一、资料文档

Kafka：中。有 kafka 作者自己写的书，网上资料也有一些。

rabbitmq：多。有一些不错的书，网上资料多。

zeromq：少。没有专门写 zeromq 的书，网上的资料多是一些代码的实现和简单介绍。

rocketmq：少。没有专门写 rocketmq 的书，网上的资料良莠不齐，官方文档很简洁，但是对技术细节没有过多的描述。

activemq：多。没有专门写 activemq 的书，网上资料多。

二、开发语言

Kafka：Scala

rabbitmq：Erlang

zeromq：c

rocketmq：java

activemq：java

三、支持的协议

Kafka：自己定义的一套...（基于 TCP）

rabbitmq：AMQP

zeromq：TCP、UDP

rocketmq：自己定义的一套...

activemq：OpenWire、STOMP、REST、XMPP、AMQP

四、消息存储

Kafka：内存、磁盘、数据库。支持大量堆积。

kafka 的最小存储单元是分区，一个 topic 包含多个分区，kafka 创建主题时，这些分区会被分配在多个服务器上，通常一个 broker 一台服务器。

分区首领会均匀地分布在不同的服务器上，分区副本也会均匀的分布在不同的服务器上，确保负载均衡和高可用性，当新的 broker 加入集群的时候，部分副本会被移动到新的 broker 上。

根据配置文件中的目录清单，kafka 会把新的分区分配给目录清单里分区数最少的目录。

默认情况下，分区器使用轮询算法把消息均衡地分布在同一个主题的不同分区中，对于发送时指定了 key 的情况，会根据 key 的 hashcode 取模后的值存到对应的分区中。

rabbitmq：内存、磁盘。支持少量堆积。

rabbitmq 的消息分为持久化的消息和非持久化消息，不管是持久化的消息还是非持久化的消息都可以写入到磁盘。

持久化的消息在到达队列时就写入到磁盘，并且如果可以，持久化的消息也会在内存中保存一份备份，这样可以提高一定的性能，当内存吃紧的时候会从内存中清除。非持久化的消息一般只存在于内存中，在内存吃紧的时候会被换入到磁盘中，以节省内存。

引入镜像队列机制，可将重要队列“复制”到集群中的其他 broker 上，保证这些队列的消息不会丢失。配置镜像的队列，都包含一个主节点 master 和多个从节点 slave,如果 master 失效，加入时间最长的 slave 会被提升为新的 master，除发送消息外的所有动作都向 master 发送，然后由 master 将命令执行结果广播给各个 slave，rabbitmq 会让 master 均匀地分布在不同的服务器上，而同一个队列的 slave 也会均匀地分布在不同的服务器上，保证负载均衡和高可用性。

zeromq：消息发送端的内存或者磁盘中。不支持持久化。

rocketmq：磁盘。支持大量堆积。

commitLog 文件存放实际的消息数据，每个 commitLog 上限是 1G，满了之后会自动新建一个 commitLog 文件保存数据。ConsumeQueue 队列只存放 offset、size、tagcode，非常小，分布在多个 broker 上。ConsumeQueue 相当于 CommitLog 的索引文件，消费者消费时会从 consumeQueue 中查找消息在 commitLog 中的 offset，再去 commitLog 中查找元数据。

ConsumeQueue 存储格式的特性，保证了写过程的顺序写盘（写 CommitLog 文件），大量数据 IO 都在顺序写同一个 commitLog，满 1G 了再写新的。加上 rocketmq 是累计 4K 才强制从 PageCache 中刷到磁盘（缓存），所以高并发写性能突出。

activemq：内存、磁盘、数据库。支持少量堆积。

五、消息事务

Kafka：支持

rabbitmq：支持。

客户端将信道设置为事务模式，只有当消息被 rabbitMq 接收，事务才能提交成功，否则在捕获异常后进行回滚。使用事务会使得性能有所下降

zeromq：不支持

rocketmq：支持

activemq：支持

六、负载均衡

Kafka：支持负载均衡。

1>一个 broker 通常就是一台服务器节点。对于同一个 Topic 的不同分区，Kafka 会尽力将这些分区分布到不同的 Broker 服务器上，zookeeper 保存了 broker、主题和分区的元数据信息。分区首领会处理来自客户端的生产请求，kafka 分区首领会被分配到不同的 broker 服务器上，让不同的 broker 服务器共同分担任务。

