Kafka or RabbitMQ

Kafka

First, Kafka has stellar performance. It outperforms RabbitMQ and all other message brokers. Tests show up to *100,000 msg/sec* even on a single server, and it scales nicely as you add more hardware.

This is result of Kafka's design:

- messages published to a topic are distributed into partitions
- messages in partition represented as a log stream
- consumer is responsible for moving through this stream
- messages from each partition are processed in-order only

In general Kafka is optimized for work with "fast" consumers. It can work with "slow" consumers pretty well, however partition-centric design has some consiquences that make dealing with some critical situations difficult.

The major limitation is that each partition can have only one logical consumer (in the consumer group). So it means that if we are working with "slow" messages single issue (slow processing) blocks all other messages submitted to this partition after that message. Different strategies can be used to resolve this issues, one of them is to run another consumer group and sychronize it with existing consumer somehow to avoid duplicated processing of messages.

RabbitMQ

RabbitMQ is mature product, easy to use, supports a huge number of developement platforms. It's performance is not so impressive as Kafka's, some tests show about *20,000 msg/sec* on a single server. It also scales well when more servers added.

RabbitMQ is designed in a more classical way:

- messages published to queues (through exchange points)
- multiple consumers can connect to a queue
- message broker distribute messages across all available consumers
- message can be redelivered if consumer fails

- delivery order guaranteed for queues with single consumer (this is not possible when queue has multiple consumers)

So, we can say that this message broker can work well for “fast” consumers, if overall throughput is enough for your requirements.

And we have additional benefit in work with “slow” consumers. In the situation, when some consumer is stuck with some very slow processing, other messages will stuck in the queue as well unless we have other consumers connected. It means that by adding more consumers to the queue we are bypassing processing limitations. Usually this is a very simple operation, just run another process and RabbitMQ will take care about the rest.

Summary

Message queues is a broad and interesting topic, we just checked the tip of the iceberg here.

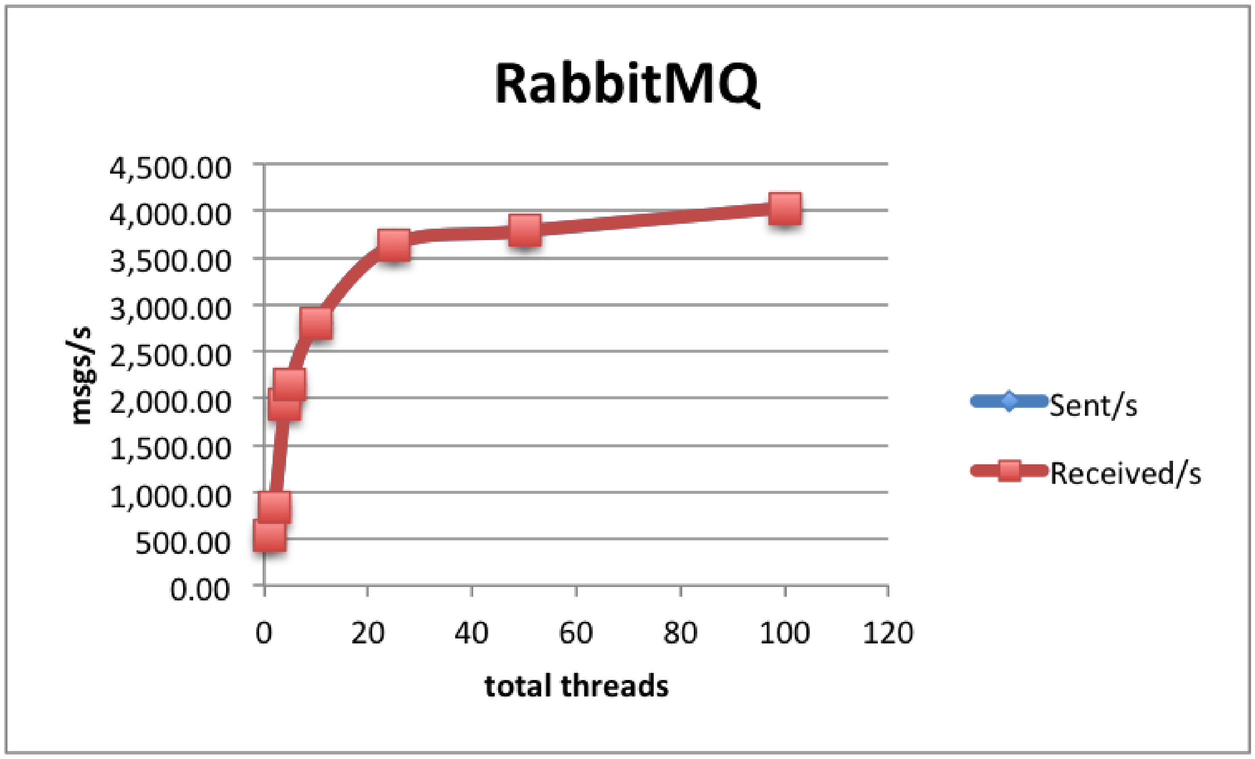
Conclusion is simple:

