消息队列中间件调研文档

MQ 对比

基本信息对比

主要关注前三个（标红）

	ActiveMQ	RabbitMQ	RocketMq	Joram	HornetQ	OpenMQ	MuleMQ	SonicMQ	ZeroMQ
关注度	高	高	中	中	中	中	低	低	中
成熟度	成熟	成熟	比较成熟	比较成熟	比较成熟	比较成熟	新产品无成功案例	成熟	不成熟
所属社区/公司	Apache	Mozilla Public License	Alibaba	OW2	Jboss	Sun	Mule	Progress	
社区活跃度	高	高	中	中	中	低	高	低	低
文档	多	多	中	多	中	中	少	少	中
特点	功能齐全，被大量开源项目使用	由于 Erlang 语言的并发能力，性能很好	各个环节分布式扩展设计，主从 HA；支持上万个队列；多种消费模式；性能很好		在 Linux 平台上直接调用操作系统的 AIO，性能得到很大的提升		性能非常好，与 MuleESB 无缝整合	性能优越的商业 MQ	低延时，高性能，最高 43 万条消息每秒
授权方式	**开源**	**开源**	**开源**	开源	开源	开源	商业	商业	开源
开发语言	Java	Erlang	Java	Java	Java	Java	Java	Java	C
支持的协议	OpenWire、STOMP、REST、XMPP、AMQP	AMQP	自己定义的一套(社区提供 JMS--不成熟)	JMS	JMS	JMS	JMS	JMS	TCP、UDP
客户端支持语言	**Java**、C、C++、Python、PHP、Perl、.net 等	**Java**、C、C++、Python、PHP、Perl 等	Java C++（不成熟）	Java	Java	Java	Java	Java、C、C++、.net	python、java、php、.net 等
持久化	内存、文件、数据库	内存、文件	磁盘文件	内存、文件	内存、文件	内存、文件	内存、文件	内存、文件、数据库	在消息发送端保存
事务	**支持**	**不支持**	**支持**	支持	支持	支持	支持	支持	不支持
集群	支持	支持	支持	支持	支持	支持	支持	支持	不支持
负载均	支持	支持	支持	支持	支持	支持	支持	支持	不支持

作者 何鹏　关注分布式存储与计算相关框架，包括 Hadoop、YARN、HBase、Storm、Spark、MQ 等　peng.he.ia@gmail.com

衡									
管理界面	一般	好	无 社区有 web console 实现	一般	无	一般	一般	好	无
部署方式	独立、嵌入	独立	独立	独立、嵌入	独立、嵌入	独立、嵌入	独立	独立	独立
评价	**优点**： 成熟的产品，已经在很多公司得到应用（非大规模场景）。有较多的文档。各种协议支持较好，有多重语言的成熟的客户端； **缺点**： 根据其他用户反馈，会出莫名其妙的问题，切会丢失消息。 其重心放到 activemq6.0 产品—apollo 上去了，目前社区不活跃，且对 5.x 维护较少； **Activemq 不适合用于上千个队列的应用场景**	**优点**： 由于 erlang 语言的特性 ,mq 性能较好；管理界面较丰富，在互联网公司也有较大规模的应用；支持 amqp 系诶，有多中语言且支持 amqp 的客户端可用 缺点： erlang 语言难度较大。集群不支持动态扩展。	**优点**： 模型简单，接口易用（JMS 的接口很多场合并不太实用）。在阿里大规模应用。目前支付宝中的余额宝等新兴产品均使用 rocketmq。集群规模大概在 50 台左右，单日处理消息上百亿；性能非常好，可以大量堆积消息在 broker 中；支持多种消费，包括集群消费、广播消费等。开发度较活跃，版本更新很快。 **缺点**： 产品较新，文档比较缺乏。 没有在 mq 核心中去实现 JMS 等接口 ,对已有系统而言不能兼容。 阿里内部还有一套未开源的 MQ API，这一层 API 可以将上层应用和下层 MQ 的实现解耦（阿里内部有多个 mq 的实现，如 notify、						

			metaq1.x，metaq2.x，rocketmq 等），使得下面 mq 可以很方便的进行切换和升级而对应用无任何影响，目前这一套东西未开源						

支持的 Protocols 对比

The following table lists the supported protocols of each broker.