每一个 broker 都缓存了元数据信息，客户端可以从任意一个 broker 获取元数据信息并缓存起来，根据元数据信息知道要往哪里发送请求。

2>kafka 的消费者组订阅同一个 topic，会尽可能地使得每一个消费者分配到相同数量的分区，分摊负载。

3>当消费者加入或者退出消费者组的时候，还会触发再均衡，为每一个消费者重新分配分区，分摊负载。

kafka 的负载均衡大部分是自动完成的，分区的创建也是 kafka 完成的，隐藏了很多细节，避免了繁琐的配置和人为疏忽造成的负载问题。

4>发送端由 topic 和 key 来决定消息发往哪个分区，如果 key 为 null，那么会使用轮询算法将消息均衡地发送到同一个 topic 的不同分区中。如果 key 不为 null，那么会根据 key 的 hashcode 取模计算出要发往的分区。

rabbitmq：对负载均衡的支持不好。

1>消息被投递到哪个队列是由交换器和 key 决定的，交换器、路由键、队列都需要手动创建。

rabbitmq 客户端发送消息要和 broker 建立连接，需要事先知道 broker 上有哪些交换器，有哪些队列。通常要声明要发送的目标队列，如果没有目标队列，会在 broker 上创建一个队列，如果有，就什么都不处理，接着往这个队列发送消息。假设大部分繁重任务的队列都创建在同一个 broker 上，那么这个 broker 的负载就会过大。（可以在上线前预先创建队列，无需声明要发送的队列，但是发送时不会尝试创建队列，可能出现找不到队列的问题，rabbitmq 的备份交换器会把找不到队列的消息保存到一个专门的队列中，以便以后查询使用）

使用镜像队列机制建立 rabbitmq 集群可以解决这个问题，形成 master-slave 的架构，master 节点会均匀分布在不同的服务器上，让每一台服务器分摊负载。slave 节点只是负责转发，在 master 失效时会选择加入时间最长的 slave 成为 master。

当新节点加入镜像队列的时候，队列中的消息不会同步到新的 slave 中，除非调用同步命令，但是调用命令后，队列会阻塞，不能在生产环境中调用同步命令。

2>当 rabbitmq 队列拥有多个消费者的时候，队列收到的消息将以轮询的分发方式发送给消费者。每条消息只会发送给订阅列表里的一个消费者，不会重复。

这种方式非常适合扩展，而且是专门为并发程序设计的。

如果某些消费者的任务比较繁重，那么可以设置 basicQos 限制信道上消费者能保持的最大未确认消息的数量，在达到上限时，rabbitmq 不再向这个消费者发送任何消息。

3>对于 rabbitmq 而言，客户端与集群建立的 TCP 连接不是与集群中所有的节点建立连接，而是挑选其中一个节点建立连接。

但是 rabbitmq 集群可以借助 HAProxy、LVS 技术，或者在客户端使用算法实现负载均衡，引入负载均衡之后，各个客户端的连接可以分摊到集群的各个节点之中。

客户端均衡算法：

1)轮询法。按顺序返回下一个服务器的连接地址。

2)加权轮询法。给配置高、负载低的机器配置更高的权重，让其处理更多的请求；而配置低、负载高的机器，给其分配较低的权重，降低其系统负载。

3)随机法。随机选取一个服务器的连接地址。

4)加权随机法。按照概率随机选取连接地址。

5)源地址哈希法。通过哈希函数计算得到的一个数值，用该数值对服务器列表的大小进行取模运算。

6)最小连接数法。动态选择当前连接数最少的一台服务器的连接地址。

zeromq：去中心化，不支持负载均衡。本身只是一个多线程网络库。

rocketmq：支持负载均衡。

一个 broker 通常是一个服务器节点，broker 分为 master 和 slave,master 和 slave 存储的数据一样，slave 从 master 同步数据。

