Message-Oriented Middleware

Linh B. Ngo

March 11, 2016

Distributed Systems [1]

- Software systems (e.g., sensor systems) are distributed
  - Increasing scales
  - Across geographical and organizational boundaries

- How to scale up the communication infrastructure accordingly?

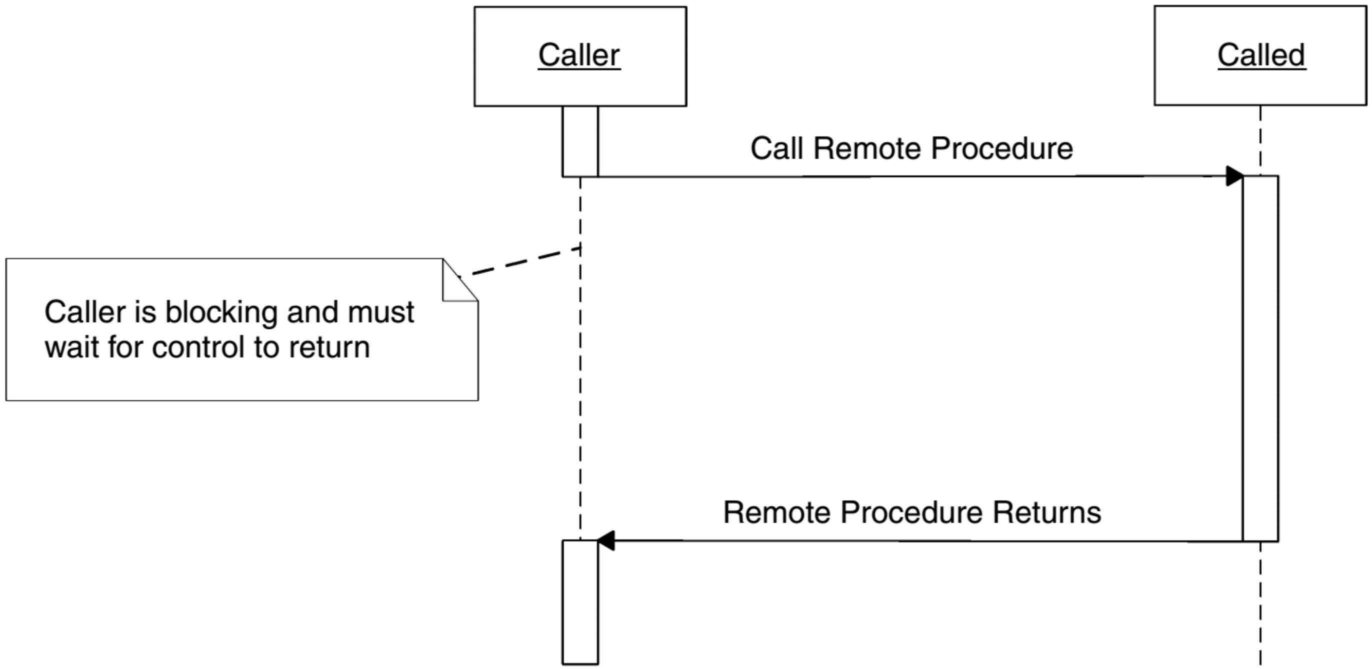
Challenges

- Complex environments
- Multiple programing languages
- Multiple hardware platforms (computing nodes and sensor platforms)
- Multiple operating systems
- Dynamic and flexible deployment
- High reliability, throughput, and resiliency
- High quality of service

