Message-oriented Middleware: A Review

Yongguo Jiang[+], Qiang Liu[+], Jian Su[+], Qianqian Liu[+], and Changshuai Qin[*+]

[+]DepartmentofComputerandTechnology,OceanUniversityofChina,Qingdao,P.R.China

***Abstract*—Message-oriented middleware (MOM) allows applications to communicate and exchange data by sending and receiving messages. With advantages of asynchronous and multi-point transmission, loosely coupling between participants etc. MOM is widely recognized as the most promising solution for communication between heterogeneous systems. This paper firstly reviews current research in MOM, then introduces main features and architecture of MOM, further elaborates on a few main functional modules, including message, queue, transmission mode and security, and finally carries out a comparative analysis of several commonly used MOM products. According to high-throughput and highly reliable data service needs for IoT and smart traffic etc. In this context, this paper analyzes opportunities and challenges to MOM when facing with new requirements and adapting to a more complicated environment.**

***Index Terms*—Message-oriented middleware(MOM), Message-queuing, Review, Subscribe/Publish, Distributed architectures**

I. Introduction

With the rapid development of information technology, traditional single application systems can not meet the actual business requirements, instead, multiple application systems are required to cooperate to solve the business problem [1]. More and more industrial and scientific applications integrate existing application systems to form distributed application systems [2]to meet business needs. However, most applications are distributed among heterogeneous hardware systems, using different operating systems as the running environment, and relying on different network architectures [3]. Complex heterogeneous environments make application integration very difficult. In order to reduce the integration cost of distributed applications, it is usually necessary to provide loosely coupled, shielded heterogeneous connection services through distributed components. Middleware technology is a type of technology that helps users to solve the heterogeneity and distribution problems.

MOM is a middleware technology consisting of a message delivery mechanism or a message queue pattern [4], which is often used to build complex distributed application systems. It simplifies data transfer between applications and shields the underlying heterogeneous operating systems and platforms. MOM Provides consistent communication and application development standards to ensure reliable, cross-platform information transfer and data exchange in distributed network environment. With the gradual maturity of large-scale distributed systems, MOM products are endless and have their own advantages.

Traditional MOM [5], [6]focuses on heterogeneous system decoupling, asynchronous messaging, and provides enterprise-level distributed application integration solutions that leverage its reliable transaction assurance mechanisms and security mechanisms to make it widely used in finance [7], cross-border e-commerce [8]and other fields. With the advent of the era of big data, LinkedIn designed and developed Kafka [9]to cope with the higher requirements for throughput of explosive log data. Alibaba designed and developed RocketMQ [8]in high-throughput, high-reliability applications. Facing with multi-tasking computing scenarios [10], MOM provides messaging services of high throughput. At the same time, in the IoT environment with high performance and network restrictions, there are a large number of heterogeneous data and devices, and it is necessary to provide more reliable and real-time MOM [11]. In recent years, in the fields of industrial Internet of Things [12], [13] smart transportation [14]–[16], smart city [17], [18], smart home [19], edge computer [20], smart grit [21]etc. MOM such as MQTT [22]ZeroMQ [23]provides powerful data transmission services and shielding heterogeneous connection services. However, for the basic structure of the existing MOM, some key issues still need to be further solved. In particular, in the new application scenario, more requirements are placed on MOM, and there is a broader research space.

This paper sorts out the main features and architecture of MOM based on the research literature and materials related to MOM in recent years, and then analyzes the main functional modules of MOM. It is explained from the aspects of message, queue, transmission mode, cluster management, security policy, transaction guarantee, user interface, log and monitoring module etc.. Then, sort out several common MOM products (JMS, Kafka, ZeroMQ, MQTT, AMQP, RocketMQ) by using CNKI database data, and compare their main features and application scenarios. Finally, this paper points out the problems and challenges in the current research in this field, and looks forward to future research hotspots and development directions.

II. MOM Overview

A. Basic Structure of MOM

MOM is a middleware system that enables various participants in a distributed system to send and receive messages. In a distributed system, MOM allows two or more applications to exchange data in the form of messages. Among them, a message is a packet including business data and control information. The participants in the system are divided into message producers and message consumers according to how the messages are used. As shown in Figure 1, the producer sends a message to the MOM by using the user interface that provided by MOM. After the message arrives at the

978-1-7281-4024-7/19/$31.00 ©2019 IEEE
DOI 10.1109/BIGCOM.2019.00023

middleware, it will be stored in the specified queue according to the control information in the message. The consumer also uses the user interface to send a name of message queue or subscription condition to the MOM to indicate the message information that the consumer needs. Producers and consumers are collectively referred to as clients, and each client is either a consumer or a producer. The MOM transmits the message to the client by using a peer-to-peer or publish-subscribe transport mode.

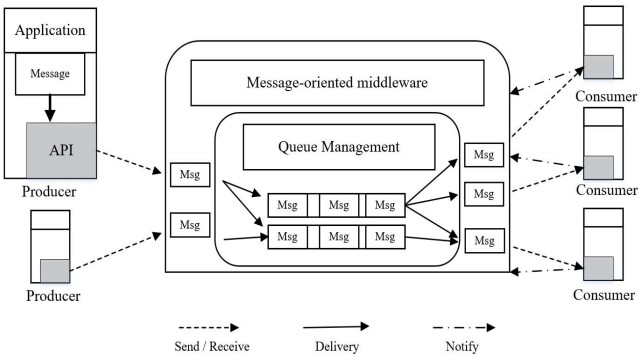

Fig. 1. Basic structure of message middleware.