1>nameserver 与每个集群成员保持心跳，保存着 Topic-Broker 路由信息，同一个 topic 的队列会分布在不同的服务器上。

2>发送消息通过轮询队列的方式发送，每个队列接收平均的消息量。发送消息指定 topic、tags、keys，无法指定投递到哪个队列（没有意义，集群消费和广播消费跟消息存放在哪个队列没有关系）。

tags 选填，类似于 Gmail 为每封邮件设置的标签，方便服务器过滤使用。目前只支 持每个消息设置一个 tag，所以也可以类比为 Notify 的 MessageType 概念。

keys 选填，代表这条消息的业务关键词，服务器会根据 keys 创建哈希索引，设置后， 可以在 Console 系统根据 Topic、Keys 来查询消息，由于是哈希索引，请尽可能 保证 key 唯一，例如订单号，商品 Id 等。

3>rocketmq 的负载均衡策略规定：Consumer 数量应该小于等于 Queue 数量，如果 Consumer 超过 Queue 数量，那么多余的 Consumer 将不能消费消息。这一点和 kafka 是一致的，rocketmq 会尽可能地为每一个 Consumer 分配相同数量的队列，分摊负载。

activemq：支持负载均衡。可以基于 zookeeper 实现负载均衡。

七、集群方式

Kafka：天然的‘Leader-Slave’无状态集群，每台服务器既是 Master 也是 Slave。

分区首领均匀地分布在不同的 kafka 服务器上，分区副本也均匀地分布在不同的 kafka 服务器上，所以每一台 kafka 服务器既含有分区首领，同时又含有分区副本，每一台 kafka 服务器是某一台 kafka 服务器的 Slave，同时也是某一台 kafka 服务器的 leader。

kafka 的集群依赖于 zookeeper，zookeeper 支持热扩展，所有的 broker、消费者、分区都可以动态加入移除，而无需关闭服务，与不依靠 zookeeper 集群的 mq 相比，这是最大的优势。

rabbitmq：支持简单集群，'复制'模式，对高级集群模式支持不好。

rabbitmq 的每一个节点，不管是单一节点系统或者是集群中的一部分，要么是内存节点，要么是磁盘节点，集群中至少要有一个是磁盘节点。

在 rabbitmq 集群中创建队列，集群只会在单个节点创建队列进程和完整的队列信息（元数据、状态、内容），而不是在所有节点上创建。

引入镜像队列，可以避免单点故障，确保服务的可用性，但是需要人为地为某些重要的队列配置镜像。

zeromq：去中心化，不支持集群。

rocketmq：常用 多对'Master-Slave' 模式，开源版本需手动切换 Slave 变成 Master

Name Server 是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。

Broker 部署相对复杂，Broker 分为 Master 与 Slave，一个 Master 可以对应多个 Slave，但是一个 Slave 只能对应一个 Master，Master 与 Slave 的对应关系通过指定相同的 BrokerName，不同的 BrokerId 来定义，BrokerId 为 0 表示 Master，非 0 表示 Slave。Master 也可以部署多个。每个 Broker 与 Name Server 集群中的所有节点建立长连接，定时注册 Topic 信息到所有 Name Server。

Producer 与 Name Server 集群中的其中一个节点（随机选择）建立长连接，定期从 Name Server 取 Topic 路由信息，并向提供 Topic 服务的 Master 建立长连接，且定时向 Master 发送心跳。Producer 完全无状态，可集群部署。