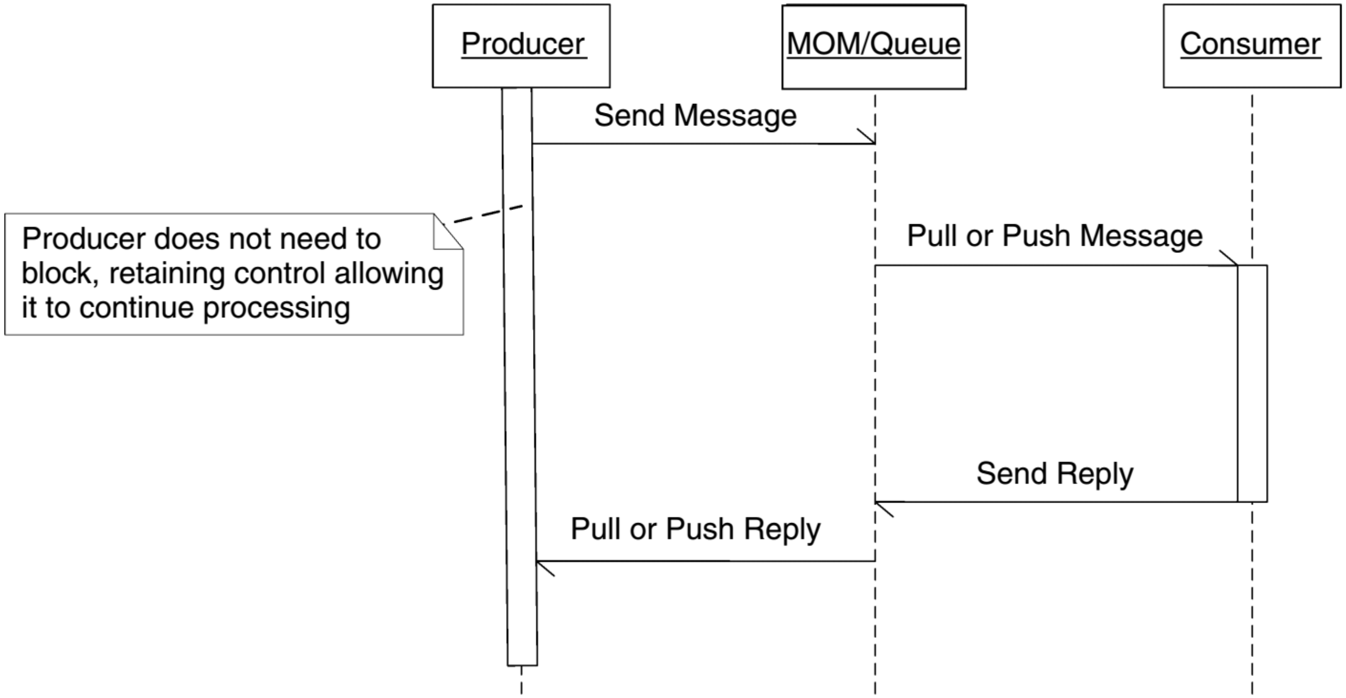
MOM: Message Oriented Middleware

- Alternative to RPC distribution mechanism
- Clean method for communication among disparate software entities
- Peer-to-peer relationship between individual clients (entities)