A typical MOM system includes basic structures such as message protocol, queue and queue management, and delivery mode. At the same time, it has basic functional modules and facilities such as cluster management, security mechanism, transaction guarantee, log management, monitoring and user interface. Among them, the message protocol is the most important part of the whole MOM system. The control information in the message defines the basic functions of the MOM, and the business information contains the payload; Queue is a buffer entity that stores messages. MOM improves system scalability and reliability by managing queues. The delivery mode refers to the topology of messages exchanged between clients, defining the propagation path of messages in MOM; Cluster management provides extensibility for MOM, enabling cluster management of multiple MOM; Security mechanism provides a variety of security services to ensure data security for distributed systems; The transaction guarantee mechanism provides a "final consistency" guarantee for distributed systems; Logging, monitoring, and user interfaces provide applications with robust monitoring and flexible development interfaces.

B. Message

In MOM, the information exchanged between different applications(processes or threads) is collectively referred to as a message, which is the basic unit of data exchange and the basic element stored in the message queue [4].

1) Composition of Message: The message is divided into two parts: the message header and the message body [24].

- The message header defines the attributes of the message, and provides basic information and control information of the message for MOM, including message type, message unique identifier, message destination, message priority, message validity period, message size, user-defined attribute,etc..
- The message body contains the content of the message or the name of the file transmitted by the message, and is used by the application that parses the message [24].

2) Management of Message: The operation of MOM to manage messages in the queue is collectively referred to as message management.

a) Controlling the message life cycle:

- In the process of message entering queue, network sending, network receiving, and receiving from the queue, the message management mechanism can detect the validity period of the message and automatically discard the message whose expiration date expires. After the message enters the queue, the message will be discarded or disposed of when the failure occurs, and the message will end its lifetime.
- The message management mechanism has the function of controlling the validity period of the message, and can discard or process the expired message, release the occupied system resources, and ensure the running efficiency of the entire system. At the same time, the control mechanism of validity period should be unaffected by the inconsistent clocks of the sender and the receiver.

b) The message selector: The message management mechanism provides the function of a message selector that can filter out specific messages from a queue or topic. Each message has a priority attribute that identifies the relative importance of other messages. At the time of message delivery, the message management mechanism in MOM can set the priority statically or dynamically, and the high priority message can be prioritized. When a message is generated, each message belongs to a message management mechanism, and the message management mechanism parses and processes the message that receives according to the corresponding protocol.

c) The message compression: In many cases, the system bottleneck is not the problems of CPU or disk, but the network bandwidth. So data compression is very important. MOM provides the function of message compression, such as Kafka, which allows batch messages to be transmitted in compressed form.

C. Queue and queue management

A queue is a buffer entity that stores messages, which consists of a message index area and a data storage area. After the producer sends the message, the message is placed in the queue or directly obtained by the application, and the existence of the queue does not depend on the application.

The queue is established before the MOM receives the message. A queue can exist in the main storage area with temporary, on disk or similar auxiliary storage area for saving to prevent loss, or in two places (currently it is being used and saved for recovery).

Each queue belongs to a queue manager, and the queue manager puts the messages that it receives into the corresponding queue. Before using a queue, you must open the queue and specify what you want to do, such as browsing messages, receiving messages, placing messages in queues, querying queue properties, setting queue properties, and more. Queue managers can manage queues of related operations to ensure message order, message persistence, and message reliability.

a) Message ordering: In most usage scenarios, the processing order of data is important. The messages in the queue are already in order, so that the data is processed in a specific order.

b) Message persistence: Message persistence is the mechanism for converting program data into persistent and transient states. In layman's terms, instantaneous data (such as in-memory data, which is easily lost due to power loss, so that can't be permanently saved) is persisted with persistent data ,such as persistence to disk and can be stored for a long time. The guarantee of persistence is mainly determined by the storage of the queue. The persistent queue will be saved on disk, fixed and persistent storage, and the non-persistent queue will be stored in memory.

c) Message reliability: The reliability of the message mainly includes the reliability of the message at the sender, the reliability of the message delivery, the reliability of the message storage, and the expansion of the queue. The reliability guarantee is provided by the queue management module of the message storage and queue expansion.

- Reliability of message storage: A single MOM cluster becomes a master-slave node, and a slave node subscribes to all messages of the master node to perform message backup. This is an asynchronous operation, and the slave node receives less information than the master node branch message. In the synchronous replication mode, the message received by the master node is actively written to the slave node, and the success message is returned to the sender after receiving the response from the slave node.
- Expansion of the queue: When the queue is used, there is a problem that the initial storage space is insufficient. By expanding the server, the number of queues can be almost expanded arbitrarily. Each queue is persistent, whose length depends on the amount of disk space, and can be consumed anywhere in the queue.

Queue and queue management provide support for MOM to improve the reliability of message transmission. WANG Yan et al. [26] based on Kafka's low-level interface set, used the reliable coordination system (zookeeper) to construct the structure and mode of saving the message location identifier (offset), define the interface for the user to obtain data and the location identifier of the user to submit the message. The interface links the user's behavior of data consumption with the behavior of the message location identifier to ensure the reliability of the data and ensure that the message will never be lost, but may be transmitted repeatedly. This design solution solves the data quality problem caused by the Kafka native consumer's independence from the record of the user's consumption data and the data consumption location, thus ensuring the reliability of the data.

D. Message Delivery Mode

Message delivery mechanism is a kind of topology that implements the exchange of messages between clients. MOM utilizes efficient and reliable message delivery of platform-independent data exchanges. With message delivery, MOM can extend communication between processes in distributed environment. MOM provides message delivery modes point-to-point and publish/subscribe.