	ActiveMQ	RabbitMQ	RocketMQ	HornetQ	Qpid	ZeroMQ
AMQP	1.0	0-8, 0-9, 0-9-1	-	announced	1.0	-
MQTT	✓	✓	-	-	-	-
OpenWire	✓	-	-	-	-	-
REST	✓	✓	-	✓	-	-
STOMP	✓	✓	-	✓	-	-
STOMP over Websockets	✓	✓	-	✓	-	-
XMPP	✓	Over Gateway	-	-	-	-

Client Interfaces 对比

Access to a message broker is not only reserved for Java applications. For many brokers there are client APIs in different programming languages. It is even common for client and broker to use different platforms, eg when a Java application accesses an Erlang written RabbitMQ broker.Since AMQP has standardized the protocol on the "cable level", it is even possible to connect the AMQP client library of a broker of another broker. For this to work the AMQP versions have to match because each version of the protocol are not compatible.Through the STOMP protocol different client platforms can be connected to a broker. STOMP is favorited to access a broker with scripting languages such as Perl, PHP and Ruby.

	ActiveMQ	RabbitMQ	RocketMQ	HornetQ	Qpid	ZeroQ
C	✓	✓	-	-	✓	✓
C++	-	✓	✓	-	✓	✓
Erlang	-	✓	-	-	-	✓
Haskell	-	✓	-	-	-	✓
Java JMS	✓	- ✓	✓	✓	✓	-

	ActiveMQ	RabbitMQ	RocketMQ	HornetQ	Qpid	ZeroQ
Java proprietary	✓	✓	-	✓	-	✓
.NET	-	✓	-	-	✓	✓
Objective-C	-	-	-	-	-	✓
Perl	-	✓	-	-	-	✓
PHP	-	✓	-	-	-	✓
Python	-	✓	-	-	✓	✓
Ruby	-	✓	-	-	✓	✓

mq benchmark[文献 1]

结论

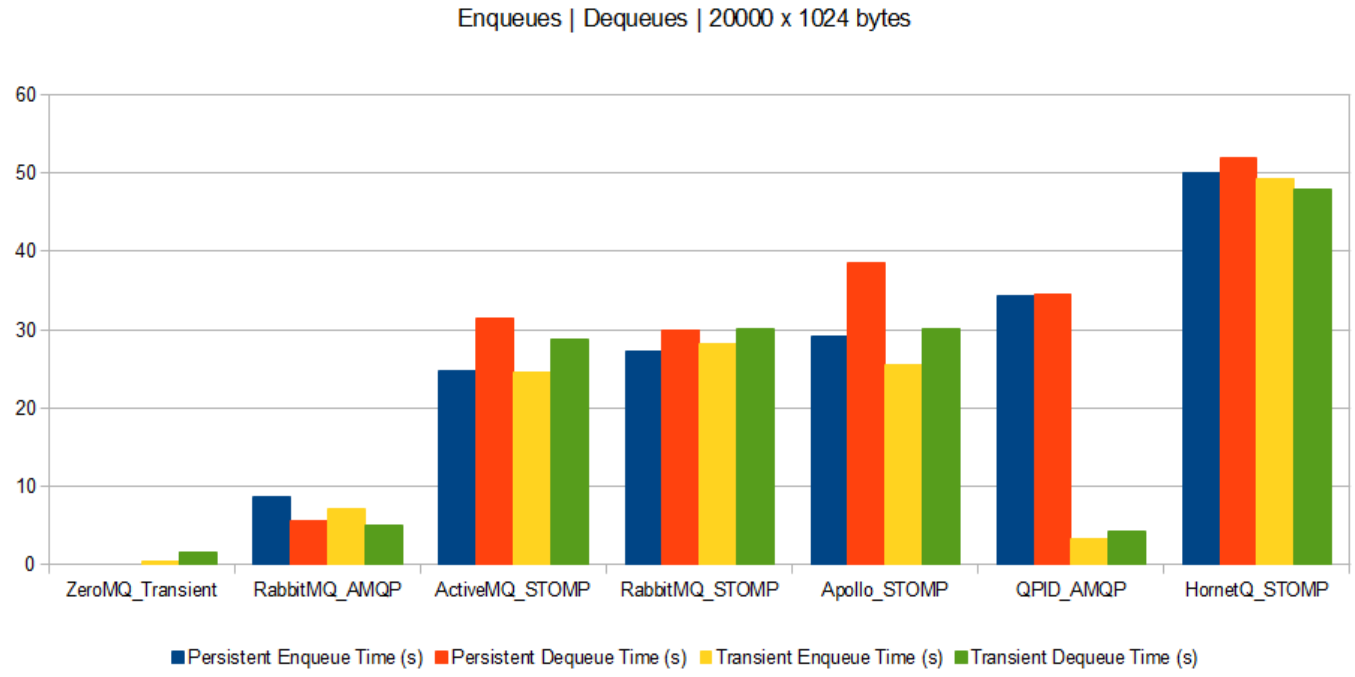
1、 在传统的 MQ 中，rabbitmq 的性能表现最好。
2、 kafka 的性能优于 rabbitmq；