Consumer 与 Name Server 集群中的其中一个节点（随机选择）建立长连接，定期从 Name Server 取 Topic 路由信息，并向提供 Topic 服务的 Master、Slave 建立长连接，且定时向 Master、Slave 发送心跳。Consumer 既可以从 Master 订阅消息，也可以从 Slave 订阅消息，订阅规则由 Broker 配置决定。

客户端先找到 NameServer, 然后通过 NameServer 再找到 Broker。

一个 topic 有多个队列，这些队列会均匀地分布在不同的 broker 服务器上。rocketmq 队列的概念和 kafka 的分区概念是基本一致的，kafka 同一个 topic 的分区尽可能地分布在不同的 broker 上，分区副本也会分布在不同的 broker 上。

rocketmq 集群的 slave 会从 master 拉取数据备份，master 分布在不同的 broker 上。

activemq：支持简单集群模式，比如'主-备'，对高级集群模式支持不好。

八、管理界面

Kafka：一般

rabbitmq：好

zeromq：无

rocketmq：无

activemq：一般

九、可用性

Kafka：非常高（分布式）

rabbitmq：高（主从）

zeromq：高。

rocketmq：非常高（分布式）

activemq：高（主从）

十、消息重复

Kafka：支持 at least once、at most once

rabbitmq：支持 at least once、at most once

zeromq：只有重传机制，但是没有持久化，消息丢了重传也没有用。既不是 at least once、也不是 at most once、更不是 exactly only once

rocketmq：支持 at least once

activemq：支持 at least once

十一、吞吐量 TPS

Kafka：极大

Kafka 按批次发送消息和消费消息。发送端将多个小消息合并，批量发向 Broker，消费端每次取出一个批次的消息批量处理。

rabbitmq：比较大

zeromq：极大

rocketmq：大

rocketMQ 接收端可以批量消费消息，可以配置每次消费的消息数，但是发送端不是批量发送。

activemq：比较大

十二、订阅形式和消息分发

Kafka：基于 topic 以及按照 topic 进行正则匹配的发布订阅模式。

【发送】

发送端由 topic 和 key 来决定消息发往哪个分区，如果 key 为 null，那么会使用轮询算法将消息均衡地发送到同一个 topic 的不同分区中。如果 key 不为 null，那么会根据 key 的 hashcode 取模计算出要发往的分区。

【接收】

1>consumer 向群组协调器 broker 发送心跳来维持他们和群组的从属关系以及他们对分区的所有权关系，所有权关系一旦被分配就不会改变除非发生再均衡(比如有一个 consumer 加入或者离开 consumer group)，consumer 只会从对应的分区读取消息。

2>kafka 限制 consumer 个数要少于分区个数,每个消息只会被同一个 Consumer Group 的一个 consumer 消费（非广播）。

3>kafka 的 Consumer Group 订阅同一个 topic，会尽可能地使得每一个 consumer 分配到相同数量的分区，不同 Consumer Group 订阅同一个主题相互独立，同一个消息会被不同的 Consumer Group 处理。

rabbitmq：提供了 4 种：direct, topic ,Headers 和 fanout。

【发送】

先要声明一个队列，这个队列会被创建或者已经被创建，队列是基本存储单元。

由 exchange 和 key 决定消息存储在哪个队列。

direct>发送到和 bindingKey 完全匹配的队列。

topic>路由 key 是含有"."的字符串，会发送到含有“*”、“#”进行模糊匹配的 bingKey 对应的队列。

fanout>与 key 无关，会发送到所有和 exchange 绑定的队列

headers>与 key 无关，消息内容的 headers 属性（一个键值对）和绑定键值对完全匹配时，会发送到此队列。此方式性能低一般不用

【接收】

rabbitmq 的队列是基本存储单元，不再被分区或者分片，对于我们已经创建了的队列，消费端要指定从哪一个队列接收消息。

当 rabbitmq 队列拥有多个消费者的时候，队列收到的消息将以轮询的分发方式发送给消费者。每条消息只会发送给订阅列表里的一个消费者，不会重复。

这种方式非常适合扩展，而且是专门为并发程序设计的。

如果某些消费者的任务比较繁重，那么可以设置 basicQos 限制信道上消费者能保持的最大未确认消息的数量，在达到上限时，rabbitmq 不再向这个消费者发送任何消息。