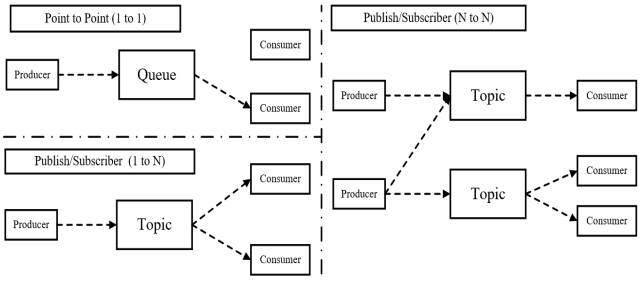

Fig. 2. Message delivery mode.

1) Point-to-Point(PTP): The point-to-point mode is a point-to-point communication mode between a message producer and a consumer. The message producer sends a message to a particular consumer identified by a certain name. There are two types of point-to-point, fire-and-forget processing and asynchronous request/response processing.

- Asynchronous dispatch processing is a process that the message producer sends a message to a queue. And it does not expect to receive a corresponding (at least not immediately received).
- Asynchronous request/response processing is a process that the message producer sends a message to the queue and then blocks the wait response queue, which is waiting for a response from the receiver.

2) Publish/Subscribe(Pub/Sub): The publish/subscribe mode is that the producer sends the message in the form of a topic. Consumers subscribe the required messages through a topic. The publish/subscribe mode makes the coupling relationship between producers and consumers more looser. The producer doesn't have to care about the destination address of the consumer, and the consumer does not have to care about the sending address of the message, but they only send or receive the message according to the subject of the message.

There are two types of publish/subscribe, durable and non-durable. Durable subscription is that messages wont disappear after the subscription relationship is established, regardless of whether the subscriber is online or not. Non-durable subscription is the subscriber who must be online in order to receive messages.

There are different research directions and research results of the problem of message delivery efficiency in the publish/subscribe mode, such as matching algorithms direction, and improvements to the publish/subscribe system.

a) Matching Algorithm: The publish/subscribe matching algorithm can efficiently find all subscription conditions that match a given event, improving the time efficiency of matching, space efficiency and efficiency of subscription maintenance [27]. Fu Ge et al. [28]proposed a solution to the on-demand subscription of message data for Kafka. They proposed the idea of designing a rule management for engine. The rule engine describes the rules through the rule description language. The rule tree is generated and stored in the rule memory by filtering certain fields and parsing the syntax description of the rule description. Regularize the data, select the middleware and data topics that the data needs to be placed from the rule tree to satisfy the filtering distribution of the message. This method solves the resource problem caused by data subscription and improves resource utilization.

b) Model improvement: As the number of subscribers and publishers increases, increasing messages can lead to architectural instability and can be problematic when the load is heavy. Wei et al. [29]proposed an improved method for publishing a subscription system, which changes the original message to be forwarded by the server and distributes part of the server's overload to the producer. After the improvement, the time required for the publish/subscribe server to send information to the subscriber is close to zero, which improves server performance while reducing costs. And in the publish/subscribe mode, updating the relationship between publishers and subscribers can be a difficult problem because they don't know each other, which will affect the performance of the system. Wei Li et al. [30]proposed a new clustering method based on the publish/subscribe system. This method can form a client cluster with strong communication relationship to ensure the redeployment of the client in the publish/subscribe system. This method will increase the number of messages and reduce the latency.

In addition, some people make efforts to improve the publish/subscribe system in specific areas. Michael Hoefling et al. [21]in the smart grid field, they proposed an adapter-based approach. The adapter which provides interfaces to the native publish/subscribe system and IEEE C37.118 communications. Connecting the adapter to the publish-subscribe architecture and integrate with the IEEE C37.118 communication entity to ensure that the publish/subscribe communication architecture can handle and forward IEEE C37.118 messages. This will increase the scalability of the number of communication users and will improve the ease of application development.

E. Cluster

A cluster is a collection of loosely coupled nodes, is a cluster. The cluster extends a single MOM to the environment of multiple MOM, and MOM cluster can provide distributed message service for customers. The cluster should provide a mechanism for receiving and processing messages distributed among several MOM systems, and supports two methods of preventing single point failure and load balancing.

1) Preventing single point failure: Preventing a single point of failure means that the entire system will not stop working due to the failure of a single node. When a subsystem in a MOM cluster fails, the cluster system can automatically select other subsystem to complete the message transmission.

2) Load balancing: The significance of load balancing is to make all nodes provide services for the outside world with the least cost and the best state which in order to improve the throughput and performance of the system, and reduce the response time of users. Moreover, load balancing enhances system reliability and minimizes single-node load. When the MOM system wants to spread the pressure of message processing on multiple systems or multiple queues, the load balancing function can be selected. After the message is sent to the cluster queue, the message is distributed among different queues according to the weight set by the system which to improve the efficiency of the system and provide better service.

Wang Zhenghe [31]proposed an optimized Kafka consumer/client load balancing algorithm. The load balancing process is completely controlled by the consumer as the manager. The rest of the consumers do not have to load balance separately, and the managers don't need to redistribute each consumer. Partition. The system monitors the health of all consumers, and the load can be rebalanced in time after a downtime consumer. The test results show that the algorithm can reduce the system overhead of Kafka consumer/client in the load balancing process and avoid the wrong load balancing result, which can effectively to guarantee the correctness of distributed scientific data processing.