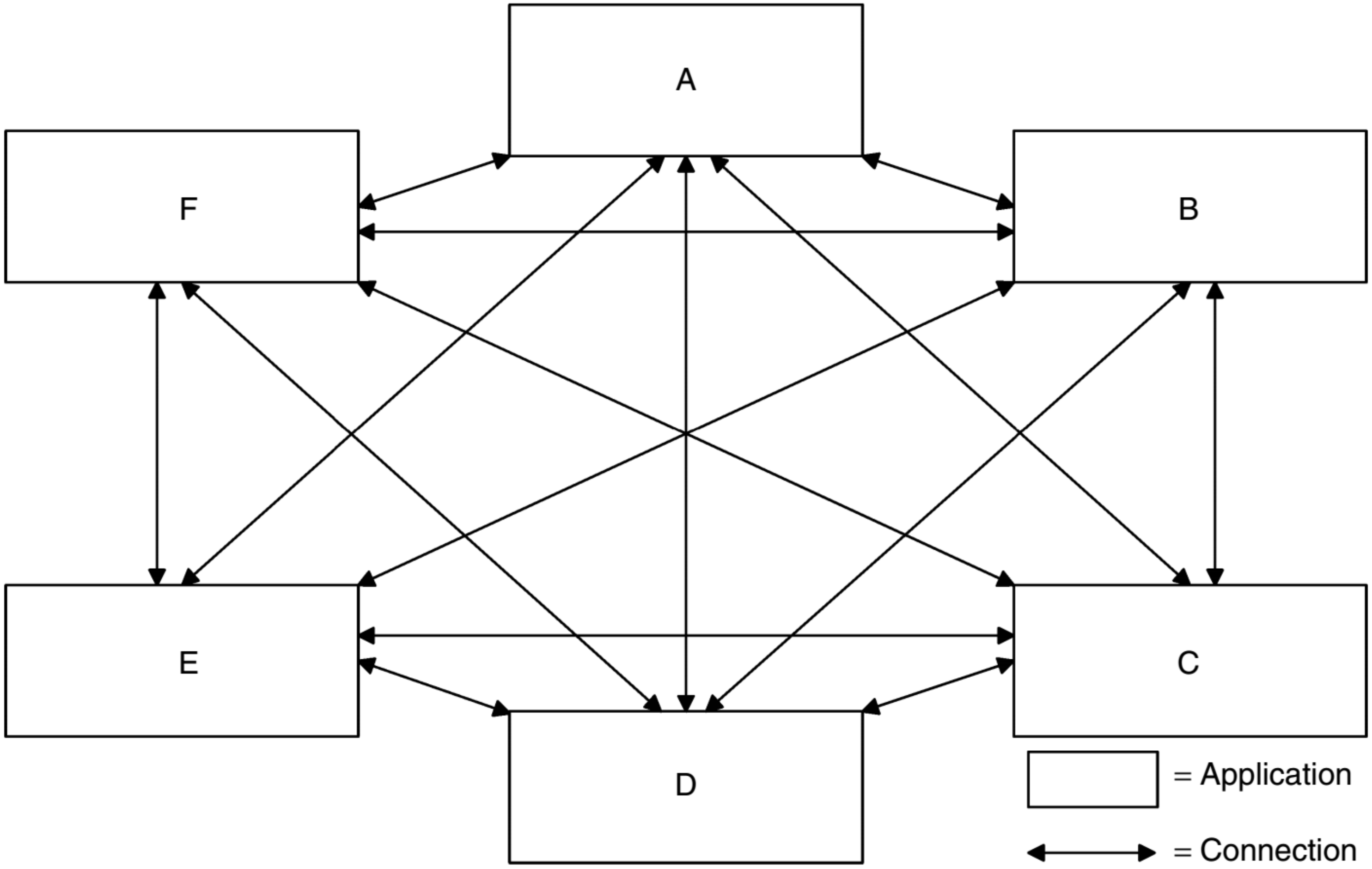
Synchronous



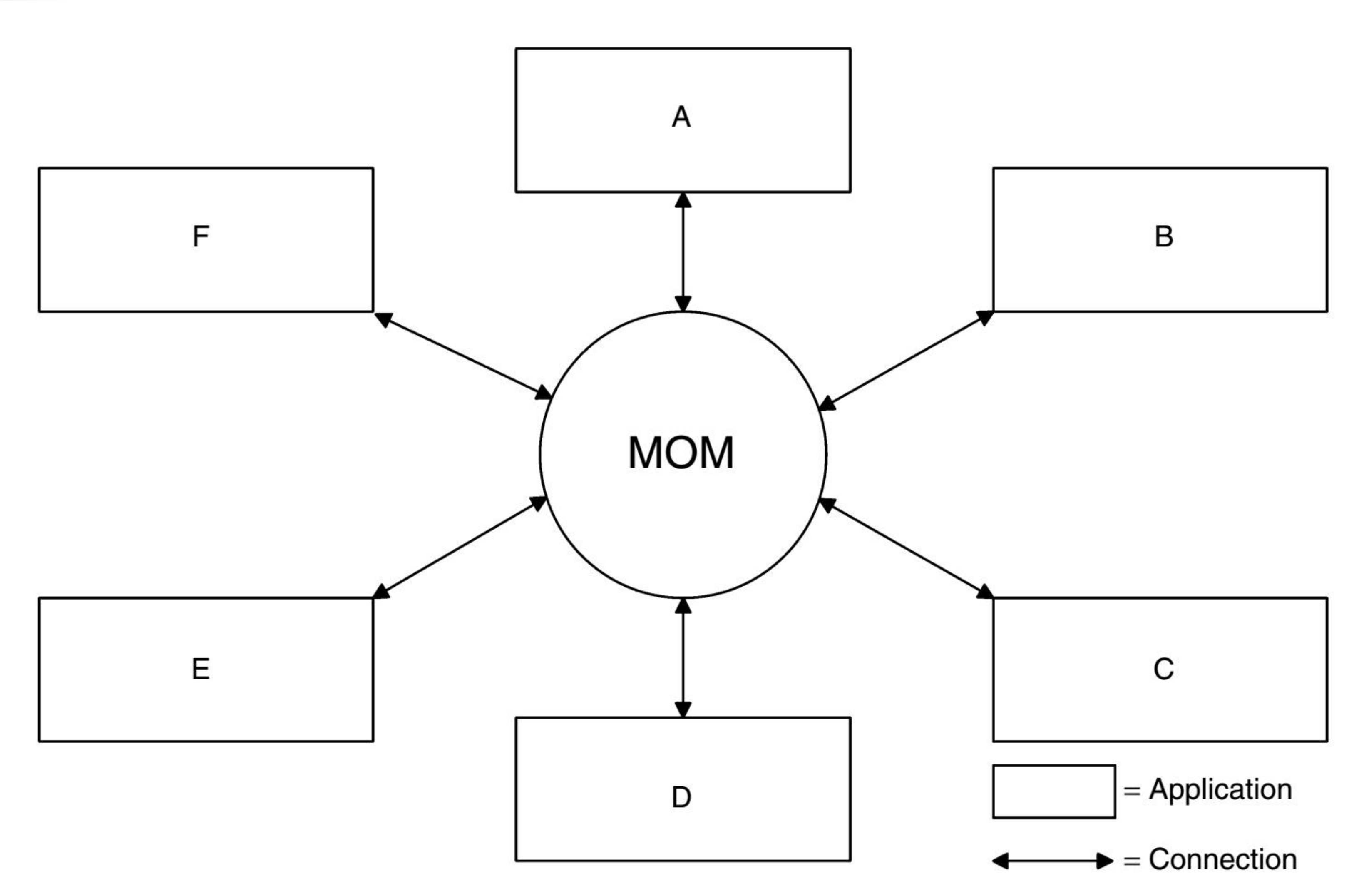
Asynchronous



Example RPC



Example MOM



RPC vs MOM: Coupling

- RPC is designed to work on object or function interfaces, resulting in tightly coupled systems

- The intermediate layer between senders and receivers in MOMs allows entities to exchange messages in a loosely coupled manner.

RPC vs MOM: Reliability

- Most RPC implementations provide little or no guaranteed reliable communication capability

- MOM relies on a store-and-forward mechanism to prevent message loss through network or system failure

RPC vs MOM: Scalability

- Due to its blocking nature, RPC can impact system performance when the participating components do not scale equally (bottleneck due to synchronization)

- As the subsystems are decoupled, MOM also decouple the performance dependency among them, resulting in the individual scaling ability.