传统 MQ 对比

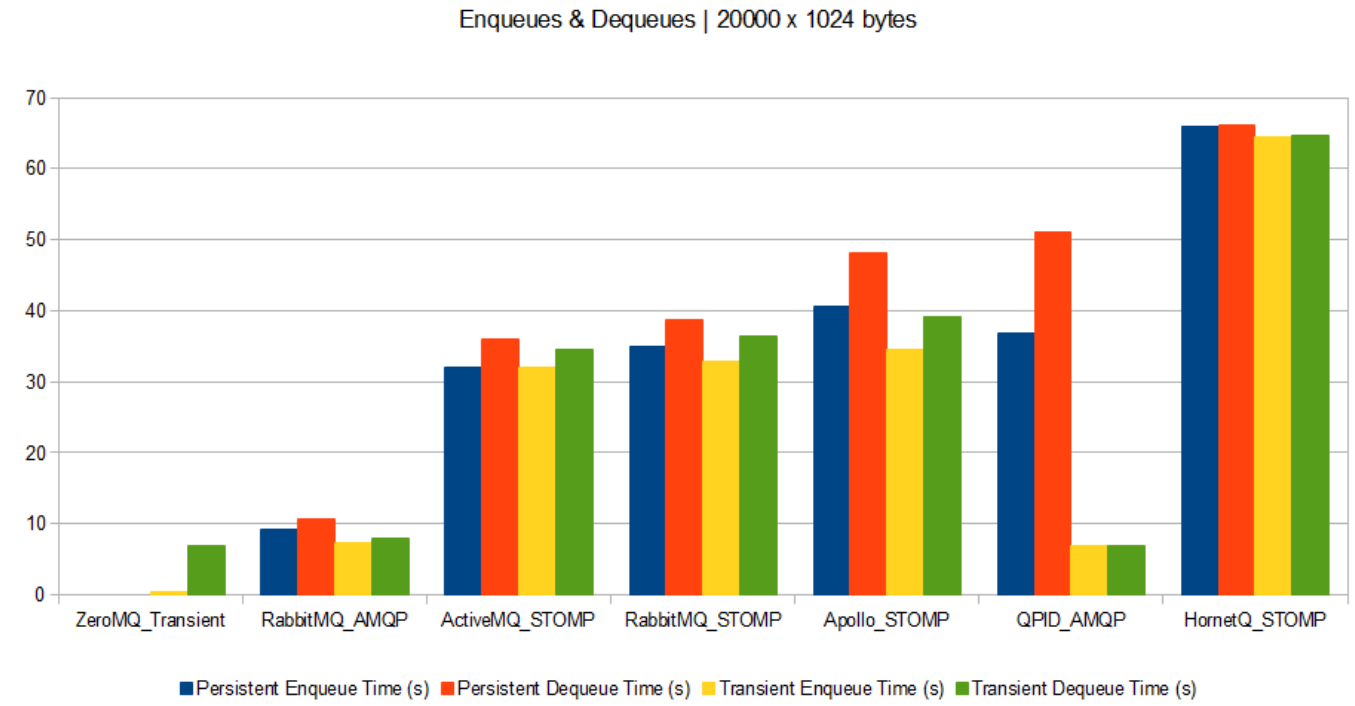
场景设计：

- **Scenario A:** Enqueuing **20,000** messages of **1024** bytes each, then dequeuing them afterwards.
- **Scenario B:** Enqueuing and dequeuing simultaneously **20,000** messages of**1024** bytes each.
- **Scenario C:** Enqueuing and dequeuing simultaneously **200,000** messages of**32** bytes each.
- **Scenario D:** Enqueuing and dequeuing simultaneously **200** messages of**32768** bytes each.