- Kafka is good for “fast” and reliable consumers
- RabbitMQ is good for “slow” and unreliable consumers

RabbitMQ:

A single-thread, single-node gives us 680 msgs/ssent&received, with a processing latency of 184 ms and send latency of 48 ms:

Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	680	680	**99**	48
1	5	2 154	2 148	107	**48**
1	25	3 844	3 844	122	66
2	1	844	843	109	
2	5	2 803	2 805	113	
2	25	3 780	3 784	141	
4	1	1 929	1 930	99	
4	5	3 674	3 673	126	
4	25	**4 331**	**4 330**	179	504

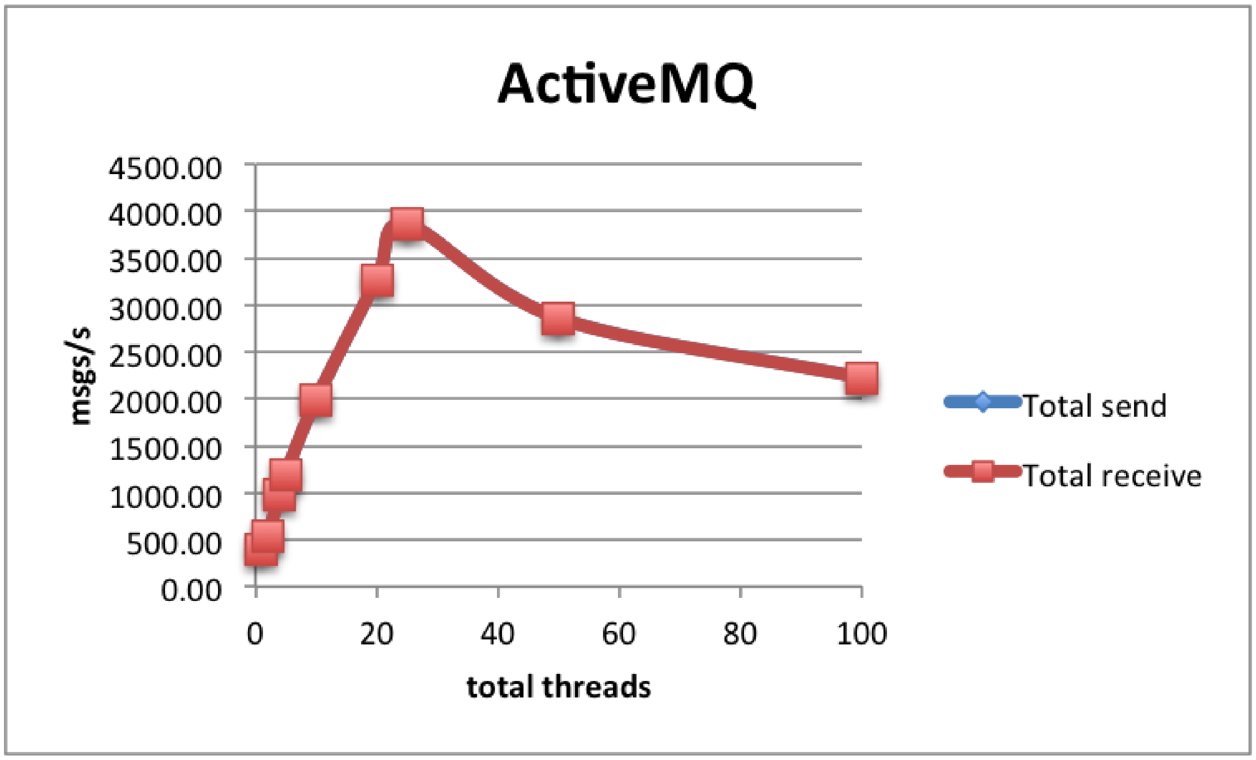


Active MQ:

Performance-wise, ActiveMQ does a bit worse than RabbitMQ, achieving at most 3 857 msgs/s with synchronous replication. This seems to be the maximum and is achieved with 1 node and 25 threads:

Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	395	395	**48**	
1	5	1 182	1 181	97	**48**
1	25	**3 857**	**3 855**	284	

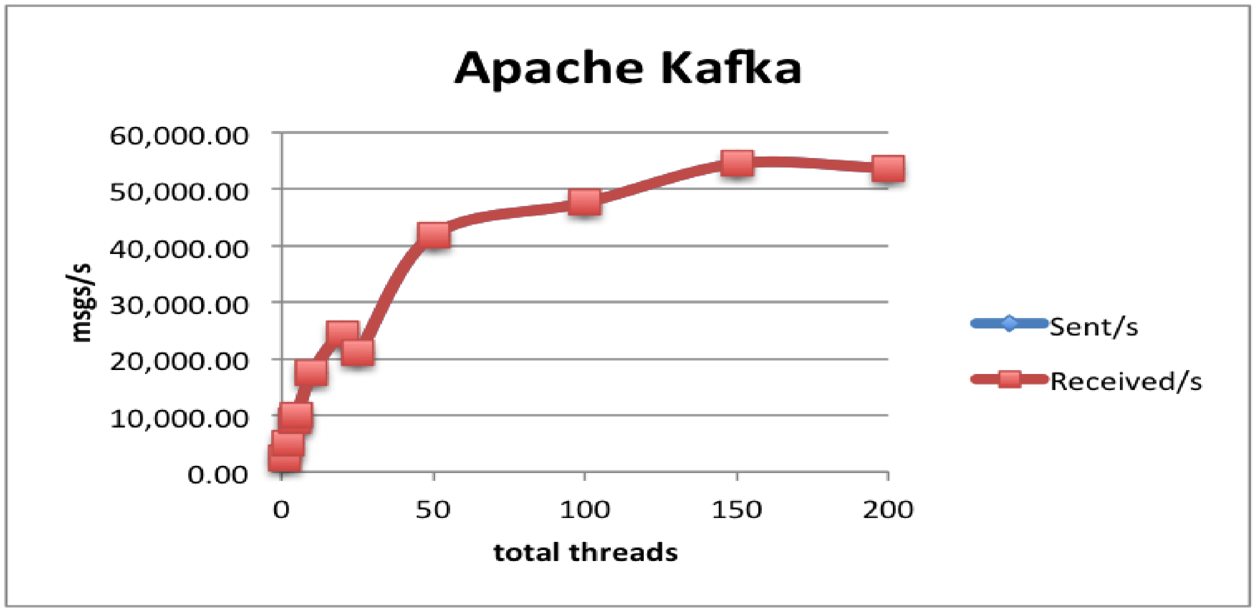
Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Notes
1	25	**3 857**	**3 855**	284	quorum_mem
2	25	2 862	2 869	778	quorum_mem
4	25	2 225	2 224	1071	quorum_mem
4	5	2 659	2 658	636	quorum_disk



Kafka:

Kafka's performance is great. With synchronous replication, a single-node single-thread achieves about **2 391 msgs/s**, and the best result is **54 494 msgs/s** with 25 sending&receiving threads and 6 client sender/receiver nodes.

Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	2 391	2 391	48	48
1	5	9 917	9 917	48	48
1	25	20 982	20 982	46	48
2	1	4 957	4 957	47	
2	5	17 470	17 470	47	
2	25	41 902	41 901	45	48
4	1	9 149	9 149	47	
4	5	24 381	24 381	47	48
4	25	47 617	47 618	47	48
6	25	**54 494**	**54 494**	**47**	**48**
8	25	53 696	53 697	47	48


Apache Kafka
msgs/s
total threads
Sent/s
Received/s