Monitoring of application exchanges with an IoT system

**Abstract—**

**Keywords—message broker, data exchange**

Ⅰ. INTRODUCTION

ideas to mention: using several paralleling pub-sub model communication protocols to achieve monitoring, visualization and adaptation based on AMQP data exchange model and existing architecture.

Objectives:

Expected results:

Research problem: except AMQP, what about others

Distributed Message Brokers are typically used to decouple separate stages of a software architecture. They permit communication between these stages asynchronously, by using the publish-subscribe paradigm. Implementing a message-oriented middleware API (BL-MOM1) enables for asynchronous communication which promotes applications to be more loosely coupled. As a result, available resources could be better utilized and improve the system performance. These message brokers are also finding new applications in the domain of IoT devices and may also be used as method to implement an event-driven processing architecture.

Question:

What are the common requirements for the implementation of message queues?

What are the divergent functionalities in the current message queue offerings?

How do each of the implementations offer reliability, partitioning and fault tolerance?

How does each implementation messaging transfer in the general framework (Section Ⅳ)?

We use the current offerings of this paradigm architectural protocols that analyze the data to implement the exchanges, namely the AMQP broker, MQTT broker, KAFKA broker and the CoAP server, to collect meta-information so that:

•Explore the log services offered by the exchange infrastructures, •Improve security of the architecture components and the exchange data, • Keep track of the exchanged messages, •Enhance the maintainability: detection of exceptions (such as problem of transfer of a message), precision of the context and the origin of the problem, alerts and notifications.

The rest of this paper is structured as follows:

• Section Ⅱ: Related work

• Section Ⅲ: Defines messaging related terms used in this paper

• Section Ⅳ: Messaging exchanges architecture and overview

• Section Ⅴ: Implementation elements

•Section Ⅵ: depicts results illustrated by a case study

• Section Ⅶ: Conclusion and future work

• Section Ⅷ: References

Ⅱ. RELATED WORK

1. *Message Oriented Middleware*

Message Oriented Middleware (MOM) is a system that transports messages between two or more different clients on a network so that they can be routed and delivered. MOM’s message communication consists of a simple operation such as sending or receiving a message, and is generally used in an email system, a chat system, and the like. The client calls the API through the MOM system and sends a message to the registered object. The client that sent the message does not need to be involved in the message transfer anymore and can perform other tasks. In addition, even if a network failure occurs, the received message can be processed. If you are using MOM, you can add management interfaces to monitor and extend performance. Therefore, the client can escape problems related to message transmission.

Metamodel based on AMQP

Moose

RabbitMQ

Mosquito

LibCoap

Kafka

Ⅲ. TECHNICAL TERMINOLOGY

These terms have special significance within the context of this paper related to RabbitMQ implementation and message exchange of the protocols.

* *Cluster*: the collection and logical grouping of nodes that running the application and sharing users, virtual hosts, queues, exchanges, bindings, and runtime parameters.
* *Virtual host*: designed for multi-tenancy and security reasons and used to separate permissions of different users
* *User*: a client application that pub/sub messages to exchange/from queues
* *Connection*: a TCP network connection between users and the broker
* *Channel*: a stream of communications which a lightweight Connection to reduce the overhead of the operating system in establishing a TCP Connection
* *Exchange*: a named entity that match the Routing key in the query table to dispatch messages to the queue according to the distribution rules
* *Binding*: a virtual connection between exchange and queue, Routing key can be included in binding
* *Queue*: a named entity as the container of messages and delivers them to consumers
* *Routing key*: a message attribute that the exchange may use when deciding how to route the message to queues
* *Message*: the application data carried across the network for the application and is composed of a header and a body
* *QoS*: the level of assurance for delivery of an application message
* *Security*: a sequence of operations that ensure protection of data and provides secure delivery of data between two parties
* *Session*: a stateful interaction between a Client and a Server
* *Topic*: a category or feed name to which messages are stored and published
* *Partition*: a part of topic which allow to parallelize a topic by splitting the data in a particular topic across multiple brokers
* *Offset*: a sequential id number that uniquely identifies each message within the partition
* *Consumer group*: a group of consumers acting as a single logical unit
* *Zookeeper*: a distributed leader to manage cluster state
* *Token*: a means of matching a response with a request which is intended for use as a client-local identifier for differentiating between concurrent requests
* *Options*: a set of properties used in both requests and responses in CoAP protocol
* *Request methods*: a means of request behavior that include GET, POST, PUT and DELETE