zeromq：点对点(p2p)

rocketmq：基于 topic/messageTag 以及按照消息类型、属性进行正则匹配的发布订阅模式

【发送】

发送消息通过轮询队列的方式发送，每个队列接收平均的消息量。发送消息指定 topic、tags、keys，无法指定投递到哪个队列（没有意义，集群消费和广播消费跟消息存放在哪个队列没有关系）。

tags 选填，类似于 Gmail 为每封邮件设置的标签，方便服务器过滤使用。目前只支 持每个消息设置一个 tag，所以也可以类比为 Notify 的 MessageType 概念。

keys 选填，代表这条消息的业务关键词，服务器会根据 keys 创建哈希索引，设置后， 可以在 Console 系统根据 Topic、Keys 来查询消息，由于是哈希索引，请尽可能 保证 key 唯一，例如订单号，商品 Id 等。

【接收】

1>广播消费。一条消息被多个 Consumer 消费，即使 Consumer 属于同一个 ConsumerGroup，消息也会被 ConsumerGroup 中的每个 Consumer 都消费一次。

2>集群消费。一个 Consumer Group 中的 Consumer 实例平均分摊消费消息。例如某个 Topic 有 9 条消息，其中一个 Consumer Group 有 3 个实例，那么每个实例只消费其中的 3 条消息。即每一个队列都把消息轮流分发给每个 consumer。

activemq：点对点(p2p)、广播（发布-订阅）

点对点模式，每个消息只有 1 个消费者；

发布/订阅模式，每个消息可以有多个消费者。

【发送】

点对点模式：先要指定一个队列，这个队列会被创建或者已经被创建。

发布/订阅模式：先要指定一个 topic，这个 topic 会被创建或者已经被创建。

【接收】

点对点模式：对于已经创建了的队列，消费端要指定从哪一个队列接收消息。

发布/订阅模式：对于已经创建了的 topic，消费端要指定订阅哪一个 topic 的消息。

十三、顺序消息

Kafka：支持。

设置生产者的 max.in.flight.requests.per.connection 为 1，可以保证消息是按照发送顺序写入服务器的，即使发生了重试。

kafka 保证同一个分区里的消息是有序的，但是这种有序分两种情况

1>key 为 null，消息逐个被写入不同主机的分区中，但是对于每个分区依然是有序的

2>key 不为 null，消息被写入到同一个分区，这个分区的消息都是有序。

rabbitmq：不支持

zeromq：不支持

rocketmq：支持

activemq：不支持

十四、消息确认

Kafka：支持。

1>发送方确认机制

ack=0，不管消息是否成功写入分区

ack=1，消息成功写入首领分区后，返回成功

ack=all，消息成功写入所有分区后，返回成功。

2>接收方确认机制

自动或者手动提交分区偏移量，早期版本的 kafka 偏移量是提交给 Zookeeper 的，这样使得 zookeeper 的压力比较大，更新版本的 kafka 的偏移量是提交给 kafka 服务器的，不再依赖于 zookeeper 群组，集群的性能更加稳定。

rabbitmq：支持。

1>发送方确认机制，消息被投递到所有匹配的队列后，返回成功。如果消息和队列是可持久化的，那么在写入磁盘后，返回成功。支持批量确认和异步确认。

2>接收方确认机制，设置 autoAck 为 false，需要显式确认，设置 autoAck 为 true，自动确认。

当 autoAck 为 false 的时候，rabbitmq 队列会分成两部分，一部分是等待投递给 consumer 的消息，一部分是已经投递但是没收到确认的消息。如果一直没有收到确认信号，并且 consumer 已经断开连接，rabbitmq 会安排这个消息重新进入队列，投递给原来的消费者或者下一个消费者。

