MySQL中的Binlog同步。

Broker：Kafka中使用Broker来接受Producer和Consumer的请求，并把Message持久化到本地磁盘。每个Cluster当中会选举出一个Broker来担任Controller，负责处理Partition的Leader选举，协调Partition迁移等工作。

ISR(In-Sync Replica)：是Replicas的一个子集，表示目前Alive且与Leader能够“Catch-up”的Replicas集合。由于读写都是首先落到Leader上，所以一般来说通过同步机制从Leader上拉取数据的Replica都会和Leader有一些延迟(包括了延迟时间和延迟条数两个维度)，任意一个超过阈值都会把该Replica踢出ISR。每个Partition都有它自己独立的ISR。

更多关于Kafka的数据，参考：[https://segmentfault.com/a/1190000003985468](https://segmentfault.com/a/1190000003985468" \t "http://www.cnblogs.com/valor-xh/p/_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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。

**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。

**ZeroMQ**  
ZeroMQ号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZeroMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。ZeroMQ具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务器或中间件，因为你的应用程序将扮演这个服务器角色。你只需要简单的引用ZeroMQ程序库，可以使用NuGet安装，然后你就可以愉快的在应用程序之间发送消息了。但是ZeroMQ仅提供非持久性的队列，也就是说如果宕机，数据将会丢失。其中，Twitter的Storm 0.9.0以前的版本中默认使用ZeroMQ作为数据流的传输（Storm从0.9版本开始同时支持ZeroMQ和Netty作为传输模块）。

**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。

以上转自：[http://www.infoq.com/cn/articles/kafka-analysis-part-1/](http://www.cnblogs.com/ %E8%BD%AC%E8%87%AA%EF%BC%9Ahttp:/www.infoq.com/cn/articles/kafka-analysis-part-1/" \t "http://www.cnblogs.com/valor-xh/p/_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## **什么是Kafka？**

引用官方原文：  “ Kafka is a distributed, partitioned, replicated commit log service. ”

它提供了一个非常特殊的消息机制，不同于传统的mq。

官网：https://kafka.apache.org

## **它与传统的mq区别？**

* 更快！单机上万TPS
* 传统的MQ，消息被消化掉后会被mq删除，而kafka中消息被消化后不会被删除，而是到配置的expire时间后，才删除
* 传统的MQ，消息的Offset是由MQ维护，而kafka中消息的Offset是由客户端自己维护
* 分布式，把写入压力均摊到各个节点。可以通过增加节点降低压力

## **基本术语**

为方便理解，我用对比传统MQ的方式阐述这些基本术语。

Producer

Consumer

这两个与传统的MQ一样，不解释了

Topic

Kafka中的topic其实对应传统MQ的channel，即消息管道，例如同一业务用同一根管道

Broker

集群中的KafkaServer，用来提供Partition服务

Partition

假如说传统的MQ，传输消息的通道(channel)是一条双车道公路，那么Kafka中，Topic就是一个N车道的高速公路。每个车道都可以行车，而每个车道就是Partition。

* 一个Topic中可以有一个或多个partition。
* 一个Broker上可以跑一个或多个Partition。集群中尽量保证partition的均匀分布，例如定义了一个有3个partition的topic，而只有两个broker，那么一个broker上跑两个partition，而另一个是1个。但是如果有3个broker，必然是3个broker上各跑一个partition。
* Partition中严格按照消息进入的顺序排序
* 一个从Producer发送来的消息，只会进入Topic的某一个Partition（除非特殊实现Producer要求消息进入所有Partition）
* Consumer可以自己决定从哪个Partition读取数据

Offset

单个Partition中的消息的顺序ID，例如第一个进入的Offset为0，第二个为1，以此类推。传统的MQ，Offset是由MQ自己维护，而kafka是由client维护