When the instantaneous amount of data exceeds the system throughput, a reasonable load can be maintained by deleting part of the data, thereby enabling the system to perform normal processing. JiwonBang et al. [32]designed and implemented the Kafka load shedding engine to determine if load shedding is needed by the collected data.This implementation solves the problem of data starvation in Kafka where data is generated faster than consumption.

F. Security

1) Security service: The MOM system should provide security services at the network layer, channel layer and application layer. The network layer security service provides identity authentication between MOM nodes, the channel layer provides encryption and decryption of data packets at the transport layer, and the application layer provides full encryption and decryption of messages at the application layer.

The MOM system should provide a functional implementation of the default security service, such as providing secure management of the SSL protocol across the network layer and the channel layer. The MOM can provide customizable security function implementation, and should support the customization of the security interface of the message layer, network layer and transport layer:

- message layer interface, such as message encryption and compression interface.
- Network layer interfaces, such as pre-processing interfaces, link channel interfaces, and post-processing interfaces for establishing network connections.
- Transport layer interfaces, such as secure transport interfaces for network packets.

The investigation by Anton V. Uzunov et al. [33]shows that there are still many security issues in the MOM in addition to the existing security service functions. Taking the current increasingly popular publish/subscribe model as an example, systems that implement this model are inherently vulnerable to a variety of security threats. The survey proposes a specific publish/subscribe system security architecture, as well as a common system framework and stand-alone solutions to address one or more related security threats (message tampering, information disclosure, information abuse, etc.).

In order to improve the security performance of the MOM system and ensure the secure transmission of messages, Tom Goovaerts et al. [25]designed a higher security message bus architecture based on message interception and contract enhancement. Jinfu Wang et al. [34]proposed an anomaly detection system that can detect attacks and other types of failures that utilize messaging services. Cristian, Borcea et al. [35]proposed an end-to-end publish/subscribe information distribution system, PICADOR.These systems effectively protect each client in the message middleware domain and improve the security of the messaging middleware system.

In most applications, privacy is also a sensitive issue in MOM systems, and Giovanni, Di Crescenzo et al. [36]designed a new encryption tool "mixed" without sacrificing the required privacy attributes. The condition-independent transmission prototype -col" solves the problem of efficiency limitation and at the same time achieves ideal privacy protection. Protecting information confidentiality, especially the interests of subscribers, is an important issue when Brokers are located in untrusted areas (such as public clouds). Mohamed, Nabee et al. [37] proposed a novel content-based The routing decision encryption method, which allows the Broker to make routing decisions but does not disclose content with third-party providers, and resolves the confidentiality of published messages and the privacy of subscriptions. Emanuel Onica et al. [38]proposed a new technology that allows encrypted subscriptions to be updated directly at the Broker while maintaining privacy while maintaining the continuity and performance of the publish subscription service.

2) Message fault tolerance: In addition to security issues during network transmission, message recovery issues after a system crash are also important. MOM has the special ability of message fault tolerance to ensure that services are not interrupted in an environment where an element has an error. When an error occurs, the system has the ability to decide whether to continue the system without shutting down the data or to shut down the system to resume all ongoing processes. Fault tolerance usually involves a degree of redundancy. When the message fails, the system can be restored to the most recently submitted state. Once the message is restored, processing can be restarted and the aborted transaction can be committed again.

In a typical publish and subscribe system, single or multiple points of failure are unpredictable and avoidable. Therefore, it is especially important to resend messages to subscribers in the event of a failure. Reza Sherafat, Kazemzadeh et al. [39], [40]proposed a distributed publish/subscribe algorithm based on the tree topology. When the neighbor node fails, the topology mapping relationship between the nodes will be updated and the new one will be calculated in time. Forward the path, reconnect to the network topology, and send the message to the subscriber. The algorithm achieves high availability while ensuring service reliability.

For some specific areas, such as the banking system and the stock market, the packet loss rate and delay should be guaranteed to be within an allowable range, and rapid recovery of link or node failure is essential. Yue Jia et al. [41] proposed a new algorithm for pre-calculating the routing table and compressing it to save space, ensuring that the system can recover 100 percent from single link failure or single node failure.

G. Transaction guarantee

Transaction management refers to the operation of managing transactions from producer production to consumption by consumers. Transaction management is to ensure that messages arrive where they should go. At the same time, multiple messages transmission and reception can be included in one transaction, and the system guarantees that all message transmission and message reception operations in one transaction succeed at the same time, or fail at the same time. For KafKa, transactions guarantee atomic to write for each partition under the Kafka theme. For the RocketMQ adopted by Ali, the two-phase committed protocol ensures transaction consistency. It allows multiple distributed resources to participate in one transaction. The underlying transaction management is responsible for coordinating preparation, submission or rollback various resources for this transaction.

H. Log, monitor and user interface

1) Log: Log management is a tool for determining the cause of a failure or failure after a problem occurs. It also detects available problems and security threats in real time, providing a basis for automatic response problems and automatic policy configuration. It is used to record system information to facilitate users' understanding of the running status of the system and the location of the problem. The log can be used to check the running status of the system, and the errors in the system can be eliminated.

To prevent log files from taking up too much system resources, the system usually provides the function of log cleanup. For example, Kafka provides two log cleanup strategies, deletion and compression. Specify a cleanup strategy with specific parameters. Log cleanup can be controlled to

the topic level, and different cleanup strategies can be created for different topics.

