# Kafka、RabbitMQ、RocketMQ消息中间件对比报告

|  |  |  |
| --- | --- | --- |
| 状态 ：  [ V ] 草稿  [ ] 修改中  [ ] 定稿 | 文件标签： | Kafka、RabbitMQ、RocketMQ消息中间件的对比报告 |
| 版本： | V0.1 |
| 作者： |  |
| 日期： | 2018-12-14 |

**编辑历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文件名称：Kafka、RabbitMQ、RocketMQ消息中间件的对比报告 | | | | |
|  | | | | |
| 文件说明： | | | | |
|  | | | | |
| 编辑历史： | | | | |
| 编辑时间 | 版本 | 作者 | 编辑内容 | 标记 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 引言

分布式系统中,我们广泛运用消息中间件进行系统间的数据交换,便于异步解耦。现在开源的消息中间件有很多(如Kafka、RabbitMQ、RocketMQ)。目前智慧社区采用RabbitMQ作为消息中间件，但是对于海花岛预估约2w台设备的数量级，RabbitMQ可能存在一些性能隐患，因此我们对这三种(Kafka、RabbitMQ、RocketMQ)消息中间件压测，做一些性能分析，选取合适的消息中间件作为新项目的采用。

Kafka是LinkedIn开源的分布式发布-订阅消息系统，目前归属于Apache定级项目。Kafka主要特点是基于Pull的模式来处理消息消费，追求高吞吐量，一开始的目的就是用于日志收集和传输。0.8版本开始支持复制，不支持事务，对消息的重复、丢失、错误没有严格要求，适合产生大量数据的互联网服务的数据收集业务。

RabbitMQ是使用Erlang语言开发的开源消息队列系统，基于AMQP协议来实现。AMQP的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。AMQP协议更多用在企业系统内，对数据一致性、稳定性和可靠性要求很高的场景，对性能和吞吐量的要求还在其次。

RocketMQ是阿里开源的消息中间件，它是纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。RocketMQ思路起源于Kafka，但并不是Kafka的一个Copy，它对消息的可靠传输及事务性做了优化，目前在阿里集团被广泛应用于交易、充值、流计算、消息推送、日志流式处理、binglog分发等场景。

# 组件间的对比参考

**基础对比**

|  |  |  |  |
| --- | --- | --- | --- |
| - | kafka | RocketMQ | RabbitMQ |
| 设计定位 | 系统间的数据流管道，实时数据处理。 例如：常规的消息系统、网站活性跟踪，监控数据，日志收集、处理等 | 非日志的可靠消息传输。 例如：订单，交易，充值，流计算，消息推送，日志流式处理，binglog分发等 | 可靠消息传输。和RocketMQ类似。 |
| 成熟度 | 日志领域成熟 | 成熟 | 成熟 |
| 所属社区／公司 | Apache | Alibaba开发，已加入到Apache下 | Mozilla Public License |
| 社区活跃度 | 高 | 中 | 高 |
| API完备性 | 高 | 高 | 高 |
| 文档完备性 | 高 | 高 | 高 |
| 开发语言 | Scala | Java | Erlang |
| 支持协议 | 一套自行设计的基于TCP的二进制协议 | 自己定义的一套 (社区提供JMS--不成熟) | AMQP |
| 客户端语言 | C/C++、Python、Go、Erlang、.NET、Ruby、Node.js、PHP等 | Java | Java、C、 C++、 Python、 PHP、Perl 等 |
| 持久化方式 | 磁盘文件 | 磁盘文件 | 内存、文件 |