Replica

Kafka从0.8版本开始，支持消息的HA，通过消息复制的方式。在创建时，我们可以指定一个topic有几个partition，以及每个partition有几个复制。复制的过程有同步和异步两种，根据性能需要选取。 正常情况下，写和读都是访问leader，只有当leader挂掉或者手动要求重新选举，kafka会从几个复制中选举新的leader。

Kafka会统计replica与leader的同步情况。当一个replica与leader数据相差不大，会被认为是一个"in-sync" replica。只有"in-sync" replica才有资格参与重新选举。

ConsumerGroup

一个或多个Consumer构成一个ConsumerGroup，一个消息应该只能被同一个ConsumerGroup中的一个Consumer消化掉，但是可以同时发送到不同ConsumerGroup。

通常的做法，一个Consumer去对应一个Partition。

传统MQ中有queuing（消息）和publish-subscribe（订阅）模式，Kafka中也支持：

* 当所有Consumer具有相同的ConsumerGroup时，该ConsumerGroup中只有一个Consumer能收到消息，就是 queuing 模式
* 当所有Consumer具有不同的ConsumerGroup时，每个ConsumerGroup会收到相同的消息，就是 publish-subscribe 模式

## **基本交互原理**

每个Topic被创建后，在zookeeper上存放有其metadata，包含其分区信息、replica信息、LogAndOffset等   
默认路径/brokers/topics/<topic\_id>/partitions/<partition\_index>/state

Producer可以通过zookeeper获得topic的broker信息，从而得知需要往哪写数据。

Consumer也从zookeeper上获得该信息，从而得知要监听哪个partition。

## **基本CLI操作**

1. 创建Topic

./kafka-create-topic.sh --zookeeper 10.1.110.21:2181 --replica 2 --partition 3 --topic test

2. 查看Topic信息

./kafka-list-topic.sh --topic test --zookeeper 10.1.110.24:2181

3. 增加Partition

./kafka-add-partitions.sh --partition 4 --topic test --zookeeper 10.1.110.24:2181

更多命令参见：[https://cwiki.apache.org/confluence/display/KAFKA/Replication+tools](https://cwiki.apache.org/confluence/display/KAFKA/Replication+tools" \t "http://www.cnblogs.com/valor-xh/p/_blank)

## **创建一个Producer**

Kafka提供了java api，Producer特别的简单，举传输byte[] 为例

Properties p = new Properties();

props.put("metadata.broker.list", "10.1.110.21:9092");

ProducerConfig config = new ProducerConfig(props);

Producer producer = new Producer<String, byte[]>(config);

producer.send(byte[] msg);

更具体的参见:[https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example](https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example" \t "http://www.cnblogs.com/valor-xh/p/_blank)

## **创建一个Consumer**

Kafka提供了两种java的Consumer API：High Level Consumer和Simple Consumer

看上去前者似乎要更牛B一点，事实上，前者做了更多的封装，比后者要Simple的多……

具体例子我就不写了，参见

High Level Consumer: [https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example](https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example" \t "http://www.cnblogs.com/valor-xh/p/_blank)

Simple Consumer: [https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example](https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example" \t "http://www.cnblogs.com/valor-xh/p/_blank)

 摘自：[http://www.tuicool.com/articles/ruUzum](http://www.tuicool.com/articles/ruUzum" \t "http://www.cnblogs.com/valor-xh/p/_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