2) Monitor: The MOM monitoring is used to view the running status of each middleware system in real time, dynamically modify the system configuration and related operating parameters and view the transmission status of each queue message. And start, stop or delete the monitored objects. Preventing the timeliness of troubleshooting and repairing problems in the event of a failure is not guaranteed. If the log of each server is checked to find and solve the problem, this method is time-consuming and labor-intensive. It is necessary to monitor various parameters of MOM. Different products have different monitoring methods. Kafka designs and develops a message monitoring system called Kafka Eagle for monitoring. ActiveMQ supports JMX monitoring. RabbitMQ and RocketMQ use the web monitoring system to monitor various properties in real time. Some MOM products provide a monitoring interface to facilitate the construction of independent monitoring systems. For example, RabbitMQ provides a RESTful HTTP API interface for obtaining real-time data of various business monitoring needs.

3) User interface: Interfaces are used to send messages between two applications or in a distributed system for asynchronous communication. In order to shield the heterogeneity of various development platforms and improve development efficiency, MOM supports a variety of programming languages, including C, C++, Java, VisualBasic, COBOL and more. It also supports a variety of popular development tools, such as WebSphere Studio Application Developer, PowerBuiler, Microsoft Visual C++, Visual Basic, Delphi and so on. MOM provides a unified application development interface on different platforms, and only needs to be recompiled to realize the migration of the application on different platforms. The application obtains various functions provided by the MOM by calling the API interface.

III. Introduction and Comparison of Common MOM

A. MOM Qualitative comparison

Table 1 analyzes and compares well-known MOM products and protocols from the MOM messages, queues, delivery modes, clusters, security mechanisms, user interfaces, and support for transactions. As can be seen from the table, most systems and protocols support the transmission of structured message and byte stream message. In addition, as to ZeroMQ and MQTT, message persistence, transaction control and load balancing are supported. All message middleware products support the delivery mode of the publish-subscribe model, and most of them support multiple security mechanisms.

B. Activity analysis of MOM

In the CNKI full-text database, we selected MOM and the news protocol related documents selected by CNKI from 2000 to 2018 as research samples. According to the specific time of publication, the starting year has changed. The literature time series situation map can clearly and intuitively show the changes in the number of related documents in each time period. According to the number of articles issued in each year, the distribution of time series of MOM and message protocol documents in China is shown, as is shown in Figures 3 and 4.

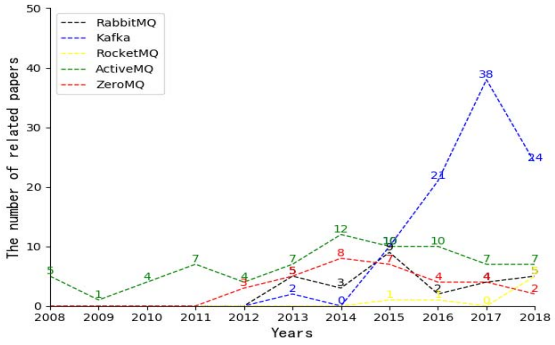

Fig. 3. Time series distribution of multiple message middleware documents.

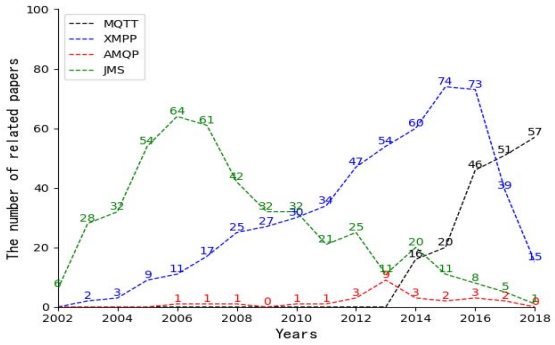

Fig. 4. Time series distribution of multiple message protocol documents.

It can be seen that the related research on ActiveMQ and JMS appeared earlier, and a large amount of relevant literature appeared between 2003 and 2010. However, with the emergence of new application scenarios such as big data and the Internet of Things, ActiveMQ's problems in single-node performance and throughput bottlenecks are gradually revealed, and is gradually replaced by emerging products such as Kafka and MQTT.

XMPP completed its standardization work in 2004 and it is mainly used in the field of instant messaging. Social media such as ICQ, Facebook and Twitter use the XMPP protocol for instant messaging. However, with the emergence of a large number of mobile devices, XMPP messages have become more and more disadvantageous due to high redundancy (based on XML) and high energy consumption. As can be seen in Figure 4, the research heat of XMPP dropped significantly after 2016.

Thanks to Kafka's outstanding performance in high throughput, Kafka has become one of the darlings of the big data era. Therefore, LinkedIn has been in a more popular position after it has been open sourced. The RocketMQ project has been

TABLE I
COMPARISON OF SOME MESSAGE-ORIENTED MIDDLEWARE

Features \ MOM		Kafka	RabbitMQ	ZeroMQ	RocketMQ	ActiveMQ	AMQP	MQTT
Message	Supported formats	String, Byte	Map, String	String, Byte	String, Byte	String, Map, Byte, Object	Integers, Bits, String, Field tables	Byte
	Supported protocols	Kafka	AMQP, STOMP, MQTT, XMPP	TCP, UDP	RocketMQ	OpenWire, STOMP, REST, XMPP, AMQP	-	-
Queue	Orderliness of consumption	yes	yes	yes	yes	yes	yes	yes
	Persistence	yes	yes	no	yes	yes	yes	no
Message delivery mode	Pub/Sub	yes	yes	yes	yes	yes	yes	yes
	Point-to-Point	-	yes	yes	yes	yes	yes	-
Cluster	Cluster solution	Natural stateless cluster	Support for simple clusters	no	Natural stateless cluster	Support for simple clusters	Support	Support
	Load balancing	yes	yes	no	yes	yes	yes	no
Security	Encryption Algorithm	SSL	TLS	CurveZMQ over TCP	TLS	TLS	TLS	TLS
	Authentication	SASL	SASL	SASL	-	JAAS	SASL	SASL
Transa-ction	Whether to support the transaction	yes	yes	no	yes	yes	yes	no
User API	Supported development language	Java, .NET, PHP, Python, Erlang, Perl, JavaScript, Ruby, Go	Java, .NET, PHP, Python, JavaScript, Ruby, Go	C, C++, JAVA, .NET, Python	Java, C++, Go	Java, C, C++, C#, Ruby, Perl, Python, PHP	C++, Ruby, Java, JMS, Python, .NET	PHP, JAVA, Python, C, C#