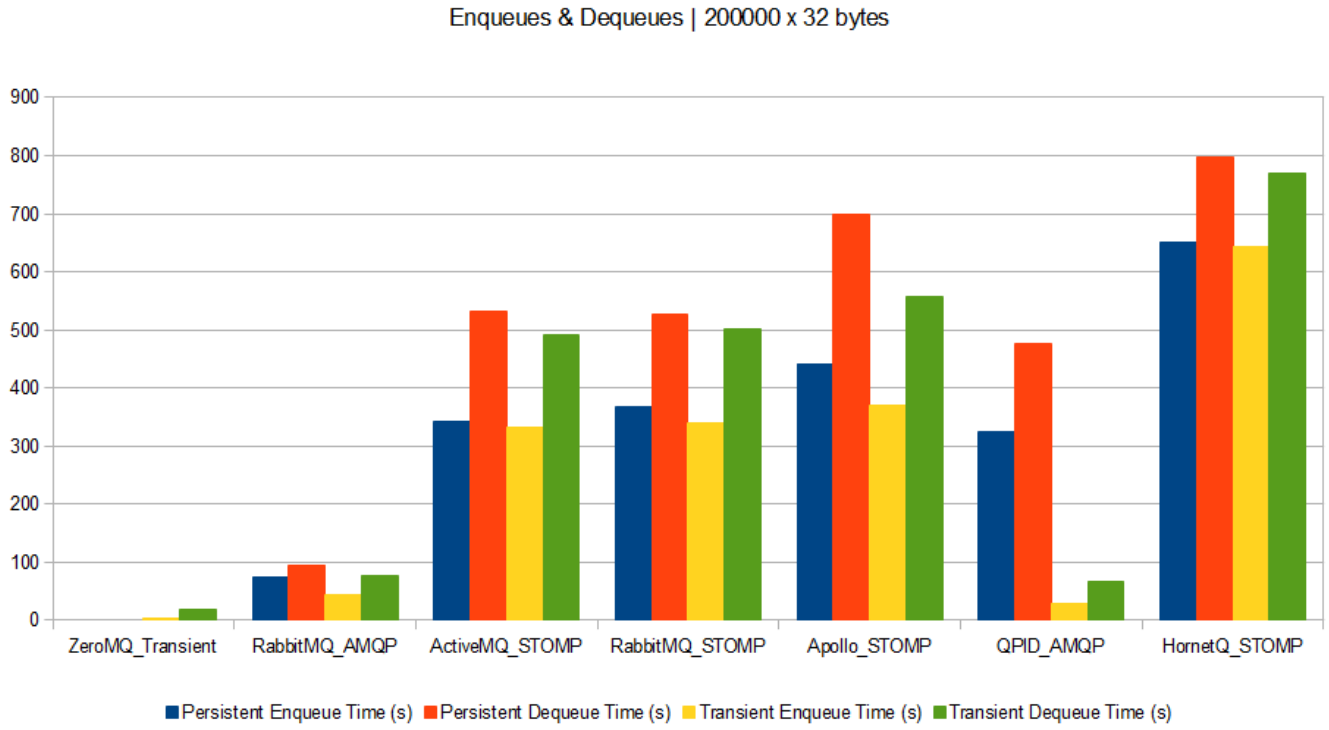
Scenario A



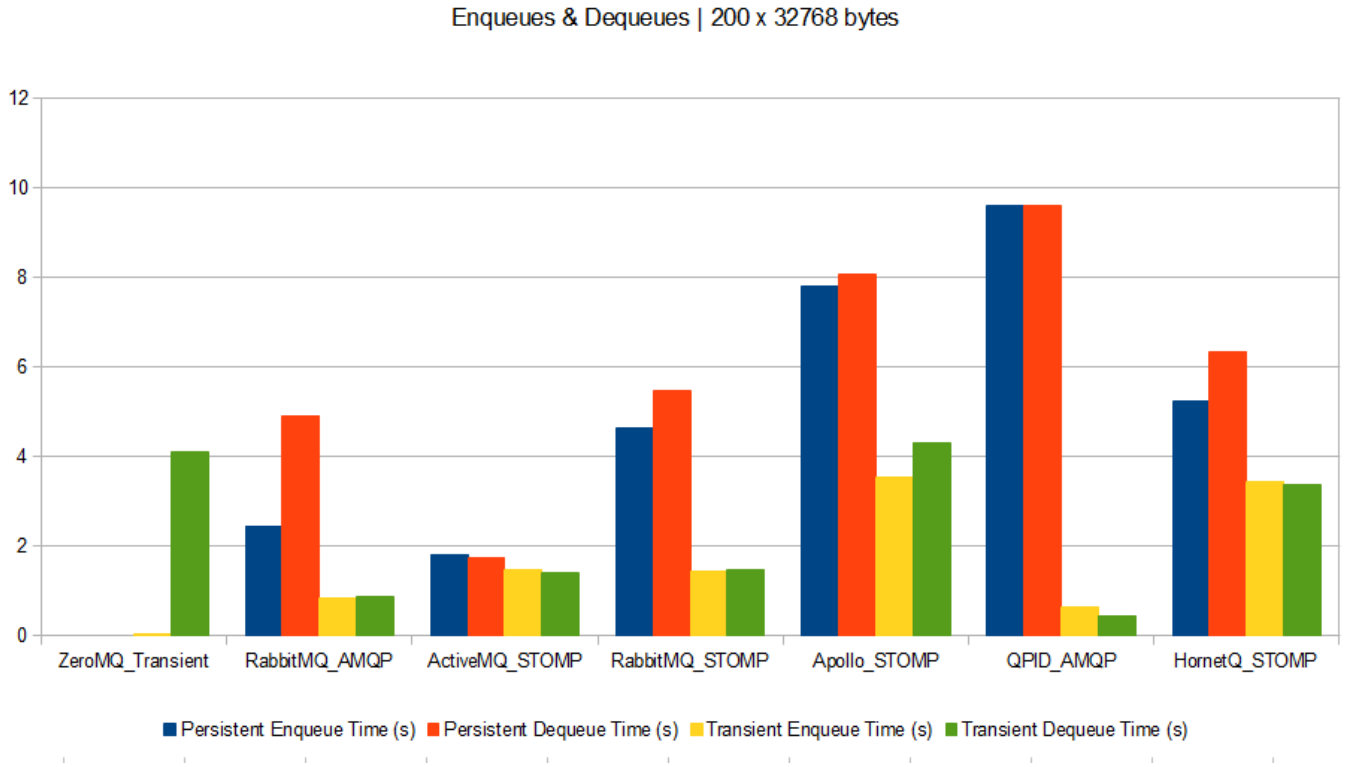
Scenario B



Scenario C



Scenario D



Kafka vs rabbitmq

结论

在 kafka 和 rabbitmq 的对比中，kafka 的性能表现较好。

作者 何鹏　关注分布式存储与计算相关框架，包括 Hadoop、YARN、HBase、Storm、Spark、MQ 等　peng.he.ia@gmail.com

简介

rocketmq 的前身为 metaq，metaq 的第一个版本是可以看成是 linkedin 的 kafka（scala）的 java 版本，并对其增加了事务的支持。rocketmq 为 metaq3.0，相比于原始 kafka，其擅长点出了原始的 log collecting 之外，还增加诸如 HA、事务等特性，使得从功能点上讲，可以替代传统大部分 MQ。

目前已经在阿里开始大规模应用，并且预计未来会逐渐在阿里取代 notify，meta 2.x 以前的所有队列。由于 rocketmq 还比较新，且官方没有给出相应的 benchmark。而 rocketmq 社区主要活跃者均是中国国内人员，因此不便于直接对比 rocketmq 和其他队列。在这里用 kafka 和其他队列进行对比，rocketmq 的性能比 kafka 还要好一些。

kafka rabbitmq benchmark

测试场景描述：

os： linux

cpu： 8core 2GHZ

memory：16GB

disks： 6 disks raid 10

network： 1Gbps

在各个对比中，均启动单个 producer，产生 1000 万条消息，单条消息大小为 200bytes；kafka 支持批量发送和接受，因此测试中分别批量值（batchsize）分别设置为 1 和 50，其中设置为 1 可以看成退化成传统 MQ。针对 Producer 其测试结果如下：