如何保证kafka的高容错性？

1. producer不使用批量接口，并采用同步模型持久化消息。
2. consumer不采用批量化，每消费一次就更新offset

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ActiveMq** | **RabbitMq** | **Kafka** |
| **producer容错，是否会丢数据** |  | 有ack模型，也有事务模型，保证至少不会丢数据。ack模型可能会有重复消息，事务模型则保证完全一致 | 批量形式下，可能会丢数据。 非批量形式下， 1. 使用同步模式，可能会有重复数据。 2. 异步模式，则可能会丢数据。 |
| **consumer容错，是否会丢数据** |  | 有ack模型，数据不会丢，但可能会重复处理数据。 | 批量形式下，可能会丢数据。非批量形式下，可能会重复处理数据。（ZK写offset是异步的） |
| **架构模型** | 基于JMS协议 | 基于AMQP模型，比较成熟，但更新超慢。RabbitMQ的broker由Exchange,Binding,queue组成，其中exchange和binding组成了消息的路由键；客户端Producer通过连接channel和server进行通信，Consumer从queue获取消息进行消费（长连接，queue有消息会推送到consumer端，consumer循环从输入流读取数据）。rabbitMQ以broker为中心；有消息的确认机制 | producer，broker，consumer，以consumer为中心，消息的消费信息保存的客户端consumer上，consumer根据消费的点，从broker上批量pull数据；无消息确认机制。 |
| **吞吐量** |  | rabbitMQ在吞吐量方面稍逊于kafka，他们的出发点不一样，rabbitMQ支持对消息的可靠的传递，支持事务，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。 | kafka具有高的吞吐量，内部采用消息的批量处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高 |
| **可用性** |  | rabbitMQ支持miror的queue，主queue失效，miror queue接管 | kafka的broker支持主备模式 |
| **集群负载均衡** |  | rabbitMQ的负载均衡需要单独的loadbalancer进行支持 | kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上 |

参考：[http://www.liaoqiqi.com/post/227](http://www.liaoqiqi.com/post/227" \t "http://www.cnblogs.com/valor-xh/p/_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

注：下文转载自：[http://blog.csdn.net/linsongbin1/article/details/47781187](http://blog.csdn.net/linsongbin1/article/details/47781187" \t "http://www.cnblogs.com/valor-xh/p/_blank)

MQ框架非常之多，比较流行的有RabbitMq、ActiveMq、ZeroMq、kafka。这几种MQ到底应该选择哪个？要根据自己项目的业务场景和需求。下面我列出这些MQ之间的对比数据和资料。

**第一部分：RabbitMQ,ActiveMq,ZeroMq比较**

**1、 TPS比较 一**

ZeroMq 最好，RabbitMq 次之， ActiveMq 最差。这个结论来自于以下这篇文章。

[http://blog.x-aeon.com/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/](http://blog.x-aeon.com/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/" \t "http://www.cnblogs.com/valor-xh/p/_blank)