open sourced late and has not been researched in academia. However, because it supports world-class distributed systems in Alibaba and effectively solves trillions of data floods, RocketMQ has become the industry's hottest MOM product.

MQTT appeared at the latest, but the current number of posts has far exceeded other protocol specifications. Due to the lightweight nature of MQTT and the excellent performance under the scene of large network fluctuations, MQTT has been researched and developed rapidly in the field of Internet of Things, and has become a popular choice in the mobile and IoT fields. ZeroMQ, which also shines in the field of Internet of Things, shows that there is continuous research literature after 2011.

RabbitMQ is an open source MOM based on the AMQP protocol. AMQP became the international standard for ISO and IEC in 2014 and is a typical application layer protocol. As can be seen from Figure 4, a lot of attention has been gained during the years of AMQP standardization work (2012-2014).

C. Contrast and application of MOM

1) MQTT: MQTT [22] is a "lightweight" communication protocol based on the publish/subscribe model, which is built on the TCP/IP protocol and is originally designed to enable stable communication for low-power devices. Its biggest advantage is that it can provide real-time and reliable message service for connecting remote devices with very little code and limited bandwidth. It is widely used in the fields of Internet of Things, small devices, mobile applications and so on. However, MQTT performs generally in throughput performance and is not suitable for transmission of large message payloads.

2) Kafka: Kafka [42] is LinkedIn's open source distributed publish/subscribe messaging system.The main feature

of Kafka is the Pull-based model that handles message consumption and has high throughput. Therefore Kafka is suitable for data collection operations for Internet services that generate large amounts of data. At the same time, multiple copies of files are used for data storage to achieve high system availability. However, poor support for message reliability is prone to message loss and message duplication.

3) RabbitMQ: RabbitMQ [43] is an open source messaging system that implements Advanced Message Queuing (AMQP). Among them, the AMQP protocol is widely used in enterprise systems and is suitable for scenarios with high data consistency, stability, and reliability requirements. RabbitMQ uses the Erlang language for development, which improves the concurrency of the system, but also increases the difficulty of custom development. In general, RabbitMQ has a relatively complete MOM function, which guarantees a high level of performance, and is the first choice for enterprise system development.

4) ActiveMQ: ActiveMQ [6] is designed to provide standard, message-oriented, messaging communication middleware that spans multiple languages and systems. AActiveMQ supports a variety of user languages including Java, C, C++, etc.. It provides a complete and flexible functional module and development interface, which is conducive for customization development, and is widely used in various enterprise applications. Because ActiveMQ kernels were developed earlier, supporters are now focusing on the next generation of products and the maintenance of the products is not guaranteed.

5) RocketMQ: RocketMQ [8] is targeted at non-logged, reliable messaging for large-scale distributed environments. From a performance perspective, the RocketMQ single broker node delivers outstanding performance and supports distributed expansion design with high throughput and high availability. However, RocketMQ only has Java, C++, and Go development interfaces. Currently RocketMQ is widely used in Alibaba for orders, transactions, stream computing, message push, log streaming, binglog distribution and other scenarios.

6) ZeroMQ: ZeroMQ [23] is a multi-threaded network library based on message queues developed specifically for high throughput and low latency scenarios through agentless design. Therefore, ZeroMQ focuses on real-time data communication scenarios and is often used in scenarios such as finance and Internet of Things. However, ZeroMQ has high development costs and requires customized development based on application requirements. Therefore, as far as the difficulty of operation is concerned, ZeroMQ is the most inappropriate choice. But if performance is the primary goal, ZeroMQ is the best choice.

IV. Research hotspot and development trend of MOM

The above parts of this paper introduces the architecture of MOM and compares the major MOM products that are popular in the market from the basic functions and application scenarios. Fundamentally, MOM provides an effective and convenient communication service for the application.With the gradual migration of the Internet system to the cloud and the application of large-scale mobile terminals, this current status puts more demands on MOM.

A. Flexible and efficient distributed cluster structure

Distributed IT systems are becoming more common and many MOM products also provide distributed clustering capabilities. However, the existing MOM cluster structure does not apply to a wide variety of scenarios. For example, in the process of dealing with machine failure or expansion, Kafka requires an expensive and slow process to rebalance data in partitions, and it may lead to service unavailability. These features are clearly not suitable for nancial areas that are sensitive to data consistency. On the other hand, there is an important application scenario to ensure that MOM provides reliable services in the case of limited hardware resources. This is also what we are researching in this area.