RPC vs MOM: Availability

- The synchronous characteristics of RPC are often led to interdependency among subsystems. Consequently, the impact of individual component failures can be propagated across the entire system.

- Without a rigid coupling of subsystems (due to synchronous communication), MOM reduces failure propagation.

MOM: Message Queues

- Fundamental concept within MOM
- FIFO
- Customizable:
  - Queue Name
  - Queue Size
  - Saving Threshold
  - Message-sorting algorithm
  - …

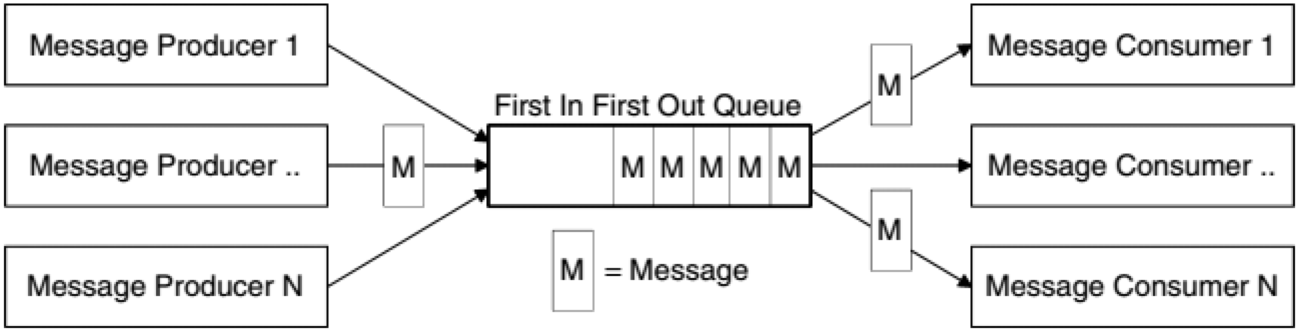
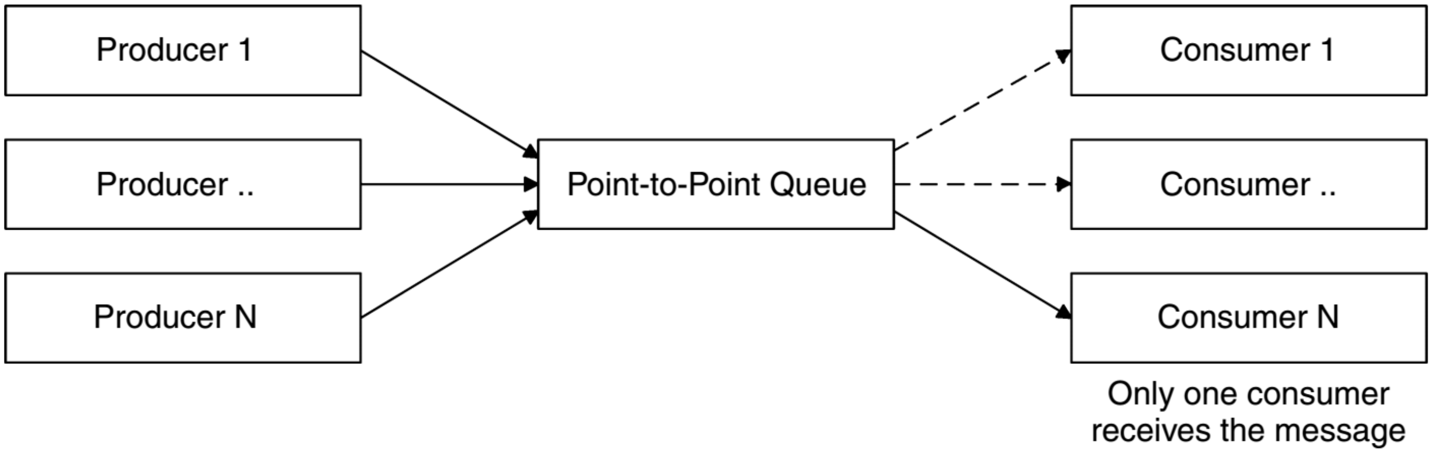
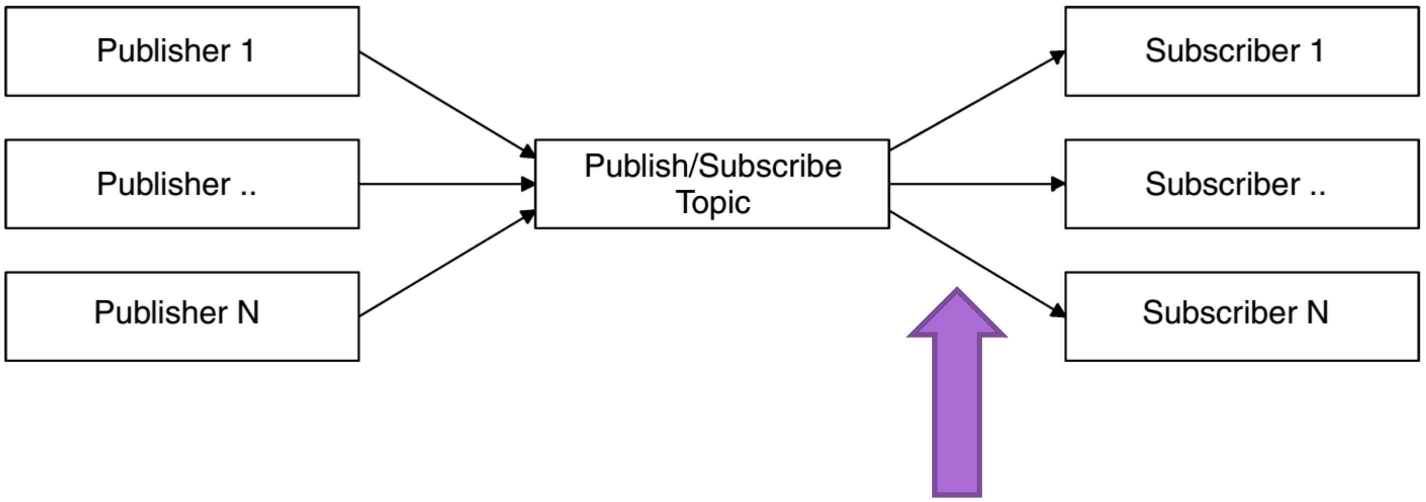