测试环境：

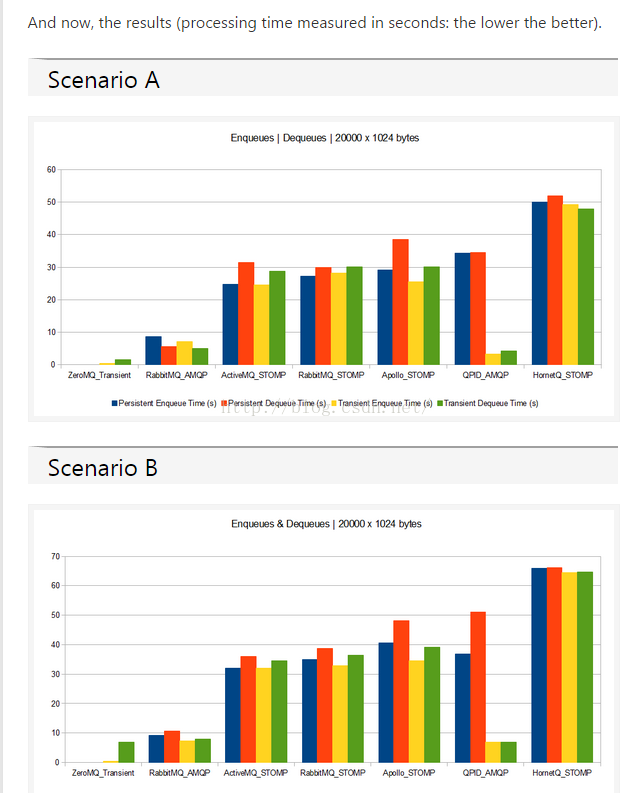
     Model: Dell Studio 1749

     CPU: Intel Core i3 @ 2.40 GHz

     RAM: 4 Gb

     OS: Windows 7 64 bits

其中包括持久化消息和瞬时消息的测试。注意这篇文章里面提到的MQ，都是采用**默认配置**的，并无调优。

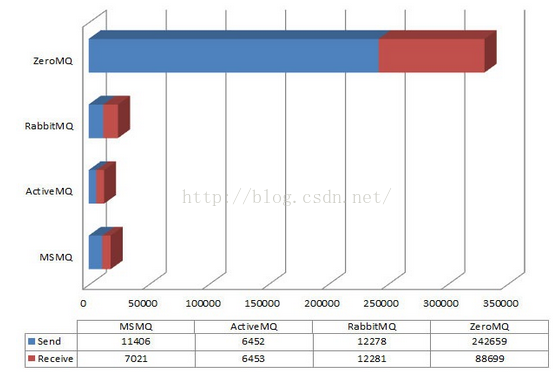


更多的统计图请参看我提供的文章url。

**2、TPS比较**二

ZeroMq 最好，RabbitMq次之， ActiveMq最差。这个结论来自于一下这篇文章。[http://www.cnblogs.com/amityat/archive/2011/08/31/2160293.html](http://www.cnblogs.com/amityat/archive/2011/08/31/2160293.html" \t "http://www.cnblogs.com/valor-xh/p/_blank)

显示的是发送和接受的每秒钟的消息数。整个过程共产生1百万条1K的消息。测试的执行是在一个Windows Vista上进行的。



**3、持久化消息比较**

zeroMq**不支持**，activeMq和rabbitMq都**支持**。持久化消息主要是指：MQ down或者MQ所在的服务器down了，消息不会丢失的机制。

**4、技术点：可靠性、灵活的路由、集群、事务、高可用的队列、消息排序、问题追踪、可视化管理工具、插件系统、社区**

      RabbitMq最好，ActiveMq次之，ZeroMq最差。当然ZeroMq也可以做到，不过自己必须手动写代码实现，代码量不小。尤其是可靠性中的：持久性、投递确认、发布者证实和高可用性。

      所以在可靠性和可用性上，RabbitMQ是首选，虽然ActiveMQ也具备，但是它性能不及RabbitMQ。

**5、高并发**

从实现语言来看，RabbitMQ最高，原因是它的实现语言是天生具备高并发高可用的**erlang**语言。

总结：

按照目前网络上的资料，RabbitMQ、activeM、zeroMQ三者中，综合来看，RabbitMQ是首选。下面提供一篇文章，是淘宝使用RabbitMQ的心得，可以参看一些业务场景。

[http://www.docin.com/p-462677246.html](http://www.docin.com/p-462677246.html" \t "http://www.cnblogs.com/valor-xh/p/_blank)

**第二部分：kafka和RabbitMQ的比较**

关于这两种MQ的比较，网上的资料并不多，最权威的的是kafka的提交者写一篇文章。[http://www.quora.com/What-are-the-differences-between-Apache-Kafka-and-RabbitMQ](http://www.quora.com/What-are-the-differences-between-Apache-Kafka-and-RabbitMQ" \t "http://www.cnblogs.com/valor-xh/p/_blank)

里面提到的要点：

1、  RabbitMq比kafka成熟，在可用性上，稳定性上，可靠性上，RabbitMq超过kafka

2、  Kafka设计的初衷就是处理日志的，可以看做是一个日志系统，针对性很强，所以它并没有具备一个成熟MQ应该具备的特性

3、  Kafka的性能（吞吐量、tps）比RabbitMq要强，这篇文章的作者认为，两者在这方面没有可比性。

这里在附上两篇文章，也是关于kafka和RabbitMq之间的比较的：

1、http://www.mrhaoting.com/?p=139

2、http://www.liaoqiqi.com/post/227