未确认的消息不会有过期时间，如果一直没有确认，并且没有断开连接，rabbitmq 会一直等待，rabbitmq 允许一条消息处理的时间可以很久很久。

zeromq：支持。

rocketmq：支持。

activemq：支持。

十五、消息回溯

Kafka：支持指定分区 offset 位置的回溯。

rabbitmq：不支持

zeromq：不支持

rocketmq：支持指定时间点的回溯。

activemq：不支持

十六、消息重试

Kafka：不支持，但是可以实现。

kafka 支持指定分区 offset 位置的回溯，可以实现消息重试。

rabbitmq：不支持，但是可以利用消息确认机制实现。

rabbitmq 接收方确认机制，设置 autoAck 为 false。

当 autoAck 为 false 的时候，rabbitmq 队列会分成两部分，一部分是等待投递给 consumer 的消息，一部分是已经投递但是没收到确认的消息。如果一直没有收到确认信号，并且 consumer 已经断开连接，rabbitmq 会安排这个消息重新进入队列，投递给原来的消费者或者下一个消费者。

zeromq：不支持，

rocketmq：支持。

消息消费失败的大部分场景下，立即重试 99%都会失败，所以 rocketmq 的策略是在消费失败时定时重试，每次时间间隔相同。

1>发送端的 send 方法本身支持内部重试，重试逻辑如下：

a)至多重试 3 次；

b)如果发送失败，则轮转到下一个 broker；

c)这个方法的总耗时不超过 sendMsgTimeout 设置的值，默认 10s，超过时间不在重试。

2>接收端。

Consumer 消费消息失败后，要提供一种重试机制，令消息再消费一次。
Consumer 消费消息失败通常可以分为以下两种情况：

1. 由于消息本身的原因，例如反序列化失败，消息数据本身无法处理（例如话费充值，当前消息的手机号被

注销，无法充值）等。定时重试机制，比如过 10s 秒后再重试。

2. 由于依赖的下游应用服务不可用，例如 db 连接不可用，外系统网络不可达等。

即使跳过当前失败的消息，消费其他消息同样也会报错。这种情况可以 sleep 30s，再消费下一条消息，减轻 Broker 重试消息的压力。

activemq：不支持

十七、并发度

Kafka：高

一个线程一个消费者，kafka 限制消费者的个数要小于等于分区数，如果要提高并行度，可以在消费者中再开启多线程，或者增加 consumer 实例数量。

rabbitmq：极高

本身是用 Erlang 语言写的，并发性能高。

可在消费者中开启多线程，最常用的做法是一个 channel 对应一个消费者，每一个线程把持一个 channel，多个线程复用 connection 的 tcp 连接，减少性能开销。

当 rabbitmq 队列拥有多个消费者的时候，队列收到的消息将以轮询的分发方式发送给消费者。每条消息只会发送给订阅列表里的一个消费者，不会重复。

这种方式非常适合扩展，而且是专门为并发程序设计的。

如果某些消费者的任务比较繁重，那么可以设置 basicQos 限制信道上消费者能保持的最大未确认消息的数量，在达到上限时，rabbitmq 不再向这个消费者发送任何消息。

zeromq：高

rocketmq：高

1>rocketmq 限制消费者的个数少于等于队列数，但是可以在消费者中再开启多线程，这一点和 kafka 是一致的，提高并行度的方法相同。

修改消费并行度方法

a) 同一个 ConsumerGroup 下，通过增加 Consumer 实例数量来提高并行度，超过订阅队列数的 Consumer 实例无效。

b) 提高单个 Consumer 的消费并行线程，通过修改参数 consumeThreadMin、consumeThreadMax

2>同一个网络连接 connection，客户端多个线程可以同时发送请求，连接会被复用，减少性能开销。

activemq：高