Figure 1.5 Message queue

Point-to-Point Messaging Model



Publish/Subscribe Model



Subscribers can **pull** from the Topic or have the messages **pushed** to themselves

Common MOM Standards

- Advanced Message Queueing Protocol (AMQP)
  - ISO standard
  - Application layer protocol
- MQ Telemetry Transport (MQTT)
  - ISO standard
  - Lightweight publish/subscribe messaging transport protocol on top of TCP/IP for M2M/IoT contexts
- Data Distribution Service
  - Message-oriented PubSub middleware standard interfacing with C++, C++11, C, Ada, Java, and Ruby
- Extensible Messaging and Presence Protocol (XMPP)
  - Message-oriented middleware based on XML.
  - Open standard

Open Source MOMs

- RabbitMQ
- Apache ActiveMQ
- ZeroMQ
- Kafka
- Apache Qpid
- IronMQ

- No in-depth performance comparison
- Different message/bandwidth characteristics call for different MOMs

RabbitMQ

- AMQP
- Erlang
- Broker architecture

Apache ActiveMQ

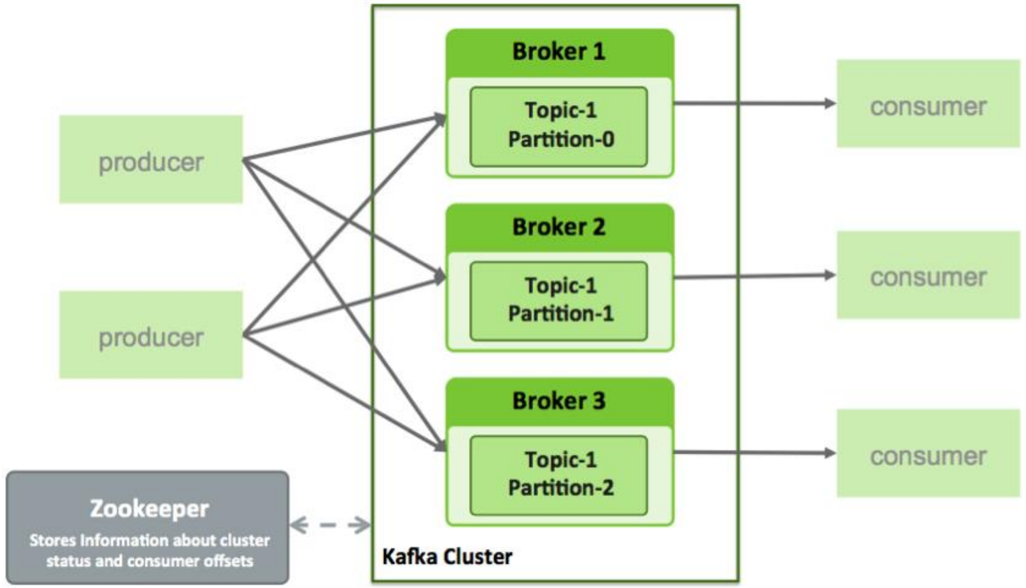
- AMQP, MQTT, Openwire, WebSockets …
- Java

ZeroMQ

- Very lightweight
- Require manual implementation
- ZMTP

Apache Kafka [2]

- Open-source message broker project (LinkedIn)
- Unified, high-throughput, low-latency platform to handle real-time data feeds
- A topic is divided into multiple partitions, and each broker stores one or more of these partitions
- Multiple producers/consumers can publish and retrieve messages at the same time



Apache Kafka: Efficiency on Single Partition

- Simple Storage
  - Each partition of a topic correspond to a logical log
  - A logical log is a set of segment files of same size (1GB)
  - Segment files are flushed to disk after a number of messages has been published (or after a determined duration)
  - Messages are exposed to consumers after they are flushed
  - No message id: message is addressed by its offset in the log
- Efficient Transfer
  - No application-level caching
  - Sequential data access
- Stateless Broker
  - Records of data consumed are not maintained
  - Data are purged from brokers after default length of 7 days