Ⅳ. MESSAGE EXCHANGE SYSTEM

To explore communication rules allowing to operate connected objects and to reach data exchanges monitoring, visualization and adaptation through AMQP, MQTT, KAFKA and CoAP protocols, the general framework is demonstrated in Figure 1, which represents the collection of data from different protocols related to the exchange of messages and the log of events.

In the general framework, using Moose technologies to build parsers for analyzing and transforming data that receives from the log services offered by the exchange infrastructures in different protocols, and a messaging metamodel is extended by all these aspects in order to implement data exchanges.

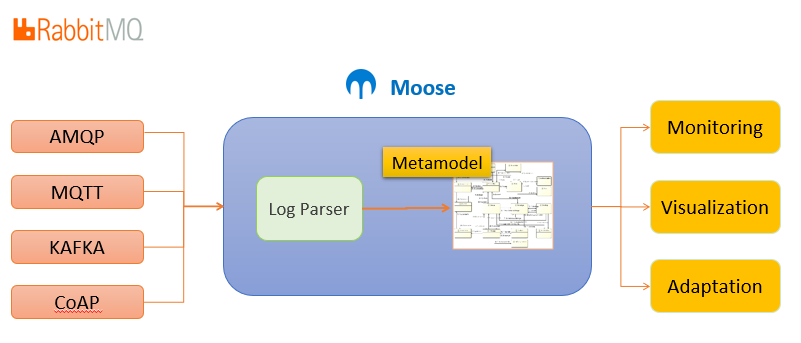


Fig. 1. The General Framework

This section will analyze how the protocols working in the message system aspects and how the architectures elements can be achieved, and compare the similarity and difference among protocols. The last section will define and extend the messaging metamodel of data exchanges which covering aspects and functionalities related to the exchanges.

*A.* *Messaging Analysis Architecture*

AMQP: a well-defined MOM protocol and there are various platforms that implement it. Among them, RabbitMQ is a message broker implemented based on AMQP. It is easy to configure the cluster and Web Management UI is provided. In addition, plugins such as MQTT Convert and CoAP are also provided, making it easy to connect to various platforms with excellent scalability.

It is an asynchronous message queuing protocol, aiming to create an open standard for passing messages between applications and systems regardless of internal design. It was initially designed for financial transaction processing systems, such as trading and banking systems, which require high guarantees of reliability, scalability, and manageability. This use-case greatly influences the design of AMQP.

Figure 2 shows the architecture of AMQP. Instead of sending messages to queues directly, producers send messages to exchanges. Exchanges then send messages to queues whose binding key matches the messages’ routing key. The queue, in turn, sends messages to consumers who are subscribed to it.