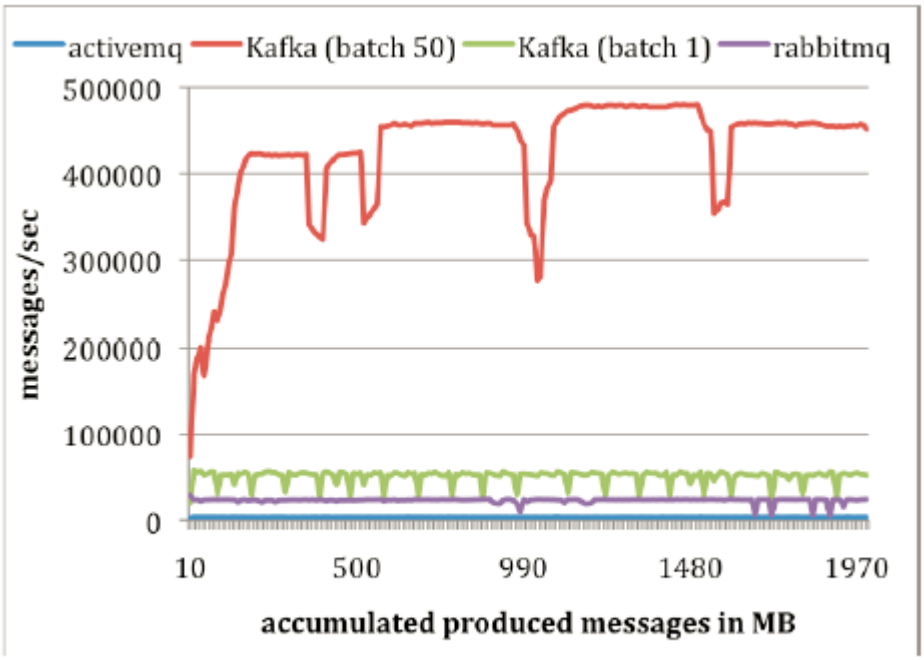


Figure 4. Producer Performance

针对 consumer：

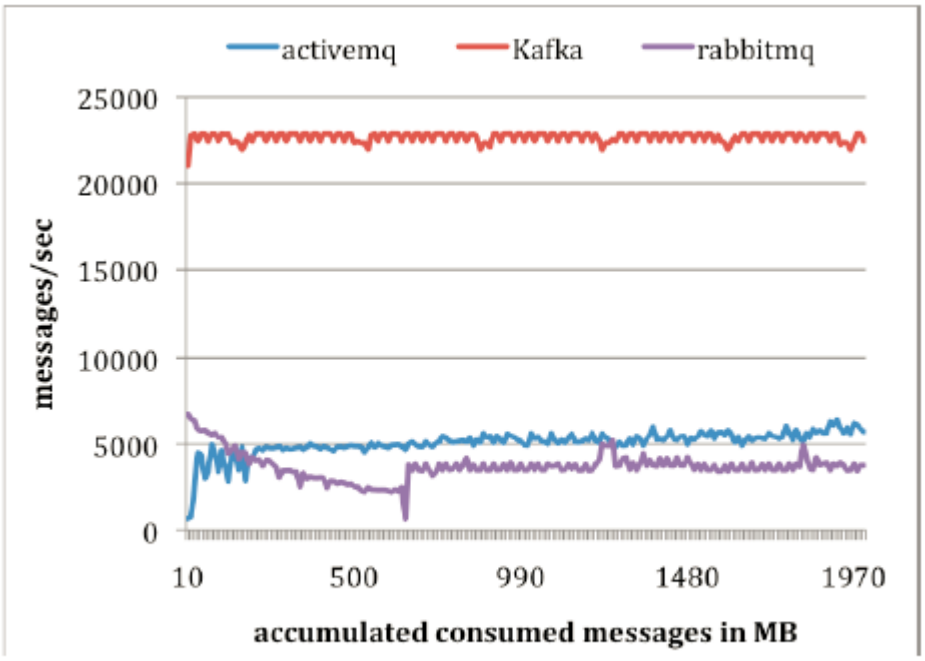


Figure 5. Consumer Performance

结论

开启批量模式时，kafka 的 produce 吞吐量极高。当关闭批量模式（batchsize=1）时，kafka 性能不亚于 rabbitmq 和 activemq。对于 consumer，由于 kafka 的设计使得 broker 不需要维护相应的 delivery state，并且数据的存储设计较好，因此吞吐量远远超过其他 MQ。当然，此 benchmark 是由 kafka 公布，其中 activemq、rabbitmq 并没有进行相应调优，并且只有单个 producer、consumer 的测试，刚好符合 kafka 的顺序读写特点，因此只能说具有一定参考意义。具体还是需要结合应用场景，进行相应的模拟测试，才能得出是否适应自己业务使用的结论。