B. Thread optimization based on multi-core technology

With the development of mobile internet, MOM faces an explosive increase in the number of service access points. How to support a large number of devices at the same time online and how to ensure two-way communication has become a new demand for MOM. However, existing MOM products are not as good as they are described in terms of concurrency. On the other hand, the application of multi-core CPU improves the concurrency performance of the server. In recent years, some teams have improved the performance of MOM products by using development languages(such as Erlang and go) with better concurrency performance. At the same time, the effort to refactor MOM in a new language is frustrating, and some teams want to get the same purpose by optimizing the thread structure. For example, the Apache ActiveMQ team is developing the next generation of MOM, i.e. Apache Apollo, through the optimization of thread structures to enhance MOM's scalability on multi-core and high-resource-configured servers.

C. A more transparent monitoring mechanism

The rise of data lineage technology improves the monitoring efficiency within the information system and facilitates the positioning of key information. More and more users want MOM to provide dynamic process tracking of message.

On the one hand is to monitor the flow of message within MOM and the status of MOM server nodes. Some MOM products provide a relatively complete service interface in terms of internal monitoring, such as ActiveMQ and RocketMQ. However, there are some MOM products that lack the necessary introspection methods and need to improve monitoring capabilities, such as RabbitMQ.

On the other hand, it is the dynamic information of message in the business process, the information that reflects the source of the message and the evolution of the situation. This is more like a message traceability mechanism that needs to be improved at the protocol.

D. Security mechanism in public cloud environment

In order to reduce costs, more and more enterprise users choose to use cloud services to provide information service infrastructure environment. When MOM is on a public cloud server, it is becoming increasingly important to protect the privacy of subscribers from being acquired by third parties. While protecting MOM performance, some groups try to use traditional or customized encryption matching technology to protect user privacy. However, the dynamic routing strategy used to improve the matching efficiency of MOM presents a new test for security problems. What kind of routing decision to protect the user's privacy, this is also an important issue.

V. Conclusion

As far as the existing message middleware products are concerned, there are still many problems to be solved. In MOM design, CAP theory still plays an important role. Most MOM systems look for a balance between C (Consistent), A (Availability), and P (Partition tolerance). In the era of big data, massive data and concurrent requests require distributed systems with better throughput and performance, and MOM designs pursue highly availability. In the business scenarios where data consistency is sensitive, such as smart transportation and industrial Internet of Things, MOM pursues a strong and consistent design. Due to the contradiction between availability and consistency, MOM need to design a more efficient routing algorithm and matching algorithm to improve the efficient forwarding of business data. On the other hand, research on redundant design and clustering strategies needs to be strengthened to improve overall performance while ensuring consistency and fault tolerance within the cluster. At the same time, the issues of data privacy and security in the distributed environment are also the key research directions. The overall security performance can be improved by the message format and proxy encryption in the MOM design.

Acknowledgment

This work is supported by the National Key R&D Program of China (Grant No. 2017YFC1405200).

References

[1] A. Banks and R. J. O. s. Gupta, "Mqtt version 3.1. 1," vol. 29, p. 89, 2014.

[2] P. A. Bernstein, "Middleware: a model for distributed system services," in *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.

[3] Y. Cao, N. Wang, G. Kamel, and Y.-J. Kim, "An electric vehicle charging management scheme based on publish/subscribe communication framework," in *IEEE Systems Journal*, vol. 11, no. 3, pp. 1822–1835, 2017.

[4] Chen Yuting, Mao Ting, Yu Bo, "A reliable messaging middleware for financial institutions," *Proceedings of the 3rd International Conference on Communication and Information Processing - ICCIP '17*, pp. 108–112, 2017.

[5] Chun Byonggon, Oh Beomseok, Cho Chihyun, Lee Dongyeob, "Design and Implementation of Lightweight Messaging Middleware for Edge Computing," in *Proceedings of the 6th International Conference on Control, Mechatronics and Automation - ICCMA 2018*, pp. 170–174, 2018.

[6] S. Davies and P. Broadhurst, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," in *Proceedings of the 6th International Conference on Control, Mechatronics and Automation*, pp. 170-174, 2018.

[7] M. Daz, C. Martn, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," in *Journal of Network and Computer Applications*, vol. 67, pp. 99–117, 2016.

[8] F. Ge, Z. Xinhua, and L. Chao, "Study of message data subscription based on multi-application big data analysis," in *Netinfo Security*, no. 11, pp. 44–49, 2017.

[9] T. Goovaerts, B. De Win, and W. Joosen, "A comparison of two approaches for achieving flexible and adaptive security middleware," in *Proceedings of the 2008 workshop on Middleware security*. ACM, Conference Proceedings, pp. 19–24, 2008

[10] W. He and L. D. Xu, "Integration of distributed enterprise applications: A survey," in *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 35–42, 2014.

[11] P. Hintjens, "ZeroMQ: messaging for many applications," " O'Reilly Media, Inc.", 2013.

[12] M. Hoefling, F. Heimgaertner, D. Fuchs, M. Menth, P. Romano, T. Tesfay, M. Paolone, J. Adolph, and V. Gronas, "Integration of ieee c37. 118 and publish/subscribe communication," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, Conference Proceedings, pp. 764–769, 2015

[13] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, Conference Proceedings, pp. 1–7, 2011

[14] I. Leontiadis, "Publish/subscribe notification middleware for vehicular networks," in *Proceedings of the 4th on Middleware doctoral symposium*. ACM, Conference Proceedings, p. 12, 2007

[15] Li Wei, Hu, Songlin, Li Jintao, Jacobsen Hans-Arno, "Community Clustering for Distributed Publish/Subscribe Systems," in *2012 IEEE International Conference on Cluster Computing*, pp. 81–89, 2012.