**可用性可靠性分析**

|  |  |  |  |
| --- | --- | --- | --- |
| - | kafka | RocketMQ | RabbitMQ |
| 部署方式 | 单机／集群 | 单机／集群 | 单机／集群 |
| 集群管理 | zookeeper | name server |  |
| 选主方式 | 从ISR中自动选举一个leader | 不支持自动选主。通过设定brokername、brokerId实现，brokername相同，brokerid=0时为maser，其他为slave | 最早加入集群的broker |
| 可用性 | 非常高 分布式、主从 | 非常高 分布式、主从 | 高 主从，采用镜像模式实现，数据量大时可能产生性能瓶颈 |
| 主从切换 | 自动切换 N个副本，允许N-1个失效；master失效以后自动从isr中选择一个主； | 不支持自动切换 master失效以后不能向master发送信息，consumer大概30s（默认）可以感知此事件，此后从slave消费；如果master无法恢复，异步复制时可能出现部分信息丢失 | 自动切换 最早加入集群的slave会成为master；因为新加入的slave不同步master之前的数据，所以可能会出现部分数据丢失 |
| 数据可靠性 | 很好 支持producer单条发送、同步刷盘、同步复制、异步。 | 很好 producer单条发送，broker端支持同步刷盘、异步刷盘，同步双写，异步复制。 | 好 producer支持同步／异步ack。支持队列数据持久化，镜像模式中支持主从同步 |
| 消息写入性能 | 非常好 每条10个字节测试：百万条/s | 很好 每条10个字节测试：单机单broker约7w/s，单机3个broker约12w/s | RAM约为RocketMQ的1/2， Disk的性能约为RAM性能的1/3 |
| 性能的稳定性 | 队列/分区多时性能不稳定，明显下降。 消息堆积时性能稳定 | 队列较多、消息堆积时性能稳定 | 消息堆积时，性能不稳定、明显下降 |
| 单机支持的队列数 | 单机超过64个队列/分区，Load会发生明显的飙高现象，队列越多，load越高，发送消息响应时间变长 | 单机支持最高5万个队列，Load不会发生明显变化 | 依赖于内存 |
| 堆积能力 | 非常好 消息存储在log中，每个分区由一个或多个segment  log文件 | 非常好 所有消息存储在同一个commit log中 | 一般 生产者、消费者正常时，性能表现稳定；消费者不消费时，性能不稳定 |
| 复制备份 | 消息先写入leader的log，followers从leader中pull数据，pull到数据以后先ack leader，然后写入log中。 ISR中维护与leader同步的列表，落后太多的follwer会被删除掉 | 同步双写 异步复制：slave启动线程从master中拉数据 | 普通模式下不复制； 镜像模式下：消息先到mster，然后写到slave上。加入集群之前的消息不会被复制到新的slave上。 |
| 消息投递实时性 | 毫秒级 具体由consumer轮询间隔时间决定 | 毫秒级 支持pull、push两种模式，延时通常在毫秒级 | 毫秒级 |

**运维**

|  |  |  |  |
| --- | --- | --- | --- |
| - | kafka | RocketMQ | RabbitMQ |
| 系统维护 | Scala语言开发，维护成本高 | java语言开发，维护成本低 | Erlang语言开发，维护成本高 |
| 部署依赖 | zookeeper | nameserver | Erlang环境 |
| 管理后台 | 官网不提供，第三方开源管理工具可供使用；不用重新开发 | 官方提供，rocketmq-console | 官方提供rabbitmqadmin |
| 管理后台功能 | Kafka Web Conslole Brokers列表；Kafka 集群中 Topic列表，及对应的Partition、LogSize等信息；Topic对应的Consumer Groups、Offset、Lag等信息； 生产和消费流量图、消息预览 KafkaOffsetMonitor： Kafka集群状态；Topic、Consumer Group列表；图形化展示topic和consumer之间的关系；图形化展示consumer的Offset、Lag等信息 Kafka Manager 管理几个不同的集群；监控集群的状态(topics, brokers, 副本分布, 分区分布)；产生分区分配(Generate partition assignments)基于集群的当前状态；重新分配分区 | Cluster、Topic、Connection、NameServ、Message、Broker、Offset、Consumer | overview、connections、channels、exchanges、queues、admin |