kafka vs rabbitmq in detail

	rabbitmq	kafka	rocketmq
设计初衷	更多工作是保证消息递交； 认为 consumer 是一直处于 alive 状态去消费消息的； broker 设计的比较重，需要记录很多状态；	充分考虑消息堆积因素，认为 consumer 不一定处于 alive 状态； 考虑各个角色的分布式； 为追求吞吐量设计； broker 设计较轻，不保存 consumer 的消费状态	同 kafka； 且考虑了事务特性；
路由特性	提供了丰富的 routing，实现了 amqp 的 exchange、binding、queuing 等 model，queue、topic 等 routing 都支持；	topic routing only	topic routing only
集群特性	对应用透明的集群式实现；	虽然是集群式的实现，但是集群对 producer 并不是完全透明，producer 需要确认消息发送到哪一个	同 kafka

		partition；同时也利用这一点实现了消息的顺序递交特性（partition 内的消息是顺序的）；	
HA	通过 mirror queue 来支持（master/slave） master 提供服务，slave 仅备份	通过 replica 来支持 queue 的 master/slave 的可以自动切换（当 master 宕掉后）	支持，但需要人工切换 master 和 slave 的角色；
消息处理量	消息量~=20k/sec;	消息量>100k/sec	消息量>100k/sec
队列数目	<1000	支持上万队列	支持上万队列
顺序投递	不支持	支持	支持
事务性	不支持	不支持	支持

Kafka is a general purpose message broker, like RabbItMQ, with similar distributed deployment goals, but with very different assumptions on message model semantics. I would be skeptical of the "AMQP is more mature" argument and look at the facts of how either solution solves your problem.

a) Use Kafka if you have a fire hose of events (100k+/sec) you need delivered in partitioned order 'at least once' with a mix of online and batch consumers, you want to be able to re-read messages, you can deal with current limitations around node-level HA (or can use trunk code), and/or you don't mind supporting incubator-level software yourself via forums/IRC.

b) Use Rabbit if you have messages (20k+/sec) that need to be routed in complex ways to consumers, you want per-message delivery guarantees, you don't care about ordered delivery, you need HA at the cluster-node level now, and/or you need 24x7 paid support in addition to forums/IRC.

Neither offers great "filter/query" capabilities - if you need that, consider using **Storm** on top of one of these solutions to add computation, filtering, querying, on your streams. Or use something like **Cassandra** as your queryable cache. Kafka is also definitely not "mature" even though it is "production ready".

Details (caveat - my opinion, I've not used either in great anger, and I have more exposure to RabbitMQ)

Firstly, on RabbitMQ vs. Kafka. They are both excellent solutions, RabbitMQ being more mature, but both have very different design philosophies. Fundamentally, I'd say **RabbitMQ is broker-centric**, focused around delivery guarantees between producers and consumers, with transient preferred over durable messages. Whereas Kafka is **producer-centric**, based around partitioning a fire hose of event data into durable message brokers with cursors, supporting batch consumers that may be offline, or online consumers that want messages at low latency.

RabbitMQ uses the broker itself to maintain state of what's consumed (via message acknowledgements) - it uses Erlang's Mnesia to maintain delivery state around the broker cluster. Kafka doesn't have message acknowledgements, it assumes the consumer tracks of what's been consumed so far. Both Kafka brokers & consumers use Zookeeper to reliably maintain their state across a cluster.

RabbitMQ presumes that consumers are mostly online, and any messages "in wait" (persistent or not) are held opaquely (i.e. no cursor). RabbitMQ pre-2.0 (2010) would fall over if your consumers were too slow, but now it's robust for online and batch consumers - but clearly large amounts of persistent messages sitting in the broker was not the main design case for AMQP in general. Kafka was based from the beginning around both online and batch

consumers, and also has producer message batching - it's designed for holding and distributing large volumes of messages.