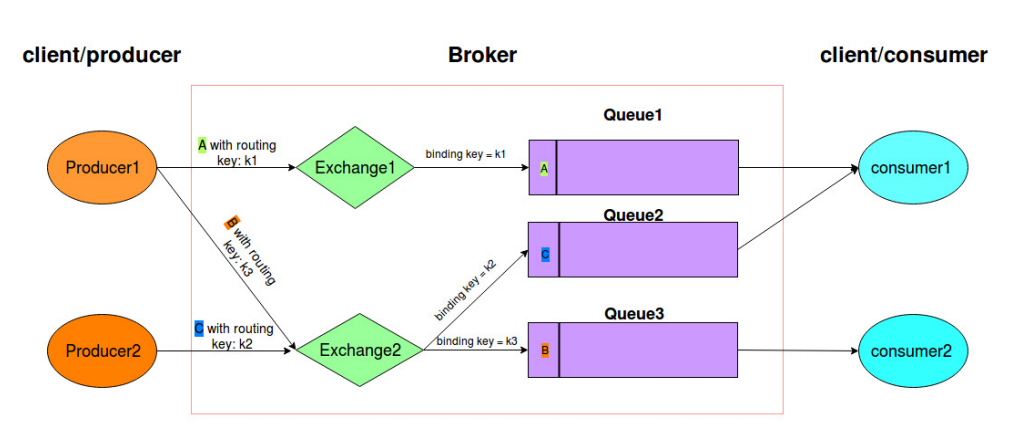


Fig. 2 AMQP Architecture

MQTT: An MQTT system consists of clients communicating with a server. A client may be either a publisher of information or a subscriber. Each client can connect to the broker. Information is organized in a hierarchy of topics.If a broker receives a topic for which there are no current subscribers, it will discard the topic unless the publisher indicates that the topic is to be retained. When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker. Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics.

KAFKA:

Kafka was developed at LinkedIn and primarily used for log processing. The architecture of Kafka has been optimized to stream data between systems and applications as fast as possible in a scalable manner. Kafka can be used either for online or offline log processing.The fact that reliability of message reception is traded off to some extent implies the loss of a single record among a multitude is not a huge deal-breaker. The rationale behind this is, for log aggregated data, delivery guarantees are unnecessary.

The general architecture for Kafka is illustrated in Figure 3. Data is divided into topics, which resemble streams of messages. Topics are further divided into partitions and each broker can possess one or more such partitions. Producers publish to topics and brokers store messages received from the producers. Messages are payloads of bytes that consumers use iterators to consume. Kafka employs the pull model for consumer message receipt, which permits consumers to consume messages at their own pace.

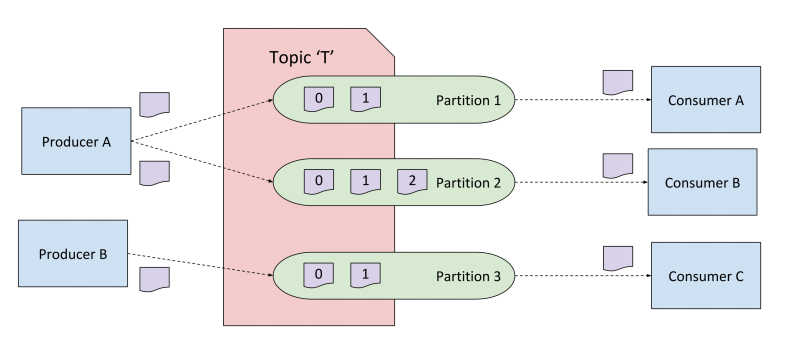


Fig. 3 Kafka Architecture

CoAP: provides a request/response interaction model between application endpoints, the work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes and networks. One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation, and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP, but rather to realize a subset of REST common with HTTP but optimized for M2M applications.

*B. Similarity and Divergence among protocols*

Table 1 analysis

*C. Messaging Metamodel*

Define a metamodel covering aspects and functionalities related to the exchanges and architectures put in place.

Figure 4 Metamodel extended

*D. Log Service*

This service subscribes the publish-subscribe channel and gets all the published messages on the channel. It is responsible for logging all submitted tasks.

Table 1. Messaging Analysis Architecture for Different Protocols

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TERMS | AMQP | MQTT | KAFKA | COAP |
| Cluster | × | × | × | × |
| V-host | × | × | × | × |
| User | × | × | × | × |
| Connection | × | × | × | × |
| Channel | × | × | × | × |
| Exchange | × | × |  | × |
| Binding | × | × |  | × |
| Queue | × | × |  | × |
| Topic |  |  | × |  |
| Partition |  |  | × |  |
| Routing Keys | × | × |  | × |
| Message | × | × | × | × |
| QoS |  | × | × | × |
| Security |  | × | × | × |
| Session |  | × |  |  |
| Zookeeper |  |  | × |  |
| Token |  |  |  | × |
| Options |  |  |  | × |
| Request Methods |  |  |  | × |

Ⅴ. IMPLEMENTATION

Ⅵ. CASE STUDY

Ⅶ. CONCLUTION AND FUTURE WORK

Ⅷ.

REFERENCES