**总结**

|  |  |  |  |
| --- | --- | --- | --- |
| - | kafka | RocketMQ | RabbitMQ |
| 优点 | 1、在高吞吐、低延迟、高可用、集群热扩展、集群容错上有非常好的表现； 2、producer端提供缓存、压缩功能，可节省性能，提高效率。 3、提供顺序消费能力 4、提供多种客户端语言 5、生态完善，在大数据处理方面有大量配套的设施。 | 1、在高吞吐、低延迟、高可用上有非常好的表现；消息堆积时，性能也很好。 2、api、系统设计都更加适在业务处理的场景。 3、支持多种消费方式。 4、支持broker消息过滤。 5、支持事务。 6、提供消息顺序消费能力；consumer可以水平扩展，消费能力很强。 7、集群规模在50台左右，单日处理消息上百亿；经历过大数据量的考验，比较稳定可靠。 | 1、在高吞吐量、高可用上较前两者有所不如。 2、支持多种客户端语言；支持amqp协议。 3、由于erlang语言的特性，性能也比较好; 使用RAM模式时，性能很好。 4、管理界面较丰富，在互联网公司也有较大规模的应用; |
| 缺点 | 1、消费集群数目受到分区数目的限制。 2、单机topic多时，性能会明显降低。 3、不支持事务 | 1、相比于kafka，使用者较少，生态不够完善。消息堆积、吞吐率上也有所不如。 2、不支持主从自动切换，master失效后，消费者需要一定的时间才能感知。 3、客户端只支持Java | 1、erlang 语言难度较大。集群不支持动态扩展。 2、不支持事务、消息吞吐能力有限 3、消息堆积时，性能会明显降低 |

# 测试环境

服务端单机部署，机器配置如下：

|  |  |
| --- | --- |
| CPU | Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz |
| 内存 | 16G |
| 硬盘 | 200G |
| 网卡 | 1000Mb/s |

应用版本：

|  |  |
| --- | --- |
| 消息中间件 | 版本 |
| Kafka | 2.1.0 |
| RocketMQ | 4.3.2 |
| RabbitMQ | 3.6.14 |

测试脚本：

|  |  |
| --- | --- |
| 压力端 | Jmeter的Java客户端 |
| 消息大小 | 128字节、3000字节、5000字节 |
| 并发数 | 能达到服务端最大TPS |
|  |  |

# 测试结果

测试结果表：

消息大小：128字节

|  |  |  |  |
| --- | --- | --- | --- |
| 消息中间件 | 消息大小 | 消息是否压缩 | 生产消息TPS |
| Kafka | 128byte | N | 58w/s |
| RocketMQ | 128byte | N | 7.5w/s |
| RabbitMQ | 128byte | N | 4w/s |

消息大小：3000字节

|  |  |  |  |
| --- | --- | --- | --- |
| 消息中间件 | 消息大小 | 消息是否压缩 | 生产消息TPS |
| Kafka | 3000byte | N | 2.9w/s |
| RocketMQ | 3000byte | N | 2.4w/s |
| RabbitMQ | 3000byte | N | 3w/s |

消息大小：5000字节

|  |  |  |  |
| --- | --- | --- | --- |
| 消息中间件 | 消息大小 | 消息是否压缩 | 生产消息TPS |
| Kafka | 5000byte | N | 2w/s |
| RocketMQ | 5000byte | Y | 5.1w/s |
| RabbitMQ | 5000byte | N | 1.8w/s |

由于RocketMQ,默认当消息大小大于4k时，会对消息进行压缩发送，而Kafka也能对消息进行压缩，因此进行RocketMQ与Kafka在消息压缩的情况下生产消息的TPS对比，结果如下

|  |  |  |  |
| --- | --- | --- | --- |
| 消息中间件 | 消息大小 | 消息是否压缩 | 生产消息TPS |
| Kafka | 5000byte | Y | 7.4w/s |
| RocketMQ | 5000byte | Y | 5.1w/s |

# 测试结论

Kafka,通过以上的压测发现在发送128字节的短消息时其吞吐量高达58w/s,远超其他两个MQ，但是当发送5000个字节的消息时，由于此时broker磁盘IO已达瓶颈，导致吞吐量严重下降。当使用lz4压缩算法对消息进行压缩后再发送，可提高吞吐量并超过RocketMQ。

RocketMQ,通过以上的压测发现其使用比较稳定，随着消息的大小也会出现吞吐量的下降，但还算比较稳定，其主要受限于CPU资源的损耗比较严重，RocketMQ默认会在当消息体大小超过4K时会对消息进行压缩，因此出现发送3000字节的消息的吞吐量比发送5000字节的吞吐量低的现象。

RabbitMQ,通过以上的压测发现其在高压生产消息的情况下会出现假死的现象，而且极其不稳定，且其吞吐量也比较低，CPU消耗较高。