Apache Kafka: Distributed Coordination

- Smallest unit of parallelism: partition
  - Each partition can only be consumed by one consumer at any given time
  - More partitions than consumers
- No central master node – Zookeeper instead
  - Detecting the addition and removal of brokers and consumers
  - Triggering rebalancing in consumers
  - Maintaining consumption relationship and keep track of the consumed offset of ach partition

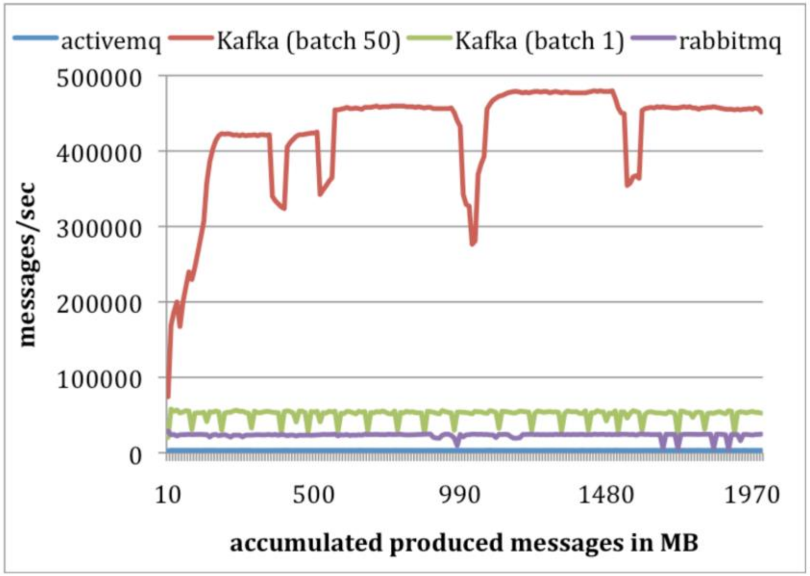
Apache Kafka: Delivery Guarantee

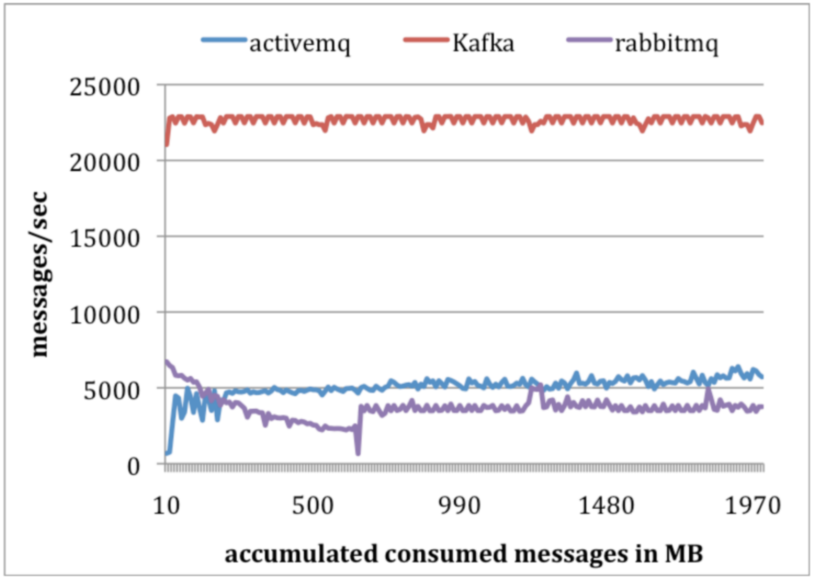
- At-least-once
- Most of the time is exactly-one
- Duplication happens when consumers crash and is revived.
- Messages from a single partition is guarantee to be delivered in-order

Apache Kafka: Performance Evaluation

- 2 machines: 8 2Ghz cores, 16GB RAM, 6 HDD (RAID 10)
- 1GB network
- *Note: No simultaneous producing/consuming test*
- Producer Test:
  - 10M messages, size 200 bytes
  - Batches of size 1 and 50
  - Results: 50K/s for size 1 and 400K/s for size 50
- Consumer Test:
  - 10M messages, size 200 bytes
  - Pull requests prefetch 1000 messages
  - Results: 22K/s