[16] M. Losciale, P. Boccadoro, G. Piro, G. Ribezzo, L. A. Grieco, and N. Blefari-Melazzi, "A novel icn-based communication bus for intelligent transportation systems," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, Conference Proceedings, pp. 1–6, 2018.

[17] J.-G. Ma, "Underlying techniques for large-scale distributed computing oriented publish/subscribe system," in *Journal of Software*, vol. 17, no. 1, 2006.

[18] C. N. Nguyen, J. Lee, S. Hwang, and J.-S. Kim, "On the role of message broker middleware for many-task computing on a?big-data platform," in *Cluster Computing*, 2018.

[19] M. Nkomo, G. P. Hancke, A. M. Abu-Mahfouz, S. Sinha, and A. J. Onumanyi, "Overlay virtualized wireless sensor networks for application in industrial internet of things: A review," in *Sensors (Basel)*, vol. 18, no. 10, 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/30249061

[20] "An open source distributed messaging and streaming data platform".

[21] J. R. Santana, M. Maggio, R. Di Bernardo, P. Sotres, and L. SNchez "On the use of information and infrastructure technologies for the smart city research in europe: A survey," in *IEICE Transactions on Communications*, vol. E101.B, no. 1, pp. 2–15, 2018.

[22] B. Snyder, D. Bosnanac, and R. Davies, "ActiveMQ in action," in Manning Greenwich Conn, vol. 47, 2011.

[23] P. Sommer, F. Schellroth, M. Fischer, and J. Schlechtendahl, "Message-oriented middleware for industrial production systems," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, Conference Proceedings, pp.1217–1223, 2018.

[24] TongTech, "Information technologySpecification for message-oriented middleware," p. 24, 2011.

[25] A. V. Ventrella, G. Piro, and L. A. Grieco, "Information-centric publish-subscribe mechanisms for intelligent transportation systems," in *2017 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, IEEE, Conference Proceedings, pp. 199–204, 2017.

[26] Y. Wang, C. J. C. E. Wang, and Software, "A design of reliable consumer based on kafka," 2016.

[27] X. Wei, "Research and design of subscription/publishing system based on message middleware," in *2011 International Conference on Electrical and Control Engineering*. IEEE, Conference Proceedings, pp. 3091–3094, 2011.

[28] X. W. XU Jing, "Summarization of message-oriented middleware," in *Computer Engineering*, vol. 31, no. 16, pp. 73–76, 2005.

[29] S. Zheng, Q. Zhang, R. Zheng, B. Q. Huang, Y. L. Song, and X. C. Chen, "Combining a multi-agent system and communication

middleware for smart home control: A universal control platform architecture," *Sensors (Basel)*, vol. 17, no. 9, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/28926957

[30] W. Zhenghe, W. Feng, D. Hui, L. Cuiyin, and Z. J. A. R. o. C. Xiaoli, "Optimized load balancing algorithm for kafka consumer/client," 2017.

[31] J. Bang, S. Son, H. Kim, Y. S. Moon, and M. J. Choi, "Design and implementation of a load shedding engine for solving starvation problems in apache kafka," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Conference Proceedings, pp. 1–4,2018.

[32] A. V. Uzunov, "A survey of security solutions for distributed publish/subscribe systems," in *Computers & Security*, vol. 61, pp.94–129, 2016.

[33] J. Wang and J. Bigham, "Anomaly detection in the case of message oriented middleware," in *Workshop on Middleware Security*, Conference Proceedings, 2008.

[34] C. Borcea, A. ? D. Gupta, Y. Polyakov, K. Rohloff, and G. Ryan, "Picador: End-to-end encrypted publish/subscribe information distribution with proxy re-encryption," in *Future Generation Computer Systems*, vol. 71, pp.177–191, 2017.

[35] G. Di Crescenzo, B. Coan, J. Schultz, S. Tsang, and R. N. Wright, "Privacy-preserving publish/subscribe: Efficient protocols in a distributed model," Springer, pp. 114–132, 2017.

[36] M. Nabeel, N. Shang, and E. Bertino, "Efficient privacy preserving content based publish subscribe systems," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, ACM, Conference Proceedings, pp. 133–144, 2012.

[37] E. Onica, P. Felber, H. Mercier, and E. Rivire, "Efficient key updates through subscription re-encryption for privacy-preserving publish/subscribe," in *Proceedings of the 16th Annual Middleware Conference*, ACM, Conference Proceedings, pp. 25–36, 2015.

[38] Kazemzadeh, Reza Sherafat, Jacobsen Hans-Arno, "Reliable and Highly Available Distributed Publish/Subscribe Service," in *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pp. 41–50, 2009.

[39] Kazemzadeh, Reza Sherafat, Jacobsen Hans-Arno, "Partition-Tolerant Distributed Publish/Subscribe Systems," in *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pp. 101–110, 2011.

[40] Jia Yue, Phillips, Chris, "Fast and Reliable IP Recovery for Overlay Routing in Mission Critical Message Oriented Middleware," in *2014 IEEE 17th International Conference on Computational Science and Engineering*, pp. 1577–1584, 2014.

[41] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, and J. Donham, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, ACM, Conference Proceedings, pp. 147–156, 2014.

[42] Wang Zhenghe, Dai Wei, Wang Feng, Deng Hui, Wei Shoulin, Zhang Xiaoli, Liang Bo, "Kafka and Its Using in High-throughput and Reliable Message Distribution," in *2015 8th Internetional Conference on Intelligent Networks and Intelligent Systems(ICINIS)*, pp. 117–120, 2015.

[43] A. Videla and J. Williams, "Rabbitmq in action," 2012.