RabbitMQ provides rich routing capabilities with AMQP 0.9.1's exchange, binding and queuing model. Kafka has a very simple routing approach - in AMQP parlance it uses topic exchanges only.

Both solutions run as distributed clusters, but RabbitMQ's philosophy is to make the cluster transparent, as if it were a virtual broker. Kafka makes it explicit, by forcing the producer to know it is partitioning a topic's messages across several nodes, this has the benefit of**preserving ordered delivery** within a partition, which is richer than what RabbitMQ exposes, which is almost always unordered delivery (the AMQP 0.9.1 model says "one producer channel, one exchange, one queue, one consumer channel" is required for in-order delivery).

Put another way, Kafka presumes that producers generate a massive stream of events on their own timetable - there's no room for throttling producers because consumers are slow, since the data is too massive. The whole job of Kafka is to provide the "shock absorber" between the flood of events and those who want to consume them in their own way -- some online, others offline - only batch consuming on an hourly or even daily basis. Kafka can deliver "at least once" semantics per partition (since maintains delivery order), just like RabbitMQ, but it does it in a very different way.

Performance-wise, if you require ordered durable message delivery, currently it looks like there's no comparison: **Kafka currently blows away RabbitMQ in terms of performance on synthetic benchmarks**. This paper indicates 500,000 messages published per second and 22,000 messages consumed per second on a 2-node cluster with 6-disk RAID 10.
http://research.microsoft.com/en...

Of course this was written by the LinkedIn guys without necessarily expert RabbitMQ input, so YMMV.

Finally, a reminder: Kafka is an early Apache incubator project. It doesn't necessarily have all the hard-learned aspects in RabbitMQ.

Now, a word on AMQP. Frankly, it seems **the standard is a mess**. Officially there is a 1.0 proposed specification that is going through the OASIS standards process. In practice it is a forked standard, one (0.9.1) supported by vendors, the other (1.0) supported by the working group. A set of generally available, widely-adopted, production-quality AMQP 1.0 implementations across the major releases (Qpid from Redhat, RabbitMQ, etc.) won't exist until 2013, if ever.

As an external observer with no inside knowledge, here is what it looks like: the working group spent 5 years on a spec, from 2003 to 2008, culminating in a widely adopted release (0.9.1). Then a subset of more powerful working group members rewrote the spec by late 2011, completely shifting the focus of the spec from a messaging model to a transport protocol (sort of like TCP++), and declared it 1.0. So, we have the strange case where the "mature" AMQP is the non-standard 0.9.1 specification and the "immature" AMQP is the actual 1.0 standard.

This isn't to suggest 1.0 isn't good technology, it likely is, but that it's a much lower-level spec than AMQP intended to be for most of its published life, and is not widely supported yet beyond prototypes and one GA implementation that I know of (IIT SwiftMQ). The RabbitMQ folks have a prototype that has layers the 0.9.1 model on top of 1.0 but have not committed to a GA timeframe.

So, in my opinion, AMQP has lost some of its sheen, as while there's ample evidence it is interoperable from the

various connect-fests over the years, the standards politics have delayed the official standard and called into question its widespread support. On the bright side, one can argue that AMQP has already succeeded in its goal of helping to break the hold TIBCO had on high performance, low latency messaging through 2007 or so. Now there are many options. Bet on the broker you choose to use, and don't expect bug-free interoperability for a few years (if ever).
Just wanted to add a comment on Stuarts answer. Don't use rabbit if you need clustering for HA their clustering does not actually support network partitions. This is well explained in their documentation.

参考文献：

1. http://blog.x-aeon.com/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/
2. http://www.quora.com/RabbitMQ/RabbitMQ-vs-Kafka-which-one-for-durable-messaging-with-good-query-features
3. https://www.rabbitmq.com/tutorials/amqp-concepts.html
4. http://activemq.apache.org/version-5-getting-started.html
5. http://predic8.com/activemq-hornetq-rabbitmq-apollo-qpid-comparison.htm
6. paper: Kafka: a Distributed Messaging System for Log Processing
7. http://my.oschina.net/geecoodeer/blog/194829
8.