Apache Kafka: Performance Evaluation

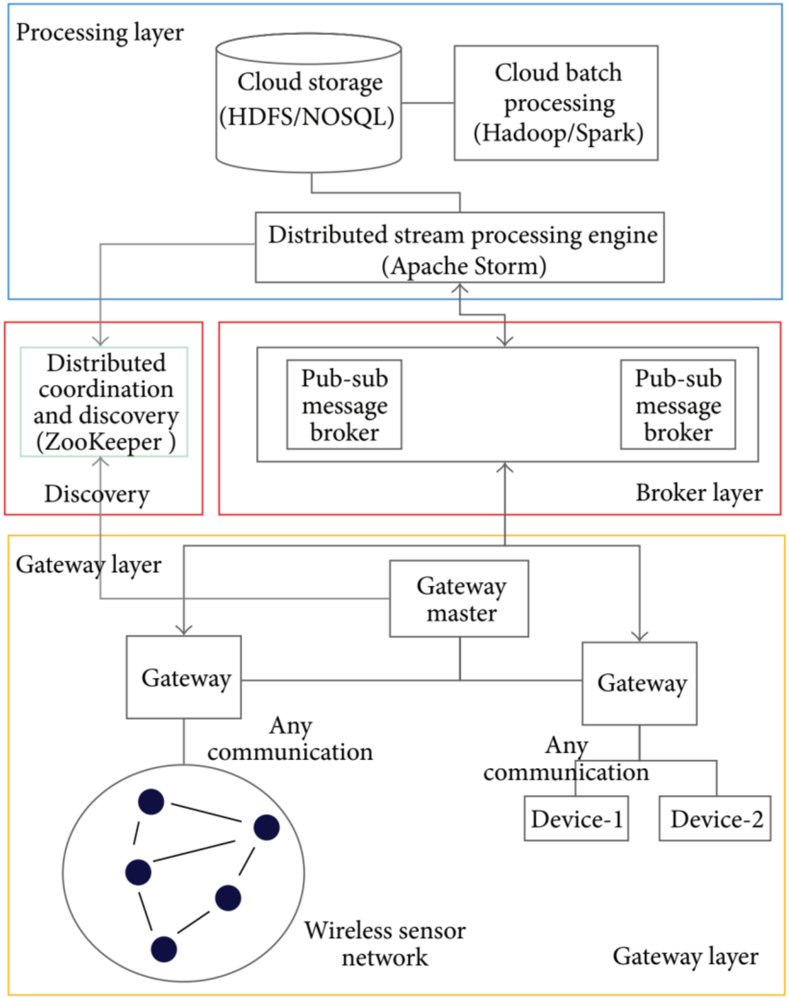




Apache Kafka: Performance Evaluation

- More efficient storage format
- Stateless broker
- sendfile API reduces overhead

Apache Kafka: Another Evaluation



Apache Kafka: Another Evaluation

- FutureGrid platform
- 8 virtual nodes: 2VCPUs, 4GBRAM, 40GB HDD
- 1 node: Storm Nimbus and Zookeeper
- 2 nodes: Gateway servers
- 3 nodes: Storm supervisors
- 2 nodes: Brokers

Apache Kafka: Another Evaluation [3]

- FutureGrid platform
- 8 virtual nodes: 2VCPUs, 4GBRAM, 40GB HDD
- 1 node: Storm Nimbus and Zookeeper
- 2 nodes: Gateway servers
- 3 nodes: Storm supervisors
- 2 nodes: Brokers
- 4 applications deployed on the two gateways producing data with constant rates:
  - Up to 100 messages per second
  - Up to 1MB in message size
  - Data are relayed into Storm, and then passed back to the gateway
- Timing measurements: roundtrip

Apache Kafka: Another Evaluation

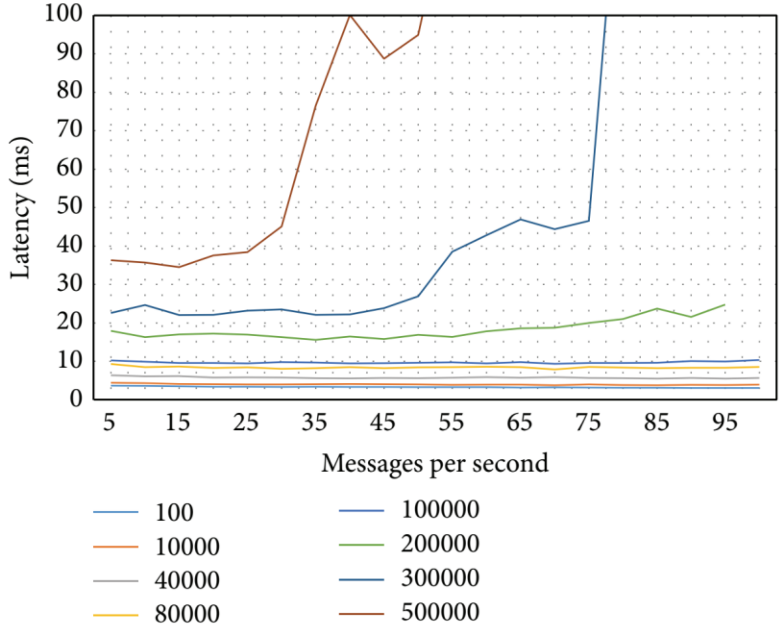


FIGURE 7: Average latency for different message sizes with RabbitMQ. The different lines are for different message sizes in bytes.

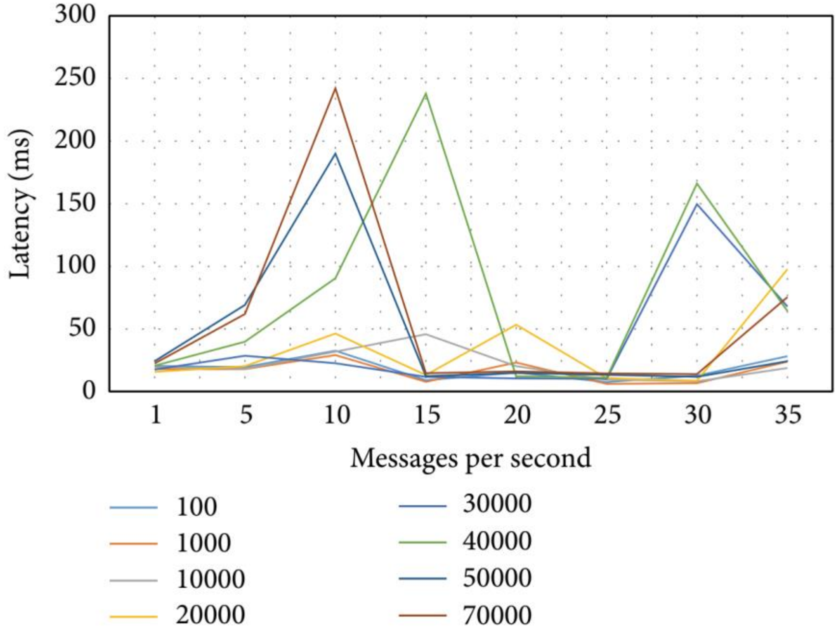


FIGURE 8: Average latency for different message sizes with Kafka. The different lines are for different message sizes in bytes.

Apache Kafka: Another Evaluation

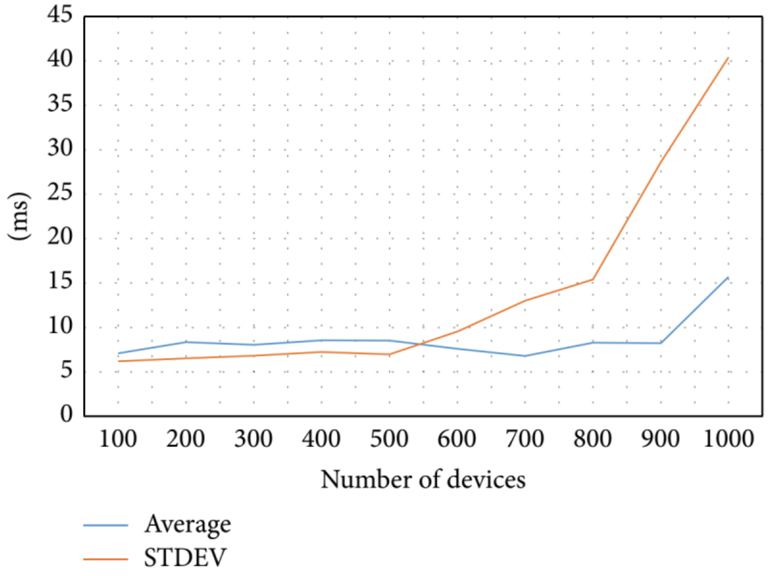


FIGURE 10: Latency with varying number of devices, RabbitMQ. The average latency and standard deviation are shown.

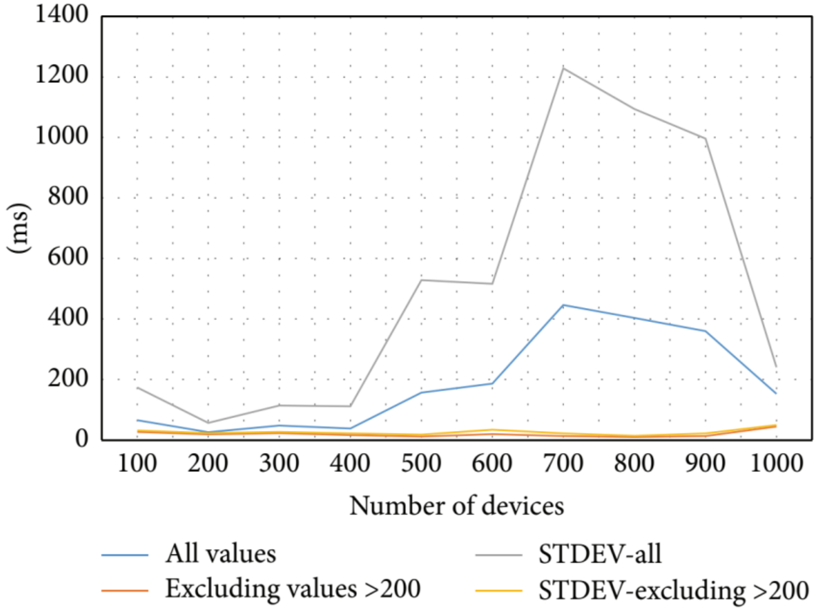


FIGURE 11: Latency with varying number of devices, Kafka. The average latency and standard deviation are shown. Also, averages calculated with omitting values over 200 are shown.

References

1. Curry, Edward. "Message-oriented middleware." Middleware for communications (2004): 1-28.
2. Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." NetDB, 2011
3. Kamburugamuve, Supun, Leif Christiansen, and Geoffrey Fox. "A framework for real time processing of sensor data in the cloud." Journal of Sensors 2015 (2015).