Message Queues

Pranav Tadepu

Synchronous vs Asynchronous Communication

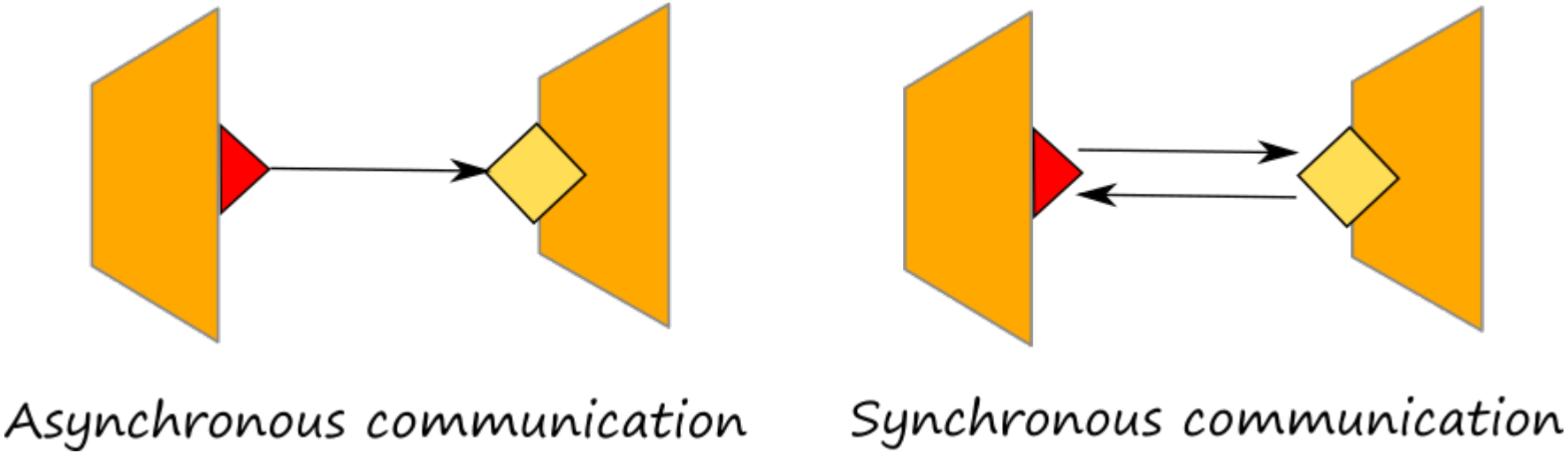


Fig 1. Synchronous vs Asynchronous Communication[5]

Introduction

- Message queues are a form of asynchronous communication in distributed systems, allowing messages to be stored and retrieved by different components of an application at different times.
- A message broker (also known as a message queue), which is essentially a kind of database that is optimized for handling message streams [1]

Messaging Domains:

- Point-To-Point Messaging
- Publish/Subscribe Messaging

Point-To-Point Messaging

- Message producers are called senders
- Message consumers are called receivers
- Exchange by means of destination called queue

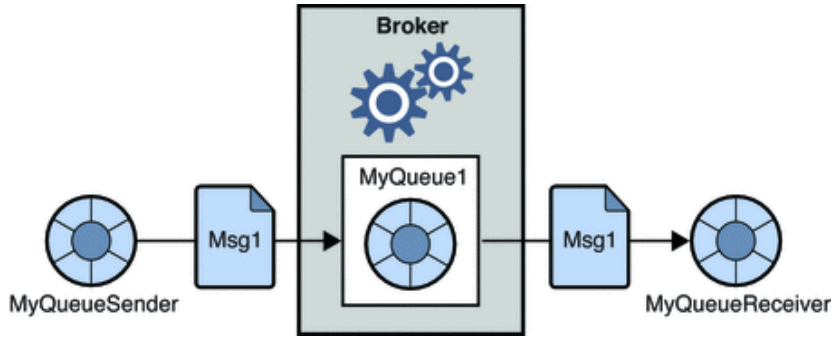


Fig 2. Simple Point-To-Point Messaging[4]

Publish/Subscribe Messaging

- Message producers are called publishers
- Message consumers are called subscribers
- They exchange by means of destination called a topic

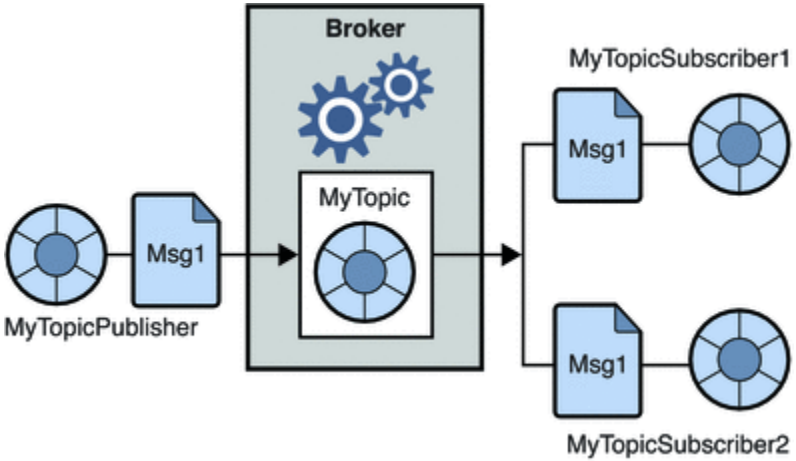


Fig 3. Simple Publish/Subscribe Messaging[4]

Advantage of Pub/Sub over Point-To-Point

- In Point-To-Point a message can be retrieved by a single consumer.
- In Pub/Sub a message can be subscribed by multiple consumers.
- Retention period

Choosing Message Queue

- Choosing the right message queue is critical for cost optimization, improved system performance, versatility, and scalability.[2]
- Message queues that are leading solutions:

  1.ActiveMQ Artemis

  2.RabbitMQ

  3.Apache Kafka

  4.Redis because although Redis can be used as a message broker, it is not primarily used for that[3]

Redis

- Usually used as database and cache[3]
- Pub/Sub feature to act as a message broker[3]
- PUBLISH command
- SUBSCRIBE command

ActiveMQ Artemis

- Open-source message broker, written in java[3]
- supports both publish-subscribe mode and point-to-point mode[3]
- built using a modular architecture
- Supports multiple messaging protocols including AMQP, MQTT, STOMP

Apache Kafka

- open-source distributed streaming platform[3]
- Uses peer to peer architecture, stores records in distributed and fault-tolerant manner

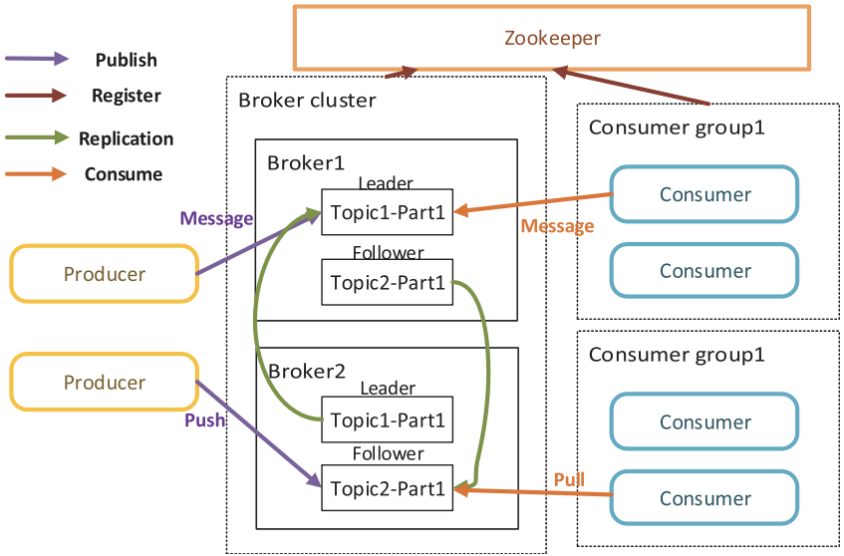


Fig 4. Kafka architecture[2]

RabbitMQ

- open-source message broker
- uses exchanges and queues to route messages between producers and consumers.[3]

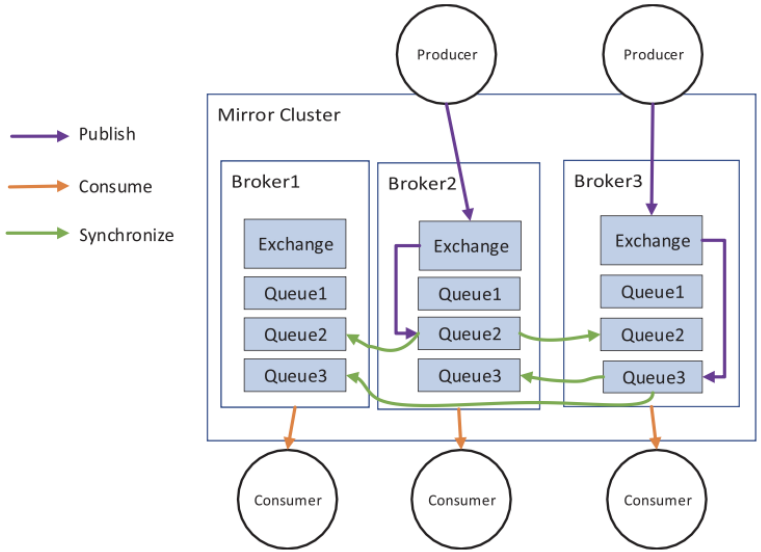


Fig 5. RabbitMQ architecture[2]

Methodology

- In a research they used a single producer and consumer
- OpenMessaging Benchmark tool by The Linux Foundation[2]
- Considered two metrics latency and throughput.
- Low latency means messages are delivered quickly[3]
- High throughput means system can handle large volumes of messages efficiently[3]

Flow diagram

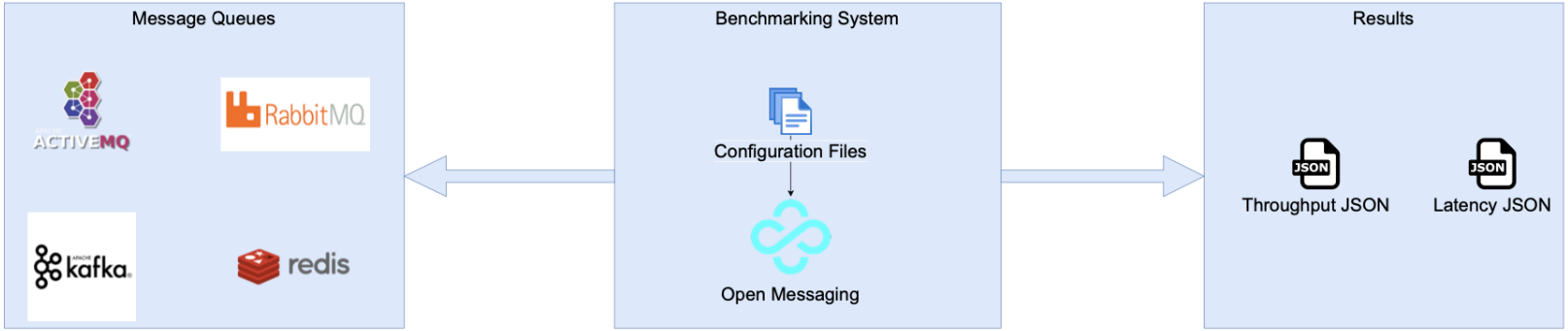


Fig 6. Flow diagram of methodology[3]

Metrics

Latency:

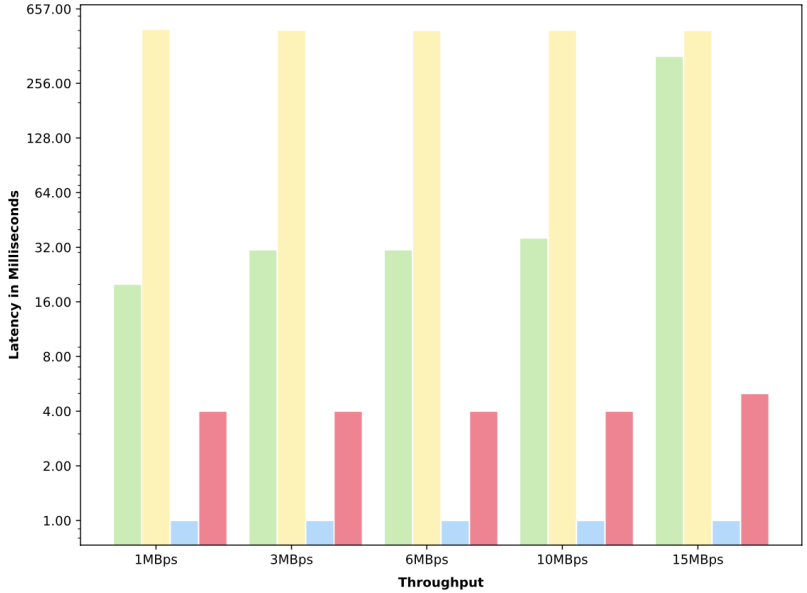
- 4 different forms of end-to-end latency is considered: 50th, 75th, 95th, 99th percentile[3]

Throughput:

- Throughput is considered in two different forms:

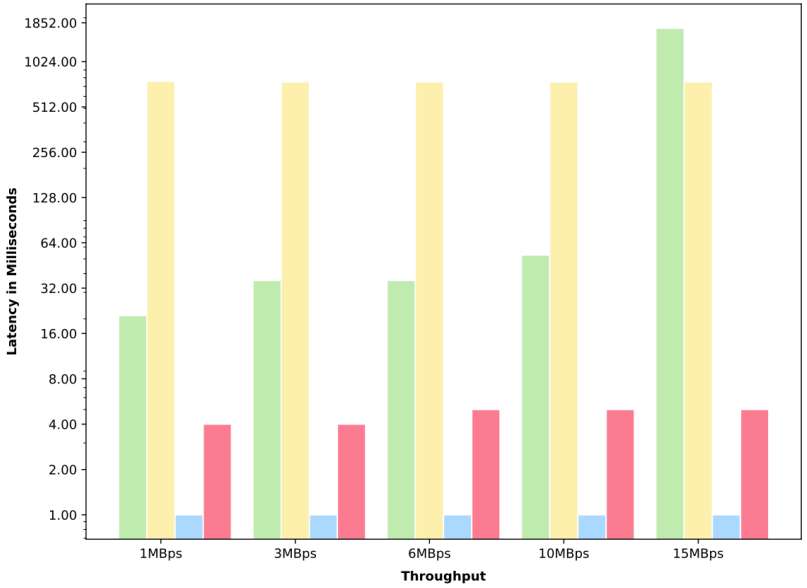
  1. Megabytes per second (MBps)

  2.Number of events per second[3]



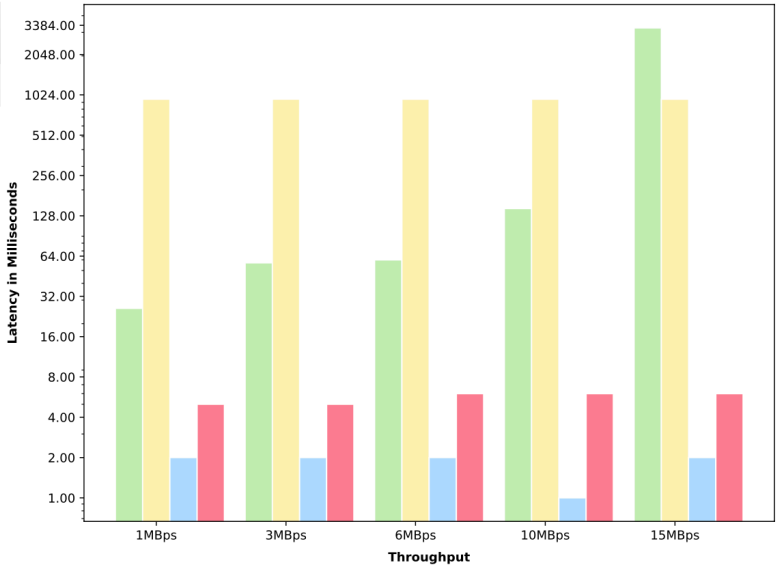
	1MBps	3MBps	6MBps	10MBps	15MBps
Artemis	20.0	31.0	31.0	36.0	360.0
RabbitMQ	507.0	501.0	500.0	501.0	500.0
Redis	1.0	1.0	1.0	1.0	1.0
Kafka	4.0	4.0	4.0	4.0	5.0

Fig 7. 50th percentile end-to-end latency



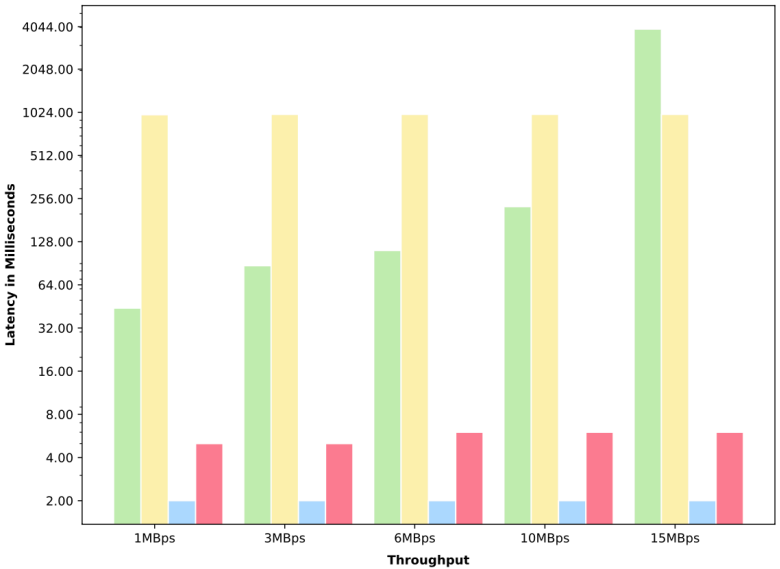
	1MBps	3MBps	6MBps	10MBps	15MBps
Artemis	21.0	36.0	36.0	53.0	1702.01
RabbitMQ	757.0	750.0	750.0	750.0	750.0
Redis	1.0	1.0	1.0	1.0	1.0
Kafka	4.0	4.0	5.0	5.0	5.0

Fig 8. 75th percentile end-to-end latency



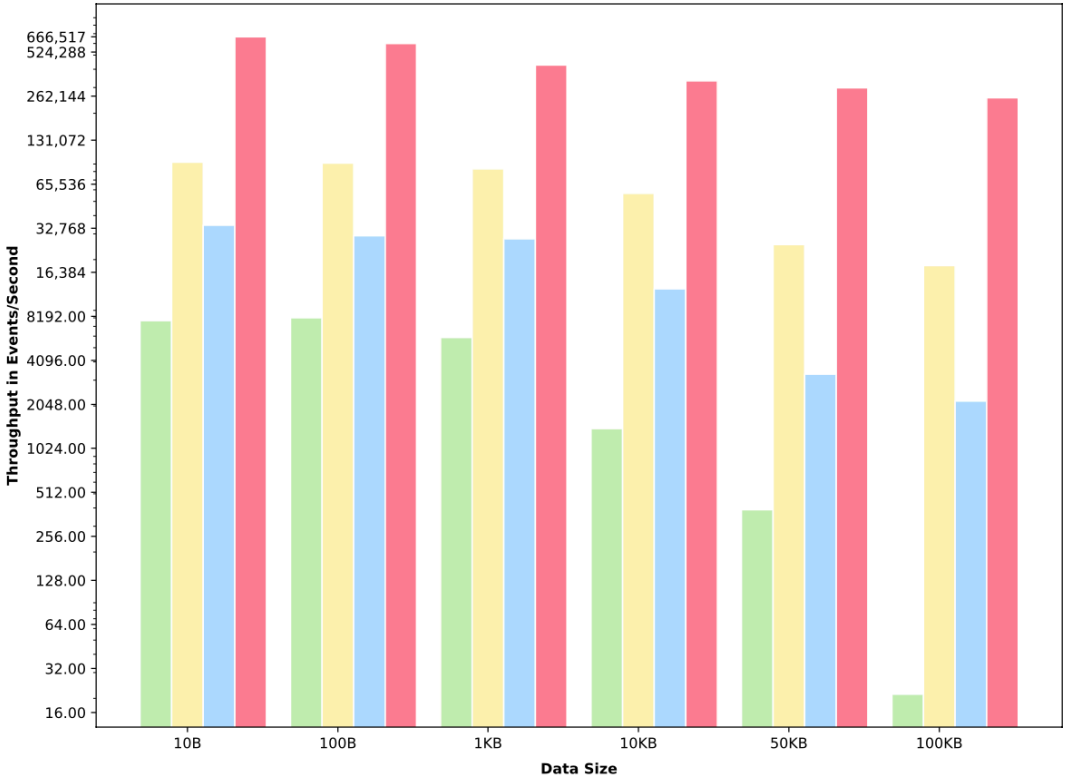
	1MBps	3MBps	6MBps	10MBps	15MBps
Artemis	26.0	57.0	60.0	145.0	3234.01
RabbitMQ	950.0	951.0	950.0	950.0	950.0
Redis	2.0	2.0	2.0	1.0	2.0
Kafka	5.0	5.0	6.0	6.0	6.0

Fig 9. The 95th percentile end-to-end latency.[3]



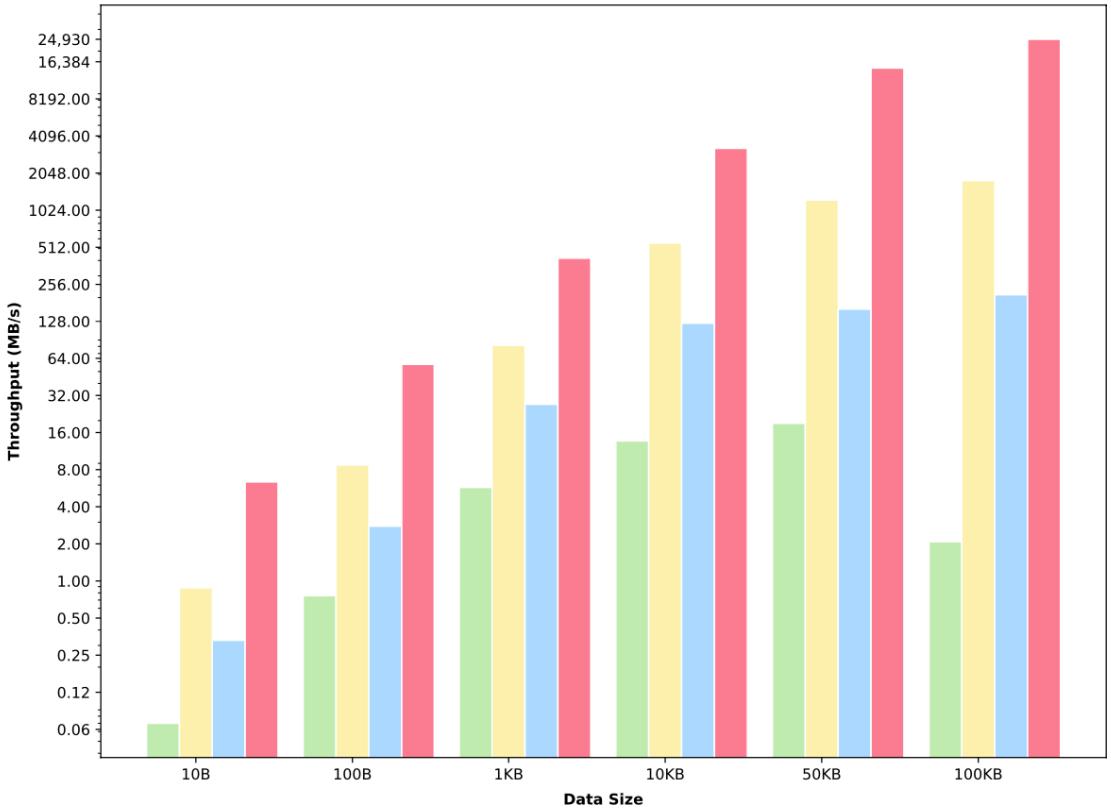
	1MBps	3MBps	6MBps	10MBps	15MBps
Artemis	44.0	87.0	111.0	225.0	3894.01
RabbitMQ	984.0	990.0	990.0	990.0	990.0
Redis	2.0	2.0	2.0	2.0	2.0
Kafka	5.0	5.0	6.0	6.0	6.0

Fig 10. The 99th percentile end-to-end latency.[3]



	10B	100B	1KB	10KB	50KB	100KB
Artemis	7652.99	8006.72	5872.98	1400.05	389.26	21.34
RabbitMQ	92,658	91,326	83,400	56,692	25,342	18,221
Redis	34,347	29,111	27,734	12,624	3298.67	2157.69
Kafka	666,517	599,794	427,361	333,145	298,590	255,285

Fig 11. Throughput in terms of events per second[3]



	10B	100B	1KB	10KB	50KB	100KB
Artemis	0.07	0.76	5.74	13.67	19.01	2.08
RabbitMQ	0.88	8.71	81.45	553.64	1237.41	1779.45
Redis	0.33	2.78	27.08	123.29	161.07	210.71
Kafka	6.36	57.2	417.34	3253.37	14,579	24,930

Fig 12. Throughput in terms of megabytes per second[3]

Results

- Redis is winner in terms of latency
- Apache Kafka had a significant higher throughput than Redis.[3]

Use case	Technology
High Throughput	Apache Kafka
Low Latency	Redis
Low Throughput and Low Latency	ActiveMQ Artemis, RabbitMQ
Low Latency and High Throughput	Apache Kafka

Sources

1. Jim N. Gray: “Queues Are Databases,” Microsoft Research Technical Report MSR-TR-95-56, December 1995.

2. G. Fu, Y. Zhang and G. Yu, "A Fair Comparison of Message Queuing Systems," in IEEE Access, vol. 9, pp. 421-432, 2021, doi: 10.1109/ACCESS.2020.3046503.

3. Maharjan, R.; Chy, M.S.H.; Arju, M.A.; Cerny, T. Benchmarking Message Queues. Telecom 2023, 4, 298–312. https://doi.org/10.3390/ telecom4020018

4. Client Programming Model, https://docs.oracle.com

5. Jolie Documentation, https://docs.jolie